

# **CPE 408330**

## **Assembly Language and Microprocessors**

### **Chapter 5: 8088/8086 Microprocessor Programming – Integer Instructions and Computations**

[Computer Engineering Department,  
Hashemite University]

# Lecture Outline

- ▶ 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*

- ▶ The data-transfer functions provide the ability to move data either between its **internal registers** or between an internal register and a storage location in **memory**.
- ▶ The data-transfer functions include
  - MOV (Move byte or word)
  - XCHG (Exchange byte or word)
  - XLAT (Translate byte)
  - LEA (Load effective address)
  - LDS (Load data segment)
  - LES (Load extra segment)

# 5.1 Data-Transfer Instructions - Move Instruction

## ► The MOVE Instruction

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)

Allowed operands for MOV 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)

# *5.1 Data-Transfer Instructions -*

## **Move Instruction**

- ▶ The MOVE Instruction

e.g. `MOV DX, CS`  
`MOV [SUM], AX`

- ▶ Note that the MOV instruction cannot transfer data directly between external memory.

# 5.1 Data-Transfer Instructions - Move Instruction

MOV DX, CS

Source = CS → word data

Destination = DX → word data

Operation: (CS) → (DX)

• State before fetch and execution

CS:IP = 0100:0100 = 01100H

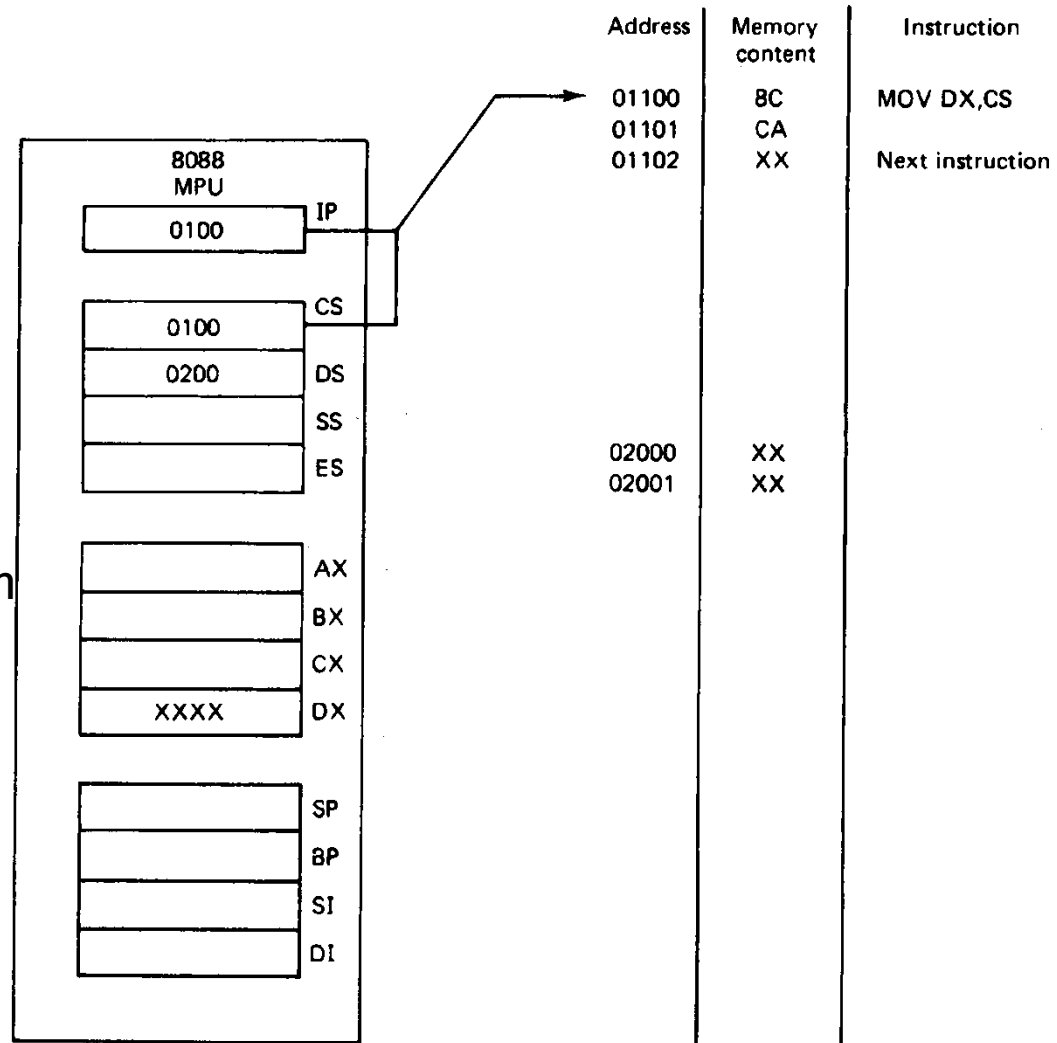
Move instruction code = 8CCA H

(01100H) = 8CH

(01101H) = CAH

(CS) = 0100H

(DX) = XXXX → don't care state



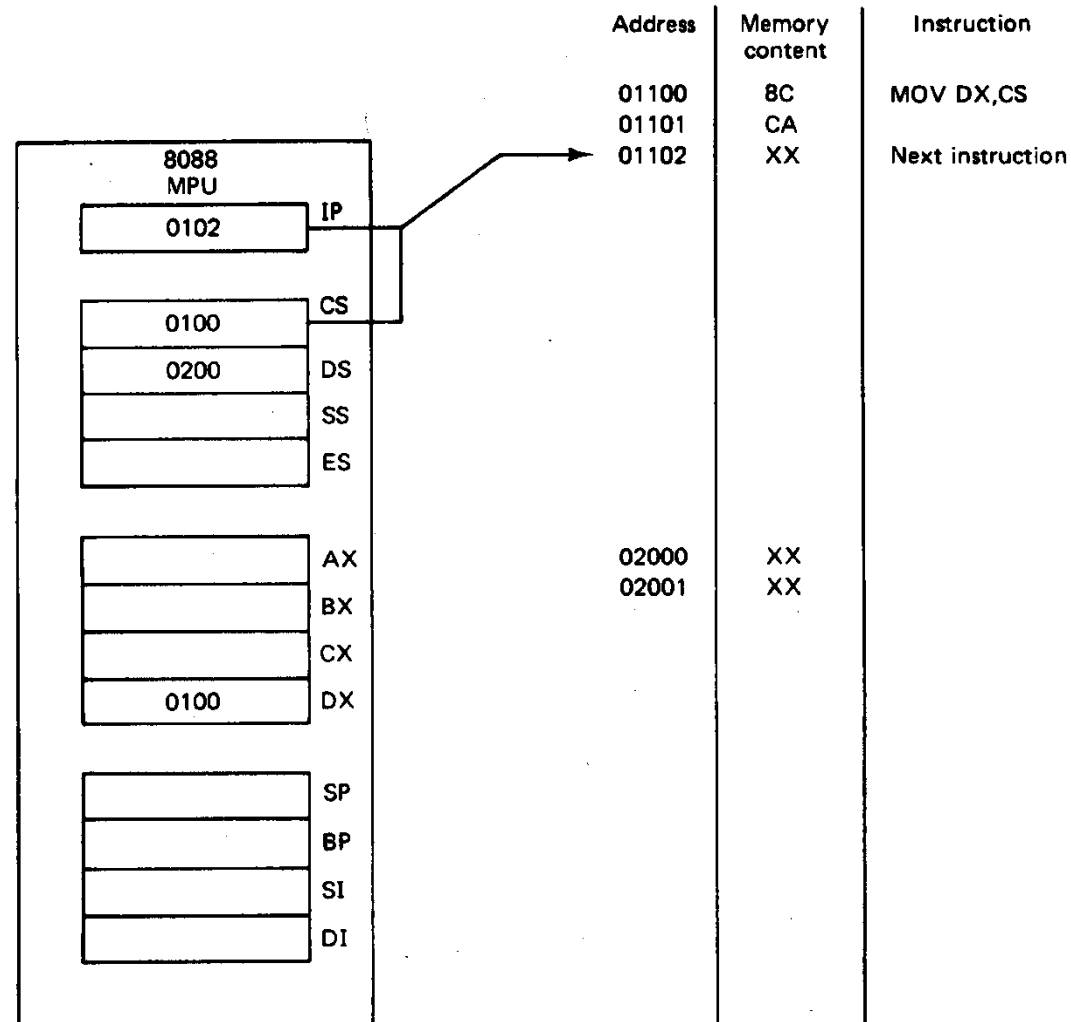
Before execution

(c)

# 5.1 Data-Transfer Instructions - Move Instruction

MOV DX, CS

- 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



(d)

After execution

# 5.1 Data-Transfer Instructions - Move Instruction

## ▶ EXAMPLE

What is the effect of executing the instruction  
`MOV CX, [SOURCE_MEM]`

Where `SOURCE_MEM` equal to  $20_{16}$  is a memory location offset relative to the current data segment starting at  $1A00_{16}$ .

## ▶ Solution:

$$\begin{aligned} ((DS)0 + 20_{16}) &\rightarrow (CL) \\ ((DS)0 + 20_{16} + 1_{16}) &\rightarrow (CH) \end{aligned}$$

Therefore CL is loaded with the contents held at memory address

$$1A000_{16} + 20_{16} = 1A020_{16}$$

and CH is loaded with the contents of memory address

$$1A000_{16} + 20_{16} + 1_{16} = 1A021_{16}$$



# 5.1 Data-Transfer Instructions - Move Instruction

## ▶ EXAMPLE

Use the DEBUG to verify

MOV CX,[20]

DS = 1A00, (DS:20) = AA55H

(1A00:20) → (CX)

## ▶ Solution:

```
CONSOLE MODE - debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B35 ES=0B35 SS=0B35 CS=0B35 IP=0100 NV UP EI PL NZ NA PO NC
0B35:0100 E375 JCXZ 0177
-a
0B35:0100 MOV CX, [20]
0B35:0104
-R DS
DS 0B35
:1A00
-E 20 55 AA
-T
AX=0000 BX=0000 CX=AA55 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1A00 ES=0B35 SS=0B35 CS=0B35 IP=0104 NV UP EI PL NZ NA PO NC
0B35:0104 A0D396 MOV AL, [96D3] DS:96D3=00
-
```

# 5.1 Data-Transfer Instructions -

## Move Instruction

- 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

DS,ES to 2000H

SS to 3000H

AX, BX to 0H

CX to 0AH

SI to 200

DI to 300

**MOV AX,2000H**

**MOV DS,AX**

**MOV ES,AX**

**MOV AX,3000H**

**MOV SS,AX**

**MOV AX,0H**

**MOV BX,AX**

**MOV CX,0AH**

**MOV DX,100H**

**MOV SI,200H**

**MOV DI,300H**

# 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)

Allowed operands for XCHG 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

XCHG [SUM],BX

Note: SUM = 1234

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

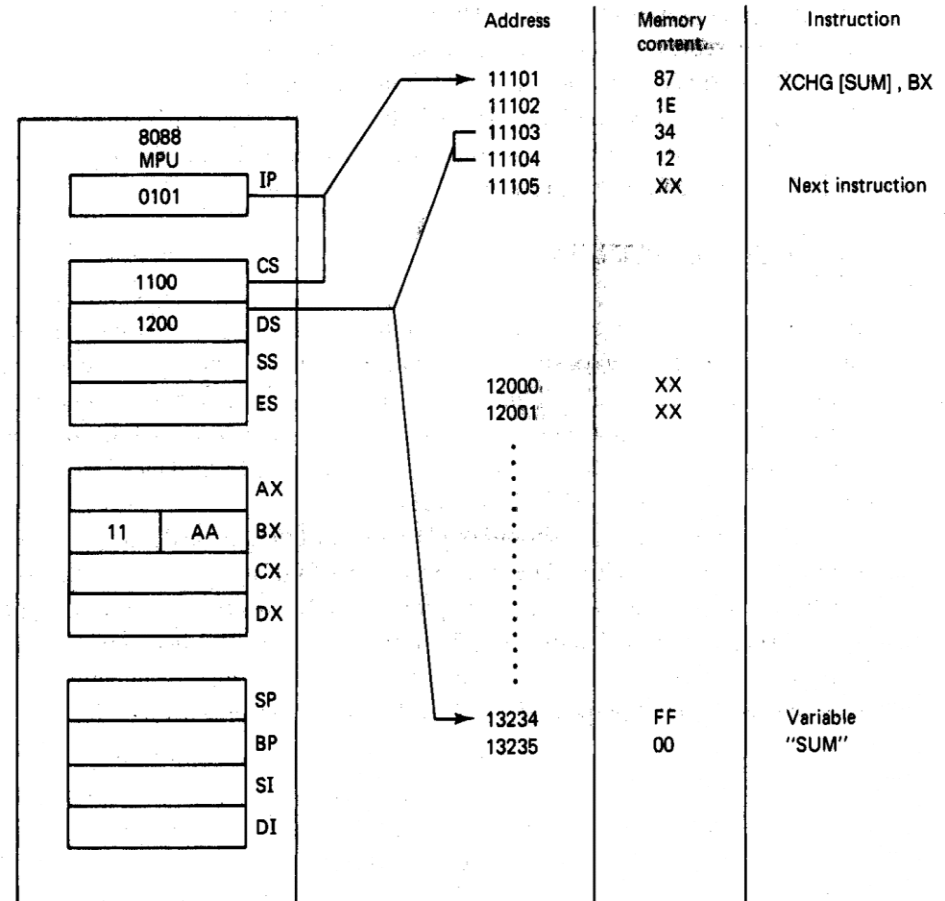
Move instruction code = 871E3412H

(01104H,01103H) = 1234H = SUM

(DS) = 1200H

(BX) = 11AA

(DS:SUM) = (1200:1234) = 00FFH



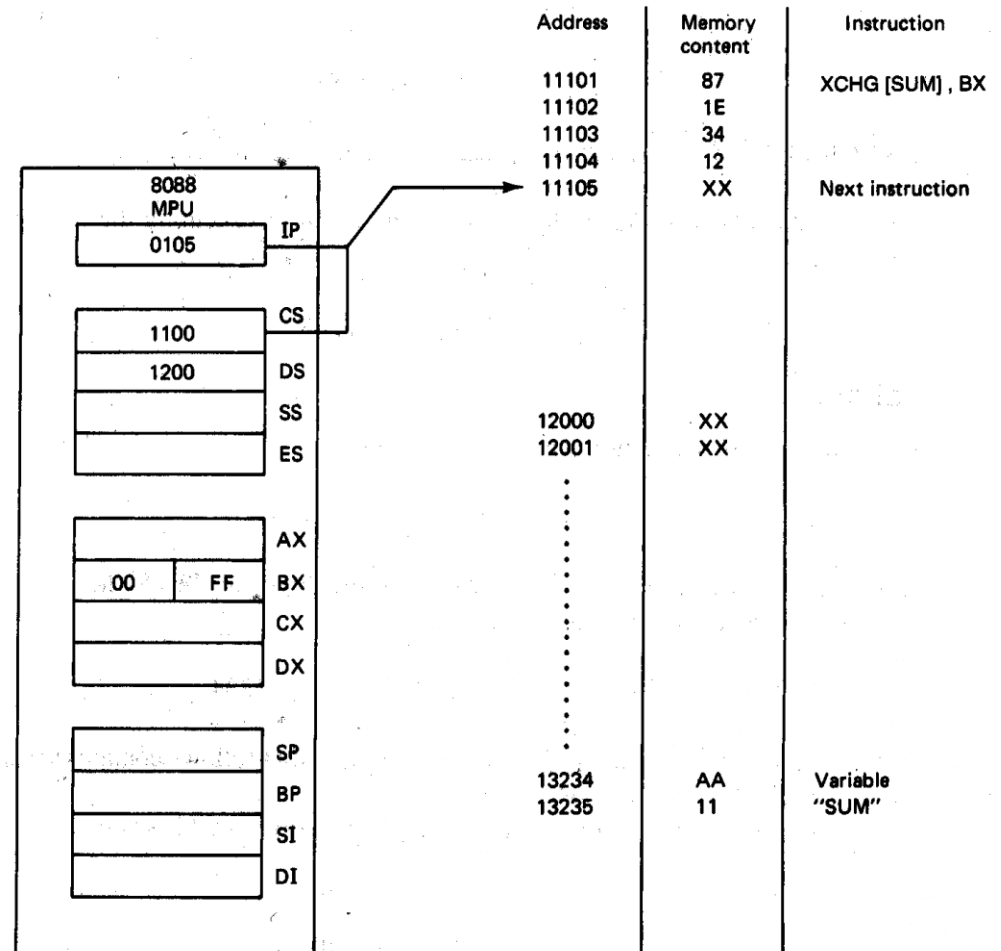
(c)

Before execution

# 5.1 Data-Transfer Instructions – Exchange Instruction

## XCHG [SUM],BX

- 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



(d)

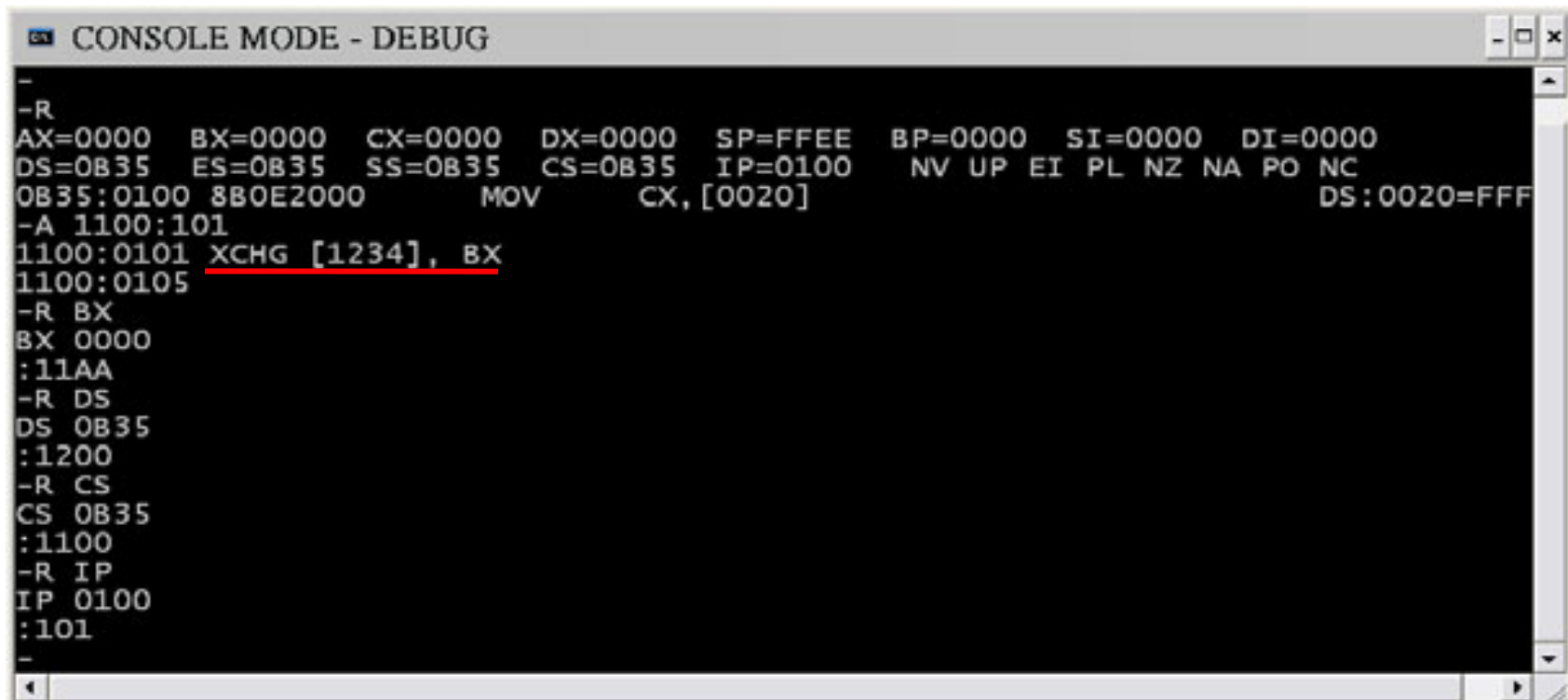
After execution

# 5.1 Data-Transfer Instructions - Exchange Instruction

## ▶ EXAMPLE

Use the DEBUG to verify the previous example.

## ▶ Solution:



```
CONSOLE MODE - DEBUG
-
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B35  ES=0B35  SS=0B35  CS=0B35  IP=0100  NV UP EI PL NZ NA PO NC
0B35:0100 8B0E2000      MOV     CX,[0020]          DS:0020=FFF
-A 1100:101
1100:0101 XCHG [1234], BX
1100:0105
-R BX
BX 0000
:11AA
-R DS
DS 0B35
:1200
-R CS
CS 0B35
:1100
-R IP
IP 0100
:101
-
```

# 5.1 Data-Transfer Instructions - Exchange Instruction

## ► Solution (cont'd):

```
CONSOLE MODE
-
-R
AX=0000 BX=11AA CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1200 ES=0B35 SS=0B35 CS=1100 IP=0101 NV UP EI PL NZ NA PO NC
1100:0101 871E3412 XCHG BX,[1234] DS:1234=11AA
-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=0B35 SS=0B35 CS=1100 IP=0105 NV UP EI PL NZ NA PO NC
1100:0105 0000 ADD [BX+SI],AL DS:00FF=00
-D 1234 1235
1200:1230 AA 11 ..
-Q
C:\>
```

# 5.1 Data-Transfer Instructions - Translate Instruction

## ► The XLAT Instruction

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	$((AL)+(BX)+(DS)0) \rightarrow (AL)$	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

$((AL)+(BX) + (DS)0) \rightarrow (AL)$

Where:

(DS)0 = Points to the active data segment

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

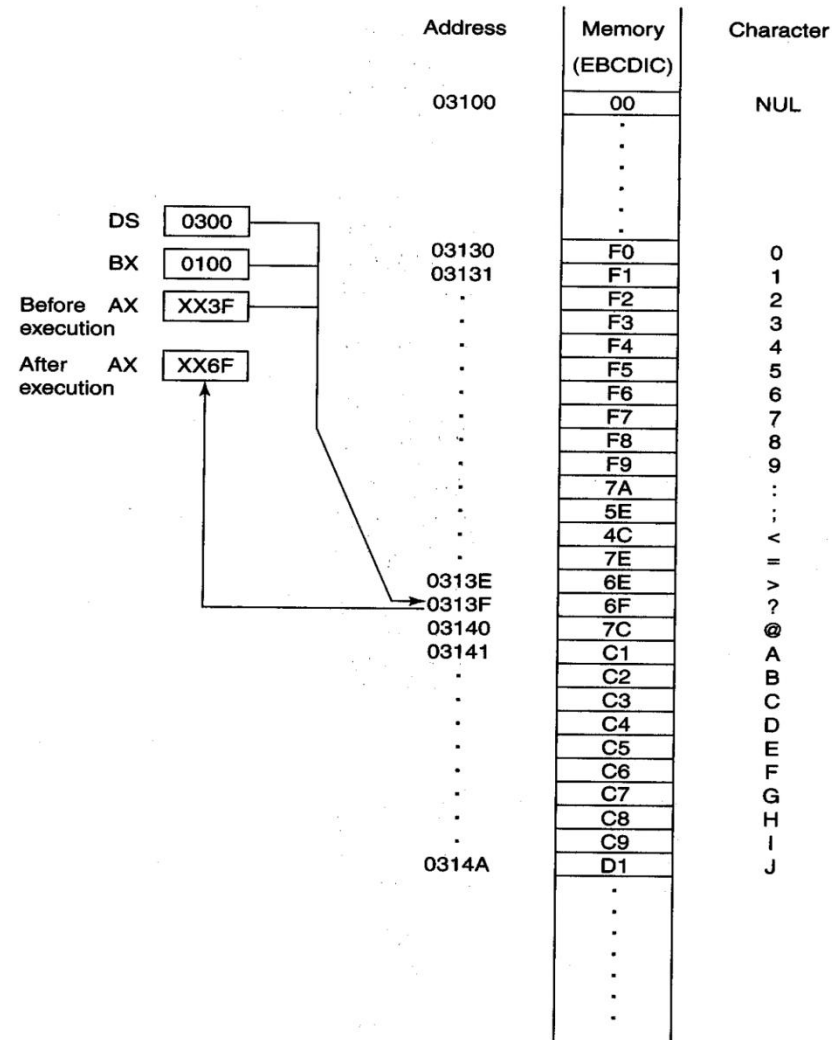
(AL) = Displacement to the element of the table that is to be accessed\*

\*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 (41H+(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

- The LEA, LDS, and LES 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

# 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 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

## • 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

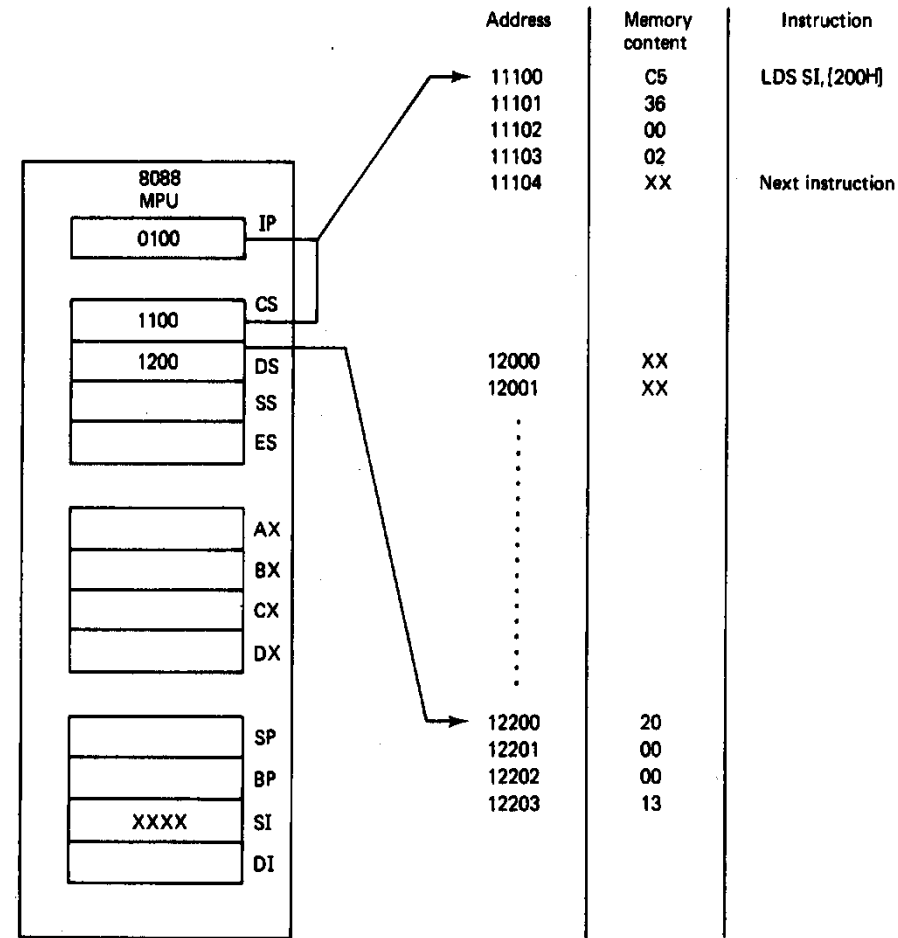
(11102H,11103H) = (EA) = 0200H

(DS) = 1200H

(SI) = XXXX → don't care state

(DS:EA) = 12200H = 0020H = Offset

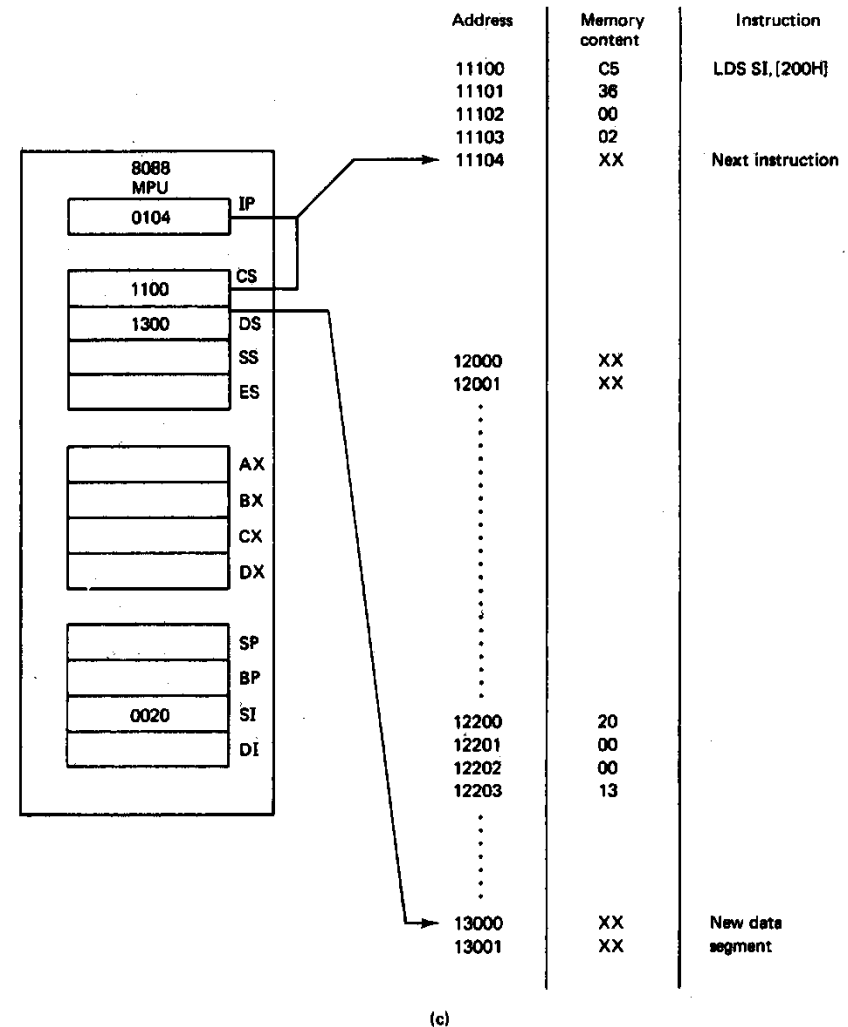
(DS:EA+2) = 12202H = 1300H = SBA



Before execution

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

- Example
- State after execution  
 $CS:IP = 1100:0104 = 11104H$   
 $01004H \rightarrow$  points to next sequential instruction  
 $(DS) = 1300H \rightarrow$  defines a new data segment  
 $(SI) = 0020H \rightarrow$  defines new offset into DS



After execution

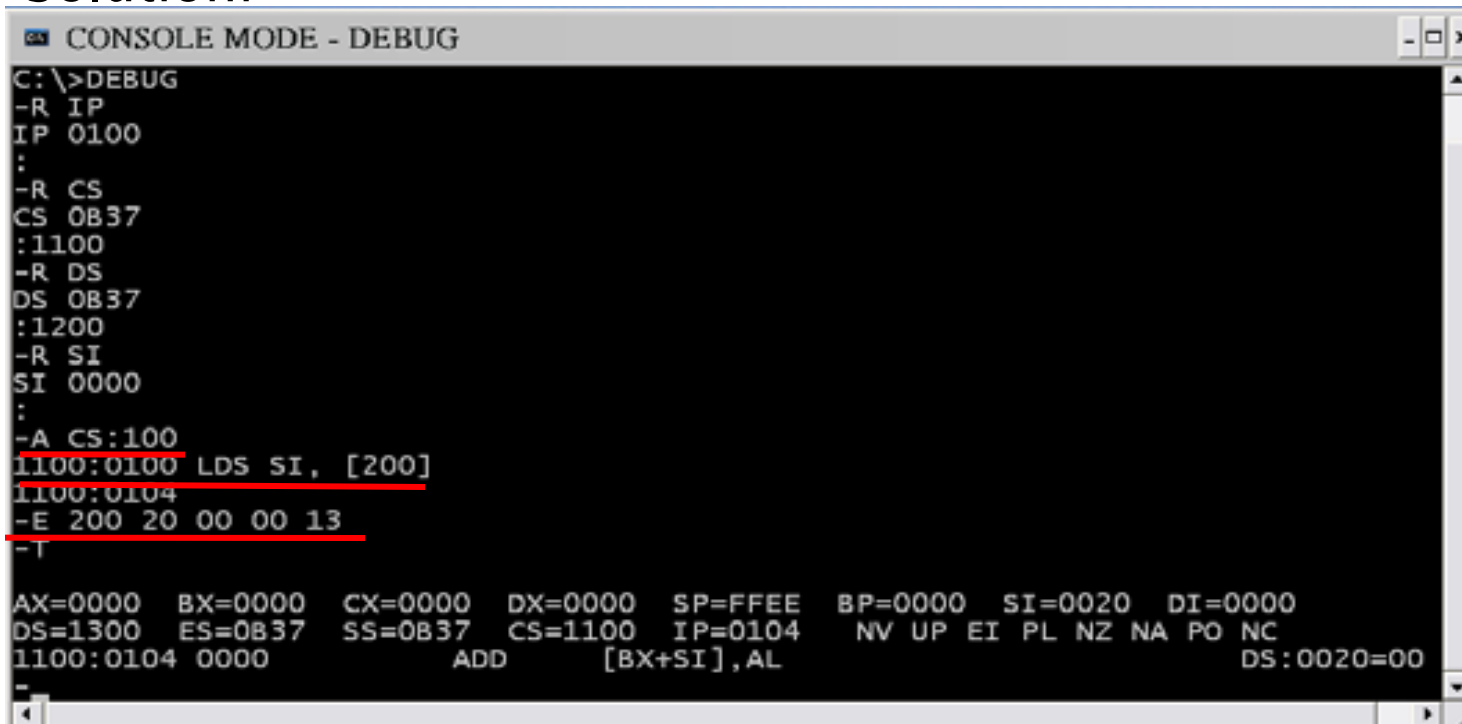
# 5.1 Data-Transfer Instructions - Load Effective Address and Load Full Pointer Instructions

## ▶ EXAMPLE

Verify the following instruction using DEBUG program.

LDS SI, [200H]

## ▶ Solution:



```
GA  CONSOLE MODE - DEBUG
C:\>DEBUG
-R IP
IP 0100
:
-R CS
CS 0B37
:1100
-R DS
DS 0B37
:1200
-R SI
SI 0000
:
-A CS:100
1100:0100 LDS SI, [200]
1100:0104
-E 200 20 00 00 13
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0020  DI=0000
DS=1300  ES=0B37  SS=0B37  CS=1100  IP=0104  NV UP EI PL NZ NA PO NC
1100:0104 0000          ADD     [BX+SI],AL          DS:0020=00
```

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

## ▶ EXAMPLE

Initializing the internal registers of the 8088 from a table  
in memory.

### ▶ Solution:

- 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

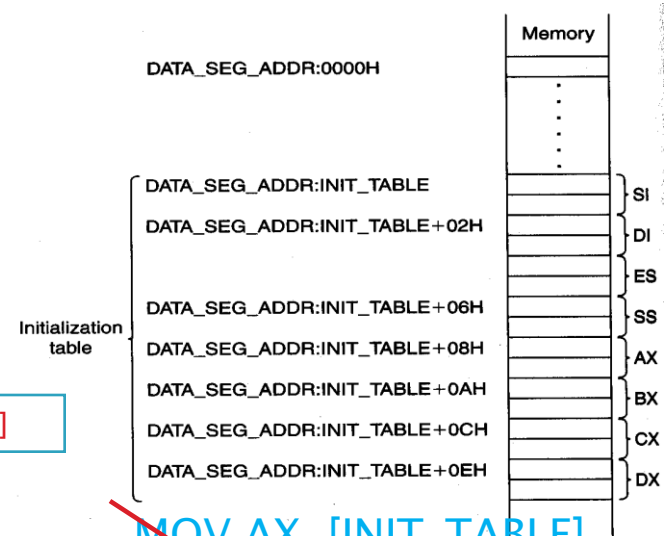
(INIT\_TABLE+8, INIT\_TABLE+9) → AX

(INIT\_TABLE+A, INIT\_TABLE+B) → BX

(INIT\_TABLE+C, INIT\_TABLE+D) → CX

(INIT\_TABLE+E, INIT\_TABLE+F) → DX

LES DI, [INIT\_TABLE+2]



~~MOV AX, [INIT\_TABLE]  
MOV SS, AX  
LDS SI, [INIT\_TABLE+02H]  
LES DI, [INIT\_TABLE+06H]  
MOV AX, [INIT\_TABLE+0AH]  
MOV BX, [INIT\_TABLE+0CH]  
MOV CX, [INIT\_TABLE+0EH]  
MOV DX, [INIT\_TABLE+10H]~~

## *5.2 Arithmetic Instructions*

- ▶ The arithmetic instructions include
  - Addition
  - Subtraction
  - Multiplication
  - Division
- ▶ Data formats
  - Unsigned binary bytes
  - Signed binary bytes
  - Unsigned binary words
  - Signed binary words
  - Unpacked decimal bytes
  - Packed decimal bytes
  - ASCII numbers

**BCD and ASCII Arithmetic: The microprocessor allows arithmetic manipulation of both BCD (Binary Coded Decimal) and ASCII data.**



# ASCII and BCD

## Processing ASCII Numbers

- 8086 provides four instructions
  - aaa** – ASCII adjust after addition
  - aas** – ASCII adjust after subtraction
  - aam** – ASCII adjust after multiplication
  - aad** – ASCII adjust before division
- \* These instructions do not take any operands
  - » Operand is assumed to be in AL

### ASCII representation

- \* Numbers are stored as a string of ASCII characters
    - » Example: 1234 is stored as 31 32 33 34H
- ASCII Code for 0,1, ...,9: 30H, 31H, ..., 39H

### BCD representation

- \* Unpacked BCD
  - » Example: 1234 is stored as 01 02 03 04H
    - Additional byte is used for sign
    - Sign byte: 00H for + and 80H for –
- \* Packed BCD
  - » Saves space by packing two digits into a byte
    - Example: 1234 is stored as 12 34H

## Processing Packed BCD Numbers

- Two instructions to process packed BCD numbers
  - daa** – Decimal adjust after addition
    - Used after **add** or **adc** instruction
  - das** – Decimal adjust after subtraction
    - Used after **sub** or **sbb** instruction
- \* No support for multiplication or division
  - » For these operations
    - Unpack the numbers
    - Perform the operation
    - Repack them

Note: ASCII = Unpacked + 30H

# ASCCI Adjustment Instructions

- ▶ Arithmetic operations are performed on numbers expressed in ASCII format , but we want the final result in decimal. (this saves conversions!)
  - Result must be in AL
- ▶ After the arithmetic operation, an adjustment must be performed on the result to convert it to the equivalent decimal result.
- ▶ This is main principle for all ASCII adjust operations.

# 5.2 Arithmetic Instructions – Addition

## Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	$(S) + (D) \rightarrow (D)$ Carry $\rightarrow$ (CF)	OF, SF, ZF, AF, PF, CF
ADC	Add with carry	ADC D, S	$(S) + (D) + (CF) \rightarrow (D)$ Carry $\rightarrow$ (CF)	OF, SF, ZF, AF, PF, CF
INC	Increment by 1	INC D	$(D) + 1 \rightarrow (D)$	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)

(a) Addition Instructions. (b) Allowed operands for ADD and ADC (c) Allowed operands for INC

- ▶ Variety of arithmetic instruction provided to support integer addition—core instructions are

- ADD  $\rightarrow$  Addition
- ADC  $\rightarrow$  Add with carry
- INC  $\rightarrow$  Increment

- ▶ Addition Instruction—**ADD**

- ADD format and operation:

**ADD D,S**

**$(S) + (D) \rightarrow (D)$**

- Add values in two registers

**ADD AX,BX**

**$(AX) + (BX) \rightarrow (AX)$**

- Add a value in memory and a value in a register

**ADD [DI],AX**

**$(DS:DI) + (AX) \rightarrow (DS:DI)$**

- Add an immediate operand to a value in a register or memory

**ADD AX,100H**

**$(AX) + IMM16 \rightarrow (AX)$**

- ▶ **Flags** updated based on result

- CF, OF, SF, ZF, AF, PF

## 5.2 Arithmetic Instructions – Addition Instructions

### ▶ EXAMPLE

Assume that the AX and BX registers contain  $1100_{16}$  and  $0ABC_{16}$ , respectively. What is the result of executing the instruction `ADD AX, BX`?

### ▶ Solution:

$$(BX) + (AX) = 0ABC_{16} + 1100_{16} = 1BBC_{16}$$

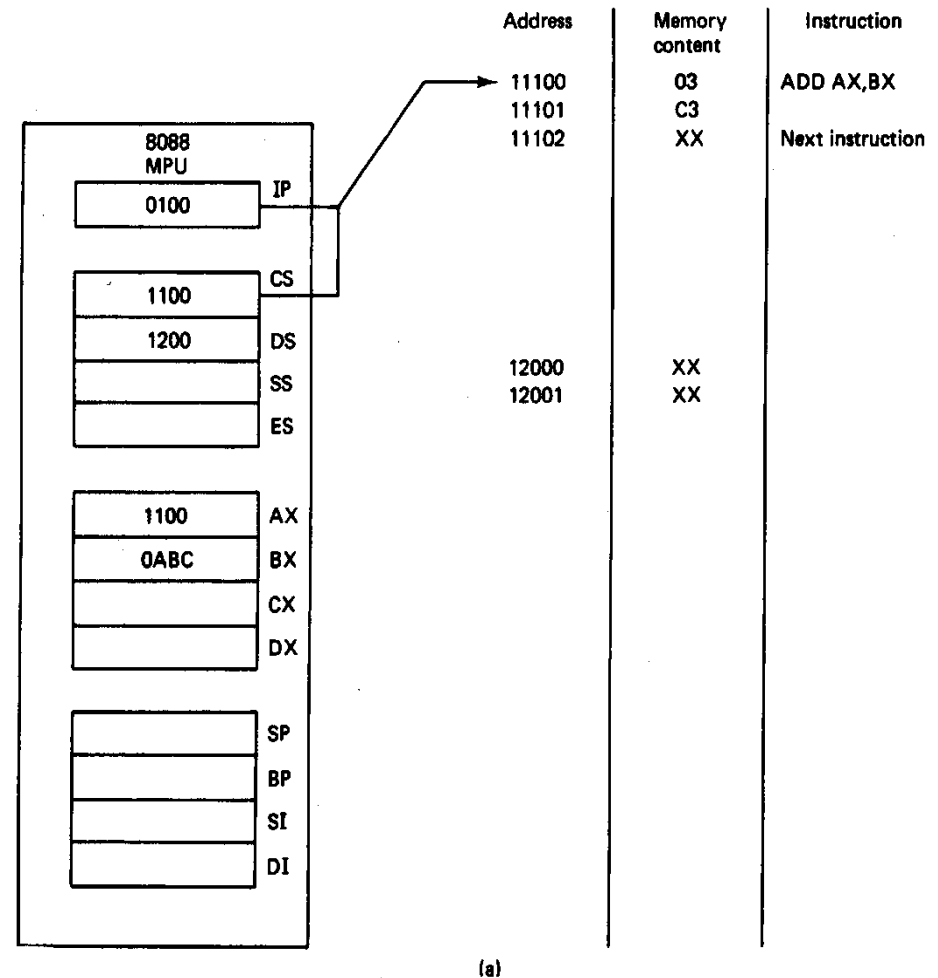
The sum ends up in destination register AX.  
That is

$$(AX) = 1BBC_{16}$$
$$CF = 0$$

# 5.2 Arithmetic Instructions – Addition

## Instructions

- State before fetch and execution
- CS:IP = 1100:0100 = 11100H  
 ADD machine code = 03C3H  
 (AX) = 1100H  
 (BX) = 0ABCH  
 (DS) = 1200H  
 (1200:0000) = 12000H = XXXX



Before execution

# 5.2 Arithmetic Instructions – Addition

## Instructions

- 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<sub>2</sub>

(BX) = unchanged

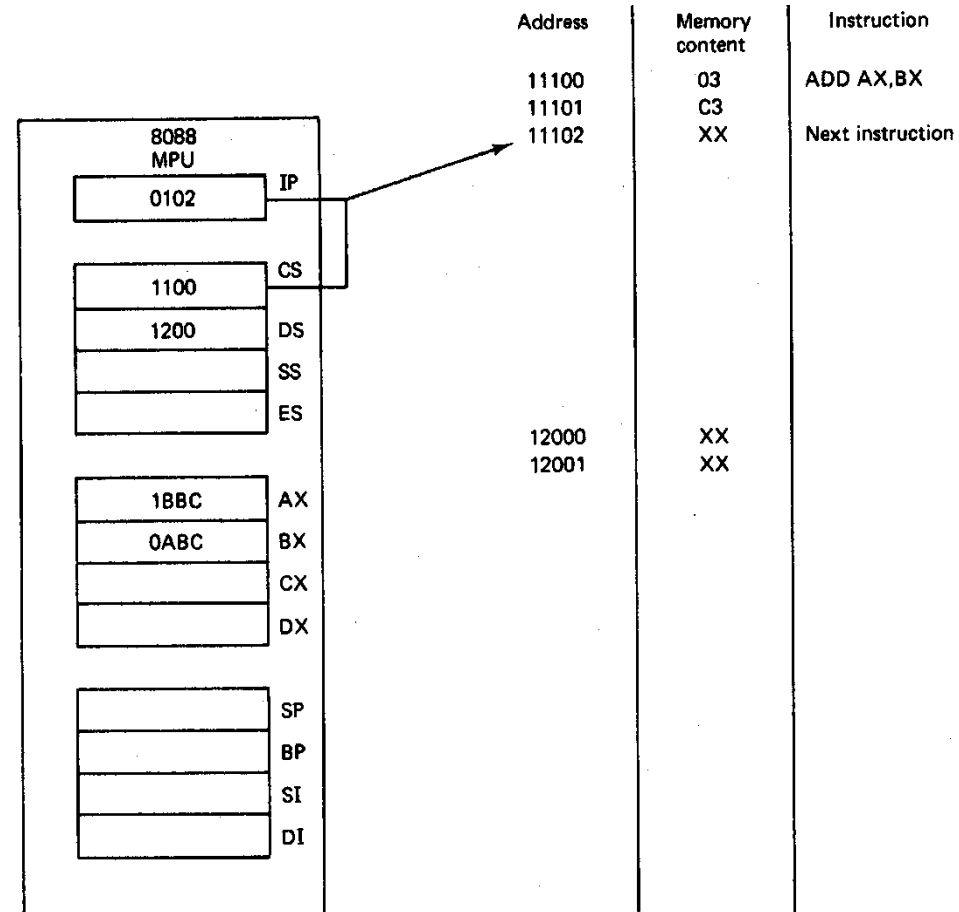
- Impact on flags

- 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**



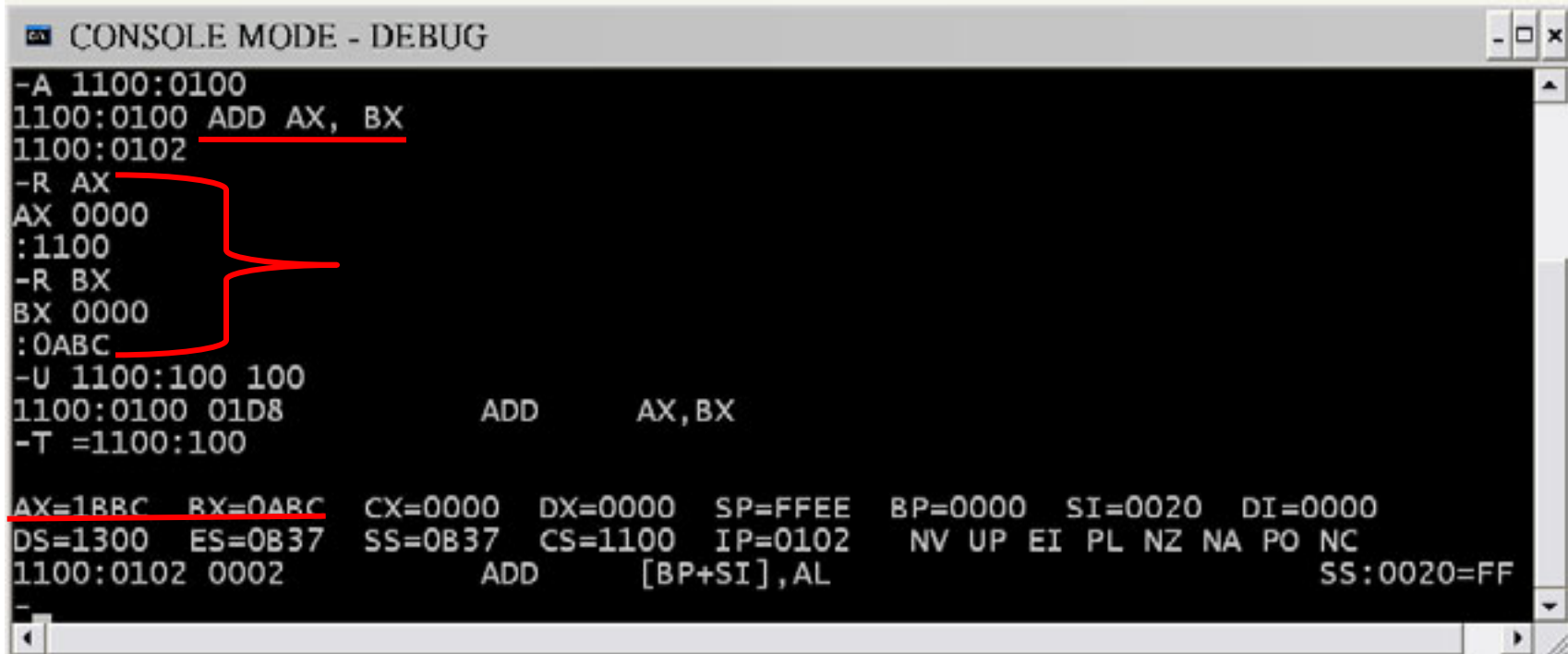
After execution

## 5.2 Arithmetic Instructions – Addition Instructions

### ▶ EXAMPLE

Verify the previous example using DEBUG program.

Solution:



```
CONSOLE MODE - DEBUG
-A 1100:0100
1100:0100 ADD AX, BX
1100:0102
-R AX
AX 0000
:1100
-R BX
BX 0000
:0ABC
-U 1100:100 100
1100:0100 01D8          ADD      AX,BX
-T =1100:100

AX=18BC  BX=0ABC  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0020  DI=0000
DS=1300  ES=0B37  SS=0B37  CS=1100  IP=0102  NV UP EI PL NZ NA PO NC
1100:0102 0002          ADD      [BP+SI],AL      SS:0020=FF
```

The screenshot shows the DEBUG console window. The title bar is "CONSOLE MODE - DEBUG". The command prompt shows the assembly of an instruction at address 1100:0100: `-A 1100:0100` followed by `1100:0100 ADD AX, BX`. The instruction is underlined. The next command is `1100:0102`. Then, the registers are displayed: `-R AX` shows `AX 0000` and `:1100`; `-R BX` shows `BX 0000` and `:0ABC`. A red bracket groups these two register displays. The next command is `-U 1100:100 100`, which disassembles the instruction at 1100:0100, showing `1100:0100 01D8 ADD AX,BX`. The final command is `-T =1100:100`, which shows the current state of the registers and the instruction pointer. The register values are: `AX=18BC`, `BX=0ABC` (underlined), `CX=0000`, `DX=0000`, `SP=FFEE`, `BP=0000`, `SI=0020`, `DI=0000`, `DS=1300`, `ES=0B37`, `SS=0B37`, `CS=1100`, `IP=0102`, and status flags `NV UP EI PL NZ NA PO NC`. The instruction pointer is `1100:0102` and the instruction is `0002 ADD [BP+SI],AL`. The segment register `SS` is `SS:0020=FF`.

## 5.2 Arithmetic Instructions – Addition Instructions

### ▶ EXAMPLE

The original contents of AX, BL, word-size memory location SUM, and carry flag (CF) are  $1234_{16}$ ,  $AB_{16}$ ,  $00CD_{16}$ , and  $0_{16}$ , respectively. Describe the results of executing the following sequence of instruction?

ADD AX, [SUM]

ADC BL, 05H

INC WORD PTR [SUM]

### ▶ Solution:

$$(AX) \leftarrow (AX) + (SUM) = 1234_{16} + 00CD_{16} = 1301_{16}$$

$$(BL) \leftarrow (BL) + \text{imm8} + (CF) = AB_{16} + 5_{16} + 0_{16} = B0_{16}$$

$$(SUM) \leftarrow (SUM) + 1_{16} = 00CD_{16} + 1_{16} = 00CE_{16}$$



## 5.2 Arithmetic Instructions – Addition Instructions

### ▶ EXAMPLE

What is the result of executing the following instruction sequence?

ADD AL, BL  
AAA

Assuming that AL contains  $32_{16}$  (ASCII code for 2) and BL contains  $34_{16}$  (ASCII code 4), and that AH has been cleared.

### ▶ Solution:

$$(AL) \leftarrow (AL) + (BL) = 32_{16} + 34_{16} = 66_{16}$$

The result after the AAA instruction is

$$(AL) = 06_{16}$$

$$(AH) = 00_{16}$$

with both AF and CF remain cleared

**Important:** Any adjustment operation will be performed on AL therefore the result must be always placed in AL before executing the adjustment operation

# ASCII Adjustment after ADD

## AAA Examples

Code	Registers
MOV AL, '2'	AL = 32H
MOV BL, '3'	BL = 33H
ADD AL, BL	AL = 65H
AAA	AL = 05

If low nibble of **AL**  $\leq 9$

- Clear the high nibble of AL
- AF = 0
- CF = 0

Code	Registers
MOV AL, '5'	AL = 35H
MOV BL, '6'	BL = 36H
ADD AL, BL	AL = 6BH
AAA	AX = 01H    AL=01H

If low nibble of **AL**  $> 9$  or **AF** = 1

**Clear the high nibble of AL**

- AL = AL + 6
- AH = AH + 1
- AF = 1
- CF = 1

Code	Registers
MOV AL, '9'	AL = 39H
MOV BL, '9'	BL = 39H
ADD AL, BL	AL = 72H
AAA	AX = 01H    AL=08H

If low nibble of **AL**  $> 9$  or **AF** = 1

**Clear the high nibble of AL**

- AL = AL + 6
- AH = AH + 1
- AF = 1
- CF = 1

## 5.2 Arithmetic Instructions – Addition Instructions

### ▶ EXAMPLE

Perform a 32-bit binary add operation on the contents of the processor's register.

### ▶ Solution:

$$(DX, CX) \leftarrow (DX, CX) + (BX, AX)$$

$$(DX, CX) = FEDCBA98_{16}$$

$$(BX, AX) = 01234567_{16}$$

MOV DX, FEDCH

MOV CX, BA98H

MOV BX, 0123H

MOV AX, 4567H

ADD CX, AX

ADC DX, BX ; Add with carry

# 5.2 Arithmetic Instructions - Subtraction Instructions

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	$(D) - (S) \rightarrow (D)$ Borrow $\rightarrow (CF)$	OF, SF, ZF, AF, PF, CF
SBB	Subtract with borrow	SBB D,S	$(D) - (S) - (CF) \rightarrow (D)$	OF, SF, ZF, AF, PF, CF
DEC	Decrement by 1	DEC D	$(D) - 1 \rightarrow (D)$	OF, SF, ZF, AF, PF
NEG	Negate	NEG D	$0 - (D) \rightarrow (D)$ $1 \rightarrow (CF)$	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)

(a) Subtraction Instructions. (b) Allowed operands for SUB and SBB (c) Allowed operands for DEC (d) Allowed operands for NEG

► Variety of arithmetic instruction provided to support integer subtraction—core instructions are

- SUB  $\rightarrow$  Subtract
- SBB  $\rightarrow$  Subtract with borrow
- DEC  $\rightarrow$  Decrement
- NEG  $\rightarrow$  Negative

# 5.2 Arithmetic Instructions - Subtraction Instructions

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	$(D) - (S) \rightarrow (D)$ Borrow $\rightarrow$ (CF)	OF, SF, ZF, AF, PF, CF
SBB	Subtract with borrow	SBB D,S	$(D) - (S) - (CF) \rightarrow (D)$	OF, SF, ZF, AF, PF, CF
DEC	Decrement by 1	DEC D	$(D) - 1 \rightarrow (D)$	OF, SF, ZF, AF, PF
NEG	Negate	NEG D	$0 - (D) \rightarrow (D)$ $1 \rightarrow$ (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)

(a) Subtraction Instructions. (b) Allowed operands for SUB and SBB (c) Allowed operands for DEC (d) Allowed operands for NEG

## Subtract Instruction—SUB

- SUB format and operation:

SUB D,S

$(D) - (S) \rightarrow (D)$

- Subtract values in two registers

SUB AX,BX

$(AX) - (BX) \rightarrow (AX)$

- Subtract a value in memory and a value in a register

SUB [DI],AX

$(DS:DI) - (AX) \rightarrow (DS:DI)$

- Subtract an immediate operand from a value in a register or memory

SUB AX,100H

$(AX) - IMM16 \rightarrow (AX)$

## Flags updated based on result

- CF, OF, SF, ZF, AF, PF

# 5.2 Arithmetic Instructions - Subtraction Instructions

```
CONSOLE MODE - DEBUG
-R BX
BX 0000
:1234
-R CX
CX 0000
:0123
-R F
NV UP EI PL NZ NA PO NC -
-A
0B37:0100 SBB BX,CX
0B37:0102
-R
AX=0000 BX=1234 CX=0123 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B37 ES=0B37 SS=0B37 CS=0B37 IP=0100 NV UP EI PL NZ NA PO NC
0B37:0100 19CB SBB BX,CX
-T
AX=0000 BX=1111 CX=0123 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B37 ES=0B37 SS=0B37 CS=0B37 IP=0102 NV UP EI PL NZ NA PE NC
0B37:0102 01ACEB78 ADD [SI+78EB],BP DS:78EB=0000
```

## ▶ Subtract with borrow instruction—**SBB**

- SBB format and operation:  
**SBB D,S**  
 $(D) - (S) - (CF) \rightarrow (D)$
- Used for extended subtractions
- Subtracts two registers and carry (borrow)

**SBB AX,BX**

- Example:

**SBB BX,CX**

$(BX) = 1234H$

$(CX) = 0123H$

$(CF) = 0$

$(BX) - (CX) - (CF) \rightarrow (BX)$

$1234H - 0123H - 0H =$

**1111H**

$(BX) = 1111H$

- **What about CF? CF=0**

If we execute instead:

**SBB CX, BX**

The result will be:

**CX= EEEF**

CF= 1

SF=1

PF=0



# 5.2 Arithmetic Instructions -

## Subtraction Instructions

- ▶ Negate instruction—**NEG** (2's complement)

- NEG format and operation

NEG D

$(0) - (D) \rightarrow (D)$

$(1) \rightarrow (CF)$

- Example:

NEG BX

$(BX) = 003AH$

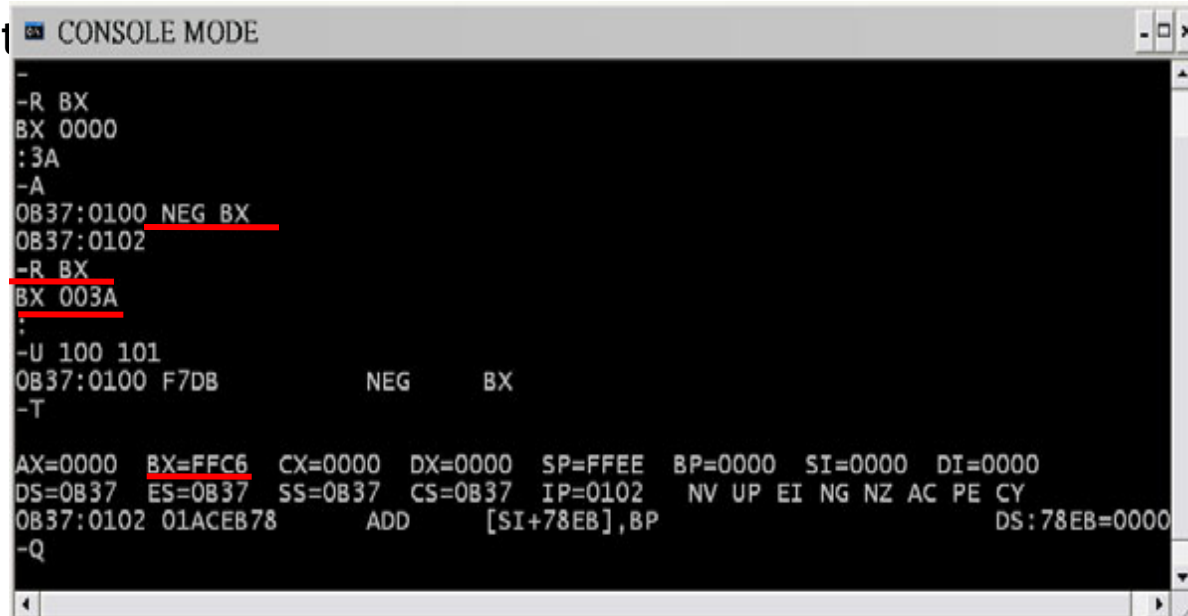
$(0) - (BX) \rightarrow (BX)$

$0000H - 003AH =$

$0000H + FFC6H$  (2's complement) =  $FFC6H$

$(BX) = FFC6H$  ;  $CF = 1$

- ▶ Since no carry is generated in this add operation, the carry flag is complemented to give  $CF = 1$ .



```
CONSOLE MODE
-
-R BX
BX 0000
:3A
-A
0B37:0100 NEG BX
0B37:0102
-R BX
BX 003A
:
-U 100 101
0B37:0100 F7DB      NEG      BX
-T
AX=0000 BX=FFC6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B37 ES=0B37 SS=0B37 CS=0B37 IP=0102  NV UP EI NG NZ AC PE CY
0B37:0102 01ACEB78  ADD      [SI+78EB],BP  DS:78EB=0000
-Q
```

# 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>
```

- ▶ Decrement instruction—**DEC**
  - DEC format and operation  
**DEC D**  
 $(D) - 1 \rightarrow (D)$
  - Used to decrement pointer—addresses
    - Example  
**DEC SI**  
 $(SI) = \text{OFFFH}$   
 $(SI) - 1 \rightarrow SI$   
 $\text{OFFFH} - 1 = \text{OFFEH}$   
 $(SI) = \text{OFFEH}$



# ASCII Adjustment after SUB: AAS Examples

Code	Registers
MOV AL, '3'	AL = 33H
MOV BL, '2'	BL = 32H
SUB AL, BL	AL = 01H
AAS	AL = 01

If low nibble of **AL**  $\leq 9$

- **Clear the high nibble of AL (no need)**
- AF = 0
- CF = 0

Code	Registers
MOV AL, '1'	AL = 31H
MOV BL, '9'	BL = 39H
SUB AL, BL	AH = 0      AL = F8H
AAS	AH = FFH      AL = 02H Answer = $-(10) + 2 = -8$

If low nibble of **AL**  $> 9$  or **AF** = 1

**Clear the high nibble of AL**

- AL = AL - 6
- AH = AH - 1
- AF = 1
- CF = 1

Code	Registers
MOV AL, '2'	AL = 32H
MOV BL, '3'	BL = 33H
SUB AL, BL	AH = 0      AL = FFH
AAS	AH = FFH      AL = 09H Answer = $-(10) + 9 = -1$

If low nibble of **AL**  $> 9$  or **AF** = 1

**Clear the high nibble of AL**

- AL = AL - 6
- AH = AH - 1
- AF = 1
- CF = 1

## *5.2 Arithmetic Instructions -* **Subtraction Instructions**

- ▶ **EXAMPLE**

Perform a 32-bit binary subtraction for variable X and Y.

- ▶ **Solution:**

```
MOV SI, 200H ; Initialize pointer for X
MOV DI, 100H ; Initialize pointer for Y
MOV AX, [SI] ; Subtract LS words
SUB AX, [DI]
MOV [SI],AX ; Save the LS word of result
MOV AX, [SI]+2 ; Subtract MS words
SBB AX, [DI]+2
MOV [SI]+2, AX ; Save the MS word of result
```

# 5.2 Arithmetic Instructions -

## Multiplication Instructions

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

(a)

Source
Reg8
Reg16
Mem8
Mem16

(b)

(a) Multiplication and Division Instructions. (b) Allowed operands

- 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
    - $(AL) \times (S8) \rightarrow (AX)$
    - product gives 16 bit result
    - $(AX) \times (S16) \rightarrow (DX), (AX)$
    - 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

### ▶ EXAMPLE

The 2's-complement signed data contents of AL are -1 and that of CL are -2. What result is produced in AX by executing the following instruction?

MUL CL

and

IMUL CL

### ▶ Solution:

(AL) = -1 (as 2's complement) =  $11111111_2 = FF_{16}$

(CL) = -2 (as 2's complement) =  $11111110_2 = FE_{16}$

Executing the MUL instruction gives

(AX) =

$11111111_2 \times 11111110_2 = 1111110100000010_2 = FD02_{16}$

Executing the IMUL instruction gives

(AX) =  $-1_{16} \times -2_{16} = 2_{16} = 0002_{16}$

If the operation is MUL CX → multiply CX by AX and store the higher order word of the result in DX and the low order word of the result in AX

## 5.2 Arithmetic Instructions - Multiplication Instructions

### ► In general:

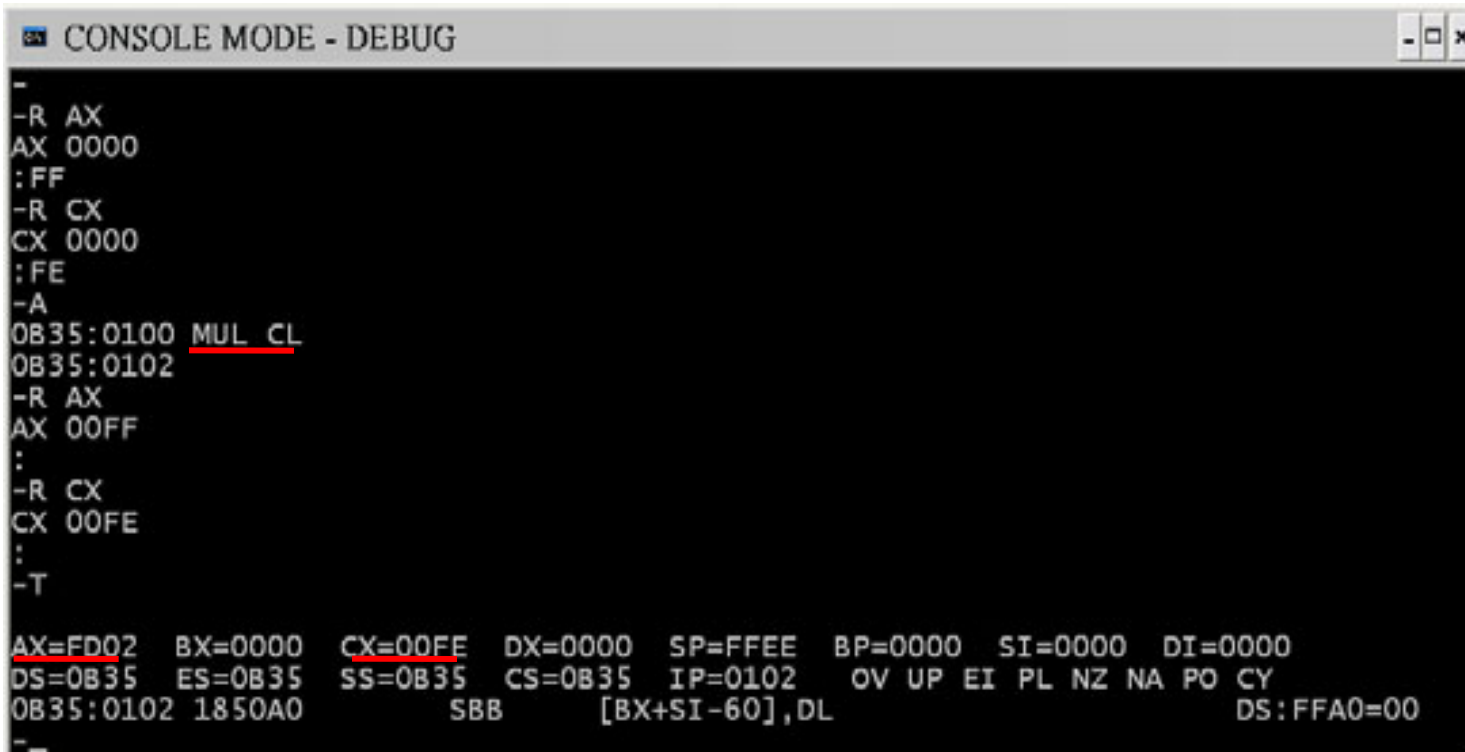
- 1 – The multiplication may take one of two forms
  - Multiply AL by 8-bit operand → result will be 16-bit saved in AX.
  - Multiply AX by 16-bit operand → result will be 32 but saved in DX,AX.
- 2 – To perform unsigned multiplication convert the two numbers into binary and perform the multiplication.
- 3 – To perform signed multiplication
  - If both operands are positive or both are negative → ignore the sign and multiply the numbers normally
  - If one operand is positive and the other is negative → multiply the numbers and perform 2's complement for the result

## 5.2 Arithmetic Instructions - Multiplication Instructions

### ▶ EXAMPLE

Verify the previous example using DEBUG program.

Solution:



```
CONSOLE MODE - DEBUG
-
-R AX
AX 0000
:FF
-R CX
CX 0000
:FE
-A
0B35:0100 MUL CL
0B35:0102
-R AX
AX 00FF
:
-R CX
CX 00FE
:
-T
AX=FD02 BX=0000 CX=00FE DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B35 ES=0B35 SS=0B35 CS=0B35 IP=0102 OV UP EI PL NZ NA PO CY
0B35:0102 1850A0 SBB [BX+SI-60],DL DS:FFA0=00
-
```

# 5.2 Arithmetic Instructions – Division

## Instructions

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

(a)

Source
Reg8
Reg16
Mem8
Mem16

(b)

(a) Multiplication and Division Instructions. (b) Allowed operands

### Integer divide instructions—DIV and IDIV

- Divide unsigned—DIV S
- Operations:

$(AX) / (S8) \rightarrow (AL) = \text{quotient}$   
 $(AH) = \text{remainder}$

- 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

$(DX,AX) / (S16) \rightarrow (AX) = \text{quotient}$   
 $(DX) = \text{remainder}$

- 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.

# Division Examples

UNSIGNED DIV	Registers (in Hex)
MOV AX, 14	AH = 00 , AL = 0E
MOV BL, 3	BL = 03
DIV BL	AH = 02 , AL = 04
MOV AX, 14	AH = 00 , AL = 0E
MOV BL, -3	BL = FD
DIV BL	Operation = 14/253 AH = 0E, AL = 00
MOV AX, -14	AH = FF , AL = F2
MOV BL, 3	BL = 03
DIV BL	Operation = 65522/3 ERROR : INT0 (Divide overflow)
MOV AX, -14	AH = FF , AL = F2
MOV BL, -3	BL = FD
DIV BL	ERROR : INT0 (Divide overflow)

SIGNED DIV	Registers (in Hex)
MOV AX, 14	AH = 00 , AL = 0E
MOV BL, 3	BL = 03
IDIV BL	AH = 02 , AL = 04
MOV AX, 14	AH = 00 , AL = 0E
MOV BL, -3	BL = FD
IDIV BL	AH = 02 , AL = FC
MOV AX, -14	AH = FF , AL = F2
MOV BL, 3	BL = 03
IDIV BL	AH = FE , AL = FC
MOV AX, -14	AH = FF , AL = F2
MOV BL, -3	BL = FD
IDIV BL	AH = FE , AL = 04

## SIGNED DIV:

The sign for the remainder == sign of the dividend



## 5.2 Arithmetic Instructions – 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

## 5.2 Arithmetic Instructions – Division Instructions

### ► In general:

1 – The division may take one of two forms

- Divide AX by 8-bit operand → The division is performed between AX/ 8-bit operand. AL will contain the quotient of the result and AH will contain the remainder of the result. IF quotient is FF then interrupt occurs.
- Divide DX,AX by 16-bit operand → The division is performed between DX,AX/ 16-bit operand. AX will contain the quotient of the result and DX will contain the remainder of the result. IF quotient is FFFF then interrupt occurs.

2 – The way in which you perform either a signed or unsigned division is similar to the mechanism used in the multiplication instruction

3 – The sign for the remainder is always similar to the sign of the dividend  
ex.  $-26 / 8 \rightarrow \text{Quotient} = -3 \text{ and Remainder} = -2$

# 5.2 Arithmetic Instructions – Convert Instructions

## ▶ EXAMPLE

What is the result of executing the following instructions?

MOV AL, 0A1H

CBW

CWD

## ▶ Solution:

$(AL) = A1_{16} = 10100001_2$

Executing the CBW instruction extends the MSB of AL

$(AH) = 11111111_2 = FF_{16}$  or

$(AX) = 1111111110100001_2$

Executing CWD instruction, we get

$(DX) = 1111111111111111_2 = FFFF_{16}$

That is,

$(AX) = FFA1_{16}$   $(DX) = FFFF_{16}$

```
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)

# 5.3 Logic Instructions

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

(a)

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

(b)

Destination
Register
Memory

(c)

(a) Logic Instructions. (b) Allowed operands for AND, OR, and XOR (c) Allowed operands for NOT

► 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

$(S) \text{ AND } (D) \rightarrow (D)$

- Logical AND of values in two registers

AND AX,BX

$(AX) \text{ AND } (BX) \rightarrow (AX)$

- Logical AND of a value in memory and a value in a register

AND [DI],AX

$(DS:DI) \text{ AND } (AX) \rightarrow (DS:DI)$

- Logical AND of an immediate operand with a value in a register or memory

AND AX,100H

$(AX) \text{ AND IMM16} \rightarrow (AX)$

- Flags updated based on result

- CF, OF, SF, ZF, PF
- AF undefined

## 5.3 Logic Instructions

### ▶ EXAMPLE

Describe the results of executing the following instructions?

```
MOV AL, 01010101B
AND AL, 00011111B
OR AL, 11000000B
XOR AL, 00001111B
NOT AL
```

### ▶ Solution:

$$(AL) = 01010101_2 \cdot 00011111_2 = 00010101_2 = 15_{16}$$

Executing the OR instruction, we get

$$(AL) = 00010101_2 + 11000000_2 = 11010101_2 = D5_{16}$$

Executing the XOR instruction, we get

$$(AL) = 11010101_2 \oplus 00001111_2 = 11011010_2 = DA_{16}$$

Executing the NOT instruction, we get

$$(AL) = (\text{NOT})11011010_2 = 00100101_2 = 25_{16}$$

## 5.3 Logic Instructions

### ▶ EXAMPLE

Verify the previous example using DEBUG program.

### ▶ Solution:

```
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]
-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:
      - $1 \text{ AND } 0 \rightarrow 0$ ;  $0 \text{ AND } 0 \rightarrow 0$ 
        - bit or bits are masked by ANDing with 0
      - $1 \text{ AND } 1 \rightarrow 1$ ;  $0 \text{ AND } 1 \rightarrow 0$ 
        - 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)  $\rightarrow$  (AX)

000FH AND FFFFH = 0000000000001111<sub>2</sub> AND 1111111111111111<sub>2</sub>  
= 0000000000001111<sub>2</sub>  
= 000FH

## 5.3 Logic Instructions – ~~Mask~~ <sup>Setting</sup> Application

- **OR** operation can be used to set a bit or bits of a byte or word to **1**
  - $X \text{ OR } 0 \rightarrow X$ ; result is **unchanged**
  - $X \text{ or } 1 \rightarrow 1$ ; 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
```
- Executing the above instructions, we get
$$(AL) = \text{XXXXXXXX}_2 \text{ OR } 00010000_2 = \text{XXX1XXXX}_2$$



# 5.4 Shift 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 $\neq 1$
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 $\neq 1$
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 $\neq 1$

(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

(b)

(a) Shift Instructions. (b) Allowed operands

► 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  $\neq 1$

**Every shift operation is equivalent to :**

**\* multiplication by 2 for Shift left**

**\* dividing by 2 for Logical Shift right**

# 5.4 Shift Instructions – Operation of the SAL/SHL Instruction

- ▶ Typical instruction—count of 1  
`SHL AX,1`

- ▶ Before execution  
Dest = (AX) = 1234H  
= 0001001000110100<sub>2</sub>  
Count = 1  
CF = X

- ▶ Operation:

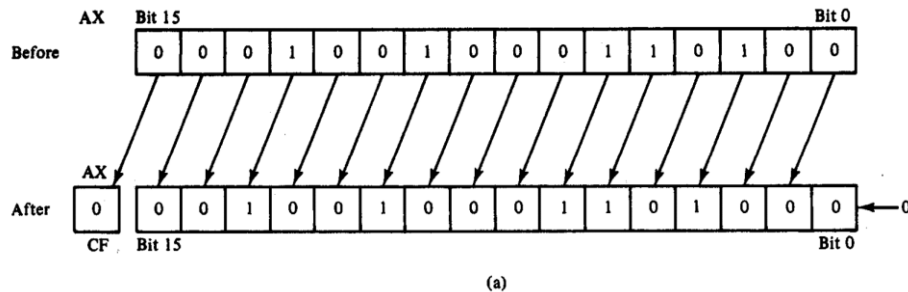
- The value in all bits of AX are shifted left one bit position
- Emptied **LSB** is filled with **0**
- Value shifted out of MSB goes to carry flag

- ▶ After execution

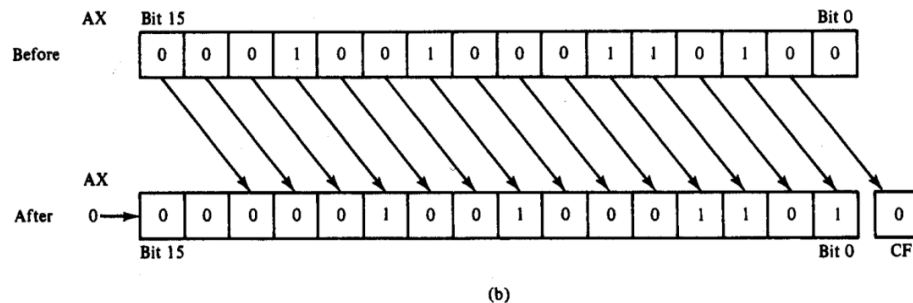
Dest = (AX) = 2468H  
= 0010010001101000<sub>2</sub>  
CF = 0

- ▶ Conclusion:

- **MSB** has been isolated in **CF** and can be acted upon by control flow instruction– conditional jump
- Result has been **multiplied by 2**



## 5.4 Shift Instructions – Operation of the SHR Instruction



- ▶ Typical instruction—count in CL

**SHR AX,CL**

- ▶ Before execution

Dest = (AX) = 1234H = 466010  
= 0001001000110100<sub>2</sub>

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 = 116510  
= 0000010010001101<sub>2</sub>

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

# 5.4 Shift Instructions – Operation of the SAR Instruction

- ▶ Typical instruction—count in CL

SAR AX,CL

- ▶ Before execution—arithmetic implies signed numbers

Dest = (AX) = 091AH

= 0000100100011010<sub>2</sub> = +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

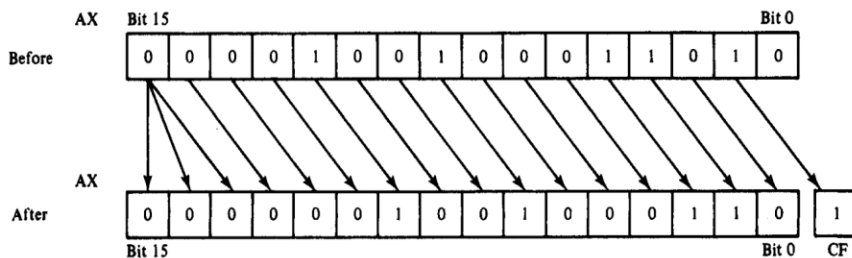
Dest = (AX) = 0246H

= 0000001001000110<sub>2</sub> = +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:  $4 \times +582 = +2328$



## 5.4 Shift Instructions – Operation of the SAR Instruction

### ▶ EXAMPLE

Assume that CL contains  $02_{16}$  and AX contains  $091A_{16}$ .

Determine the new contents of AX and the carry flag after the instruction SAR AX, CL is executed.

### ▶ Solution:

Initial (AX) =  $0000100100011010_2$

shift AX right twice: (AX) = 0000001001000110<sub>2</sub> =  $0246_{16}$

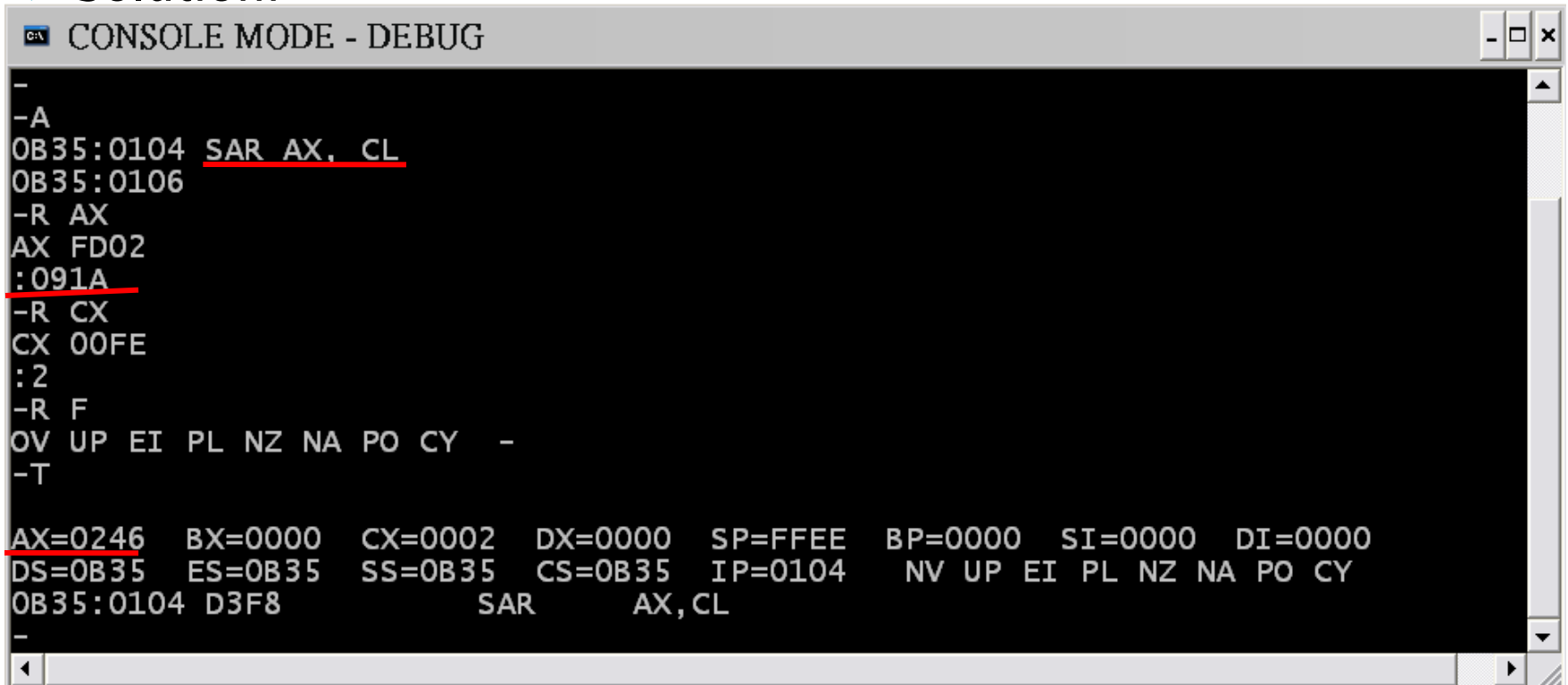
and the carry flag is (CF) =  $1_2$

## 5.4 Shift Instructions – Operation of the SAR Instruction

### ▶ EXAMPLE

Verify the previous example using DEBUG program.

### ▶ Solution:



```
CONSOLE MODE - DEBUG
-
-A
0B35:0104 SAR AX, CL
0B35:0106
-R AX
AX FD02
:091A
-R CX
CX 00FE
:2
-R F
OV UP EI PL NZ NA PO CY -
-T
AX=0246  BX=0000  CX=0002  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B35  ES=0B35  SS=0B35  CS=0B35  IP=0104  NV UP EI PL NZ NA PO CY
0B35:0104 D3F8          SAR      AX,CL
-
```

## 5.4 Shift Instructions – Operation of the SAR Instruction

### ▶ EXAMPLE

Isolate the bit B<sub>3</sub> of the byte at the offset address CONTROL\_FLAGS.

### ▶ Solution:

```
MOV AL, [CONTROL_FLAGS]
MOV CL, 04H
SHR AL, CL
```

Executing the instructions, we get

(AL)=0000B<sub>7</sub>B<sub>6</sub>B<sub>5</sub>B<sub>4</sub>

and

(CF)=B<sub>3</sub>

## 5.5 Rotate 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 $\neq 1$
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 $\neq 1$
RCL	Rotate left through carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	CF OF undefined if count $\neq 1$
RCR	Rotate right through carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	CF OF undefined if count $\neq 1$

(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

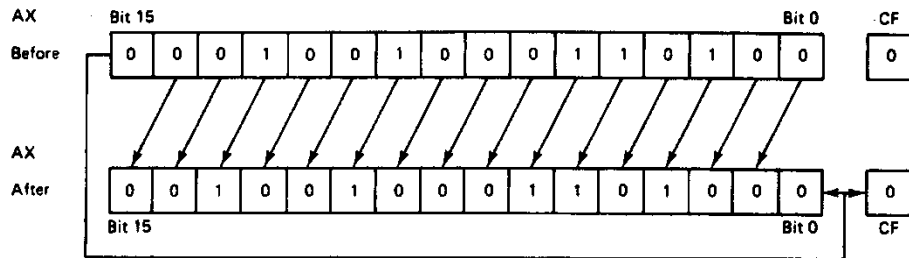
(b)

(a) Rotate Instructions. (b) Allowed operands

- ▶ 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  $\neq 1$
- ▶ Used to **rearrange** information



## 5.5 Rotate Instructions – Operation of the ROL Instruction



- ▶ Typical instruction—count of 1  
`ROL AX,1`

- ▶ Before execution

Dest = (AX) = 1234H

= 0001 0010 0011 0100<sub>2</sub>

Count = 1

CF = 0

- ▶ Operation

- The value in all bits of AX are **shifted** 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<sub>2</sub>

CF = 0

## 5.5 Rotate Instructions – Operation of the ROR Instruction

- ▶ Typical instruction—count in CL

**ROR AX,CL**

- ▶ Before execution

Dest = (AX) = 1234H

= 0001001000110100<sub>2</sub>

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

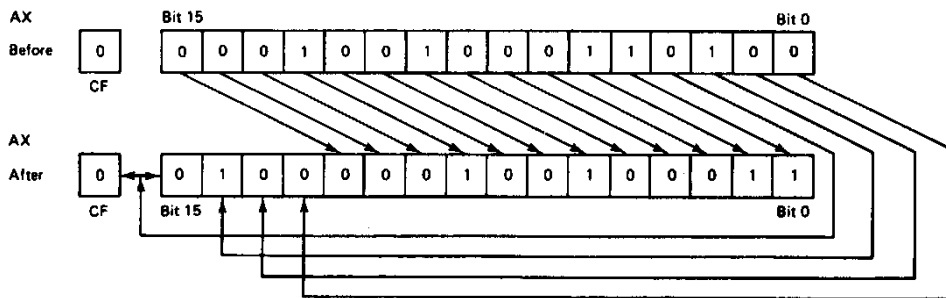
Dest = (AX) = 4123H

= 0100000100100011<sub>2</sub>

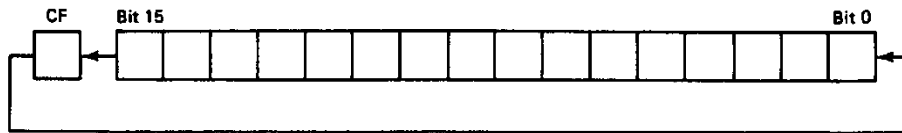
CF = 0

- ▶ **Conclusion:**

- Note that the position of hex characters in AX have be rearranged



## 5.5 Rotate Instructions – Operation of the RCL Instruction



- ▶ • Typical instruction—count in CL  
**RCL BX,CL**
- ▶ Before execution  
Dest = (BX) = 1234H  
= 0001001000110100<sub>2</sub>  
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<sub>2</sub>  
CF = 1

## 5.5 Rotate Instructions – Operation of the RCR Instruction

### ► EXAMPLE

What is the result in BX and CF after execution of the following instructions?

RCR BX, CL

Assume that, prior to execution of the instruction,  $(CL)=04_{16}$ ,  $(BX)=1234_{16}$ , and  $(CF)=0$

### ► Solution:

The original contents of BX are

$$(BX) = 0001001000110100_2 = 1234_{16}$$

Execution of the RCR command causes a 4-bit rotate right through carry to take place on the data in BX, the results are

$$(BX) = 1000000100100011_2 = 8123_{16}$$

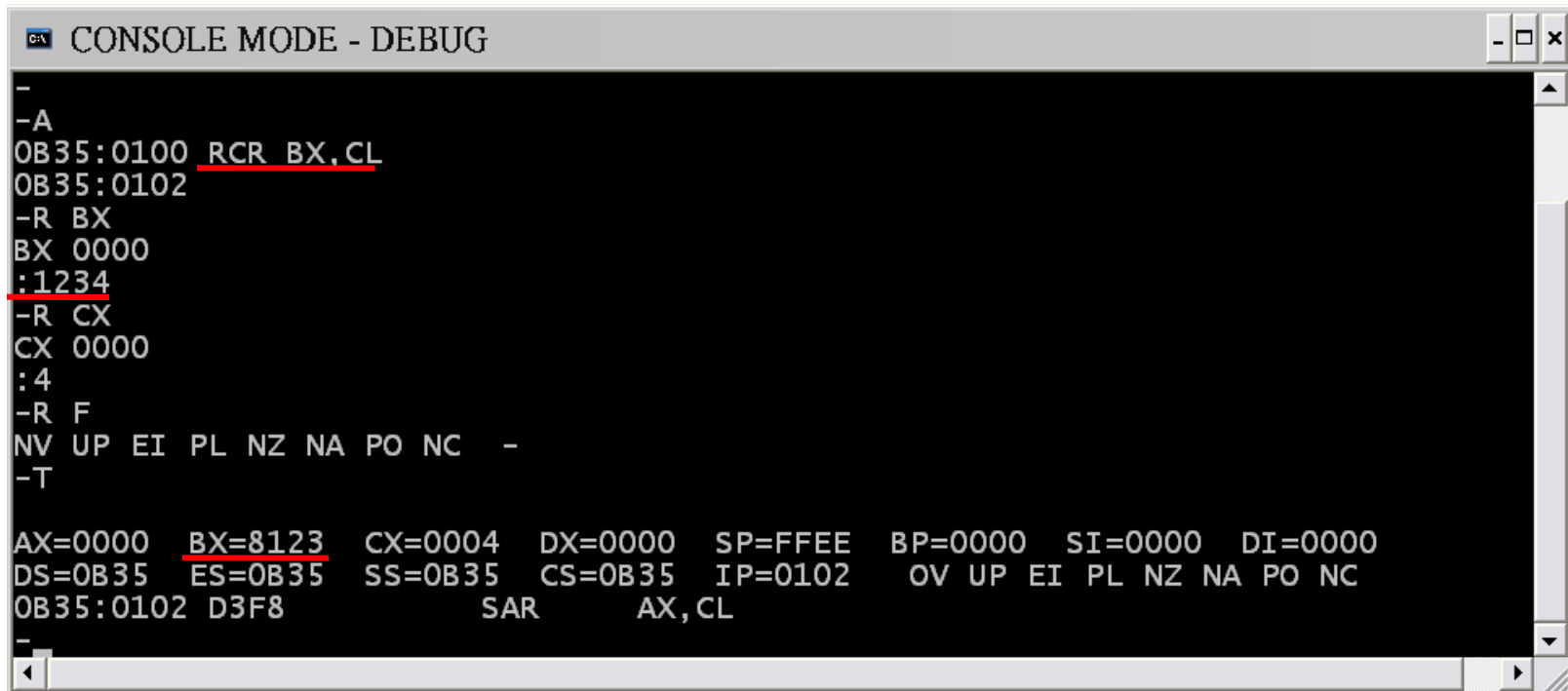
$$(CF) = 0_2$$

## 5.5 Rotate Instructions – Operation of the RCR Instruction

### ▶ EXAMPLE

Verify the previous example using DEBUG program.

### ▶ Solution:



```

C:\> CONSOLE MODE - DEBUG
-
-A
0B35:0100 RCR BX,CL
0B35:0102
-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=0B35 ES=0B35 SS=0B35 CS=0B35 IP=0102 OV UP EI PL NZ NA PO NC
0B35:0102 D3F8 SAR AX,CL
-

```

## 5.5 Rotate Instructions

### ▶ EXAMPLE

Disassembly and addition of 2 hexadecimal digits stored as a byte in memory.

### ▶ Solution:

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

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

# H.W. #5

- Solve the following problems from Chapter 5 from the course textbook:

1, 10, 26, 38, 47