Chapter 5

Real-Mode 80386DX
Microprocessor Programming 1
Part 1

Introduction

5.2 Data-Transfer Instructions

5.3 Arithmetic Instructions

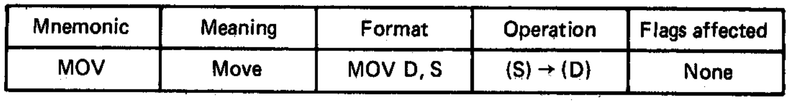
5.4 Logic Instructions

5.5 Shift Instructions

5.6 Rotate Instructions

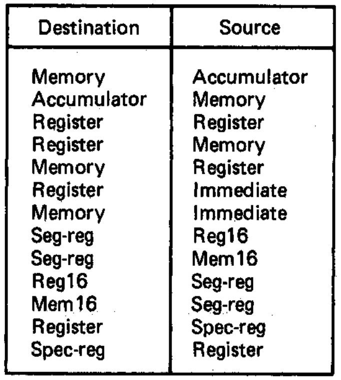
5.7 Bit Test and Bit Scan Instructions

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
Mem16	Seg-reg
Register	Spec-reg
Spec-reg	Register

(b)

- **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,EAX)

  Immediate operand (S only)

  Segment register (Seg-reg)
- **Example:**

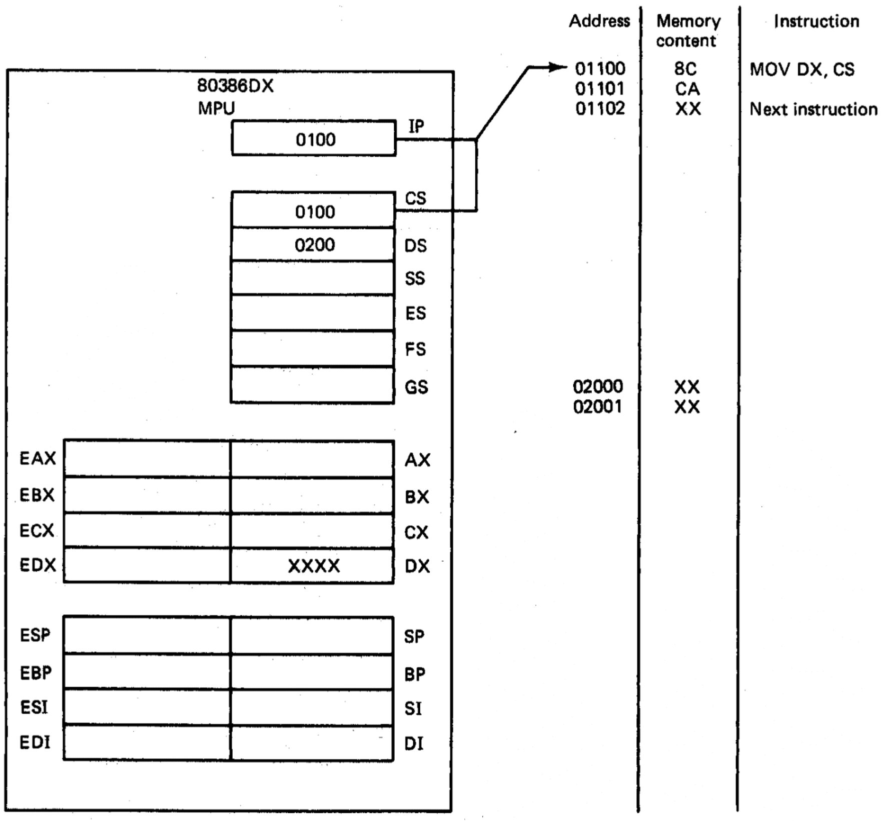
  MOV [SUM],AX

  (AL) → (address SUM)

  (AH) → (address SUM+1)

What is the addressing mode of the destination?

Example of Move Instruction



Address	Memory content	Instruction
01100	8C	MOV DX, CS
01101	CA	
01102	XX	Next instruction
02000	XX	
02001	XX	

- **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

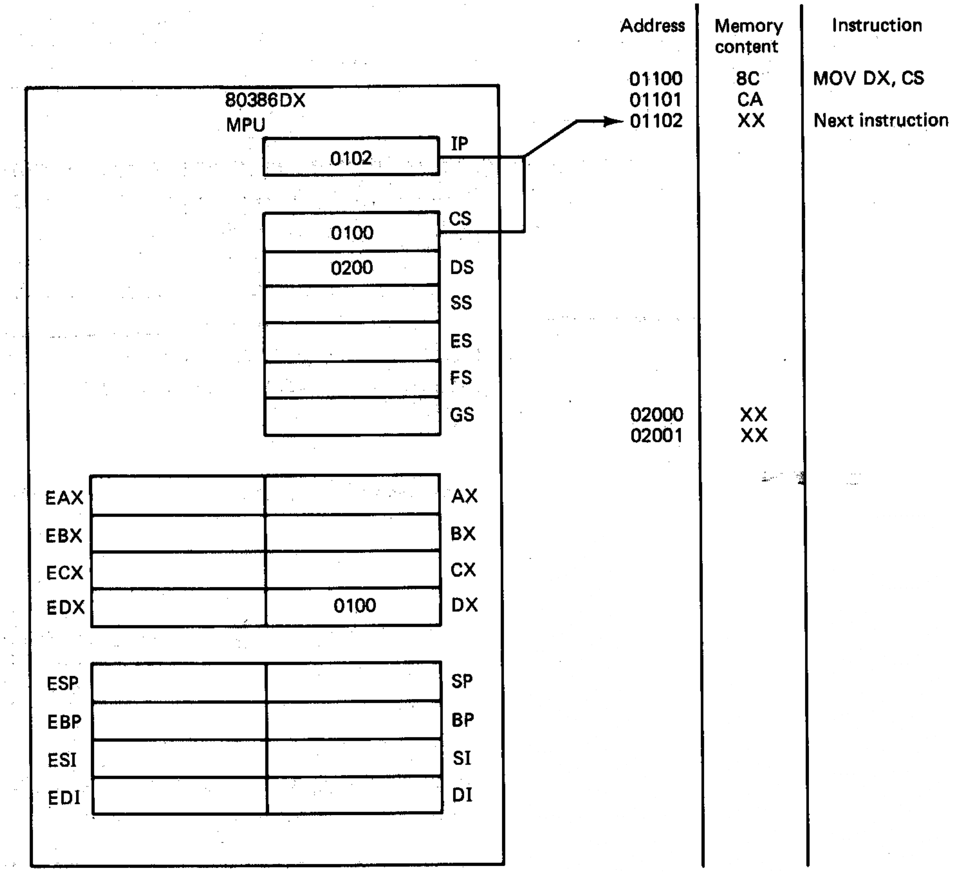
  (01100H) = 8CH

  (01101H) = CAH

  (CS) = 0100H

  (DX) = XXXX → don't care state

Example of Move Instruction



Address	Memory content	Instruction
01100	8C	MOV DX, CS
01101	CA	
01102	XX	Next instruction
02000	XX	
02001	XX	

- **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

- Rest of the bits in EDX unaffected

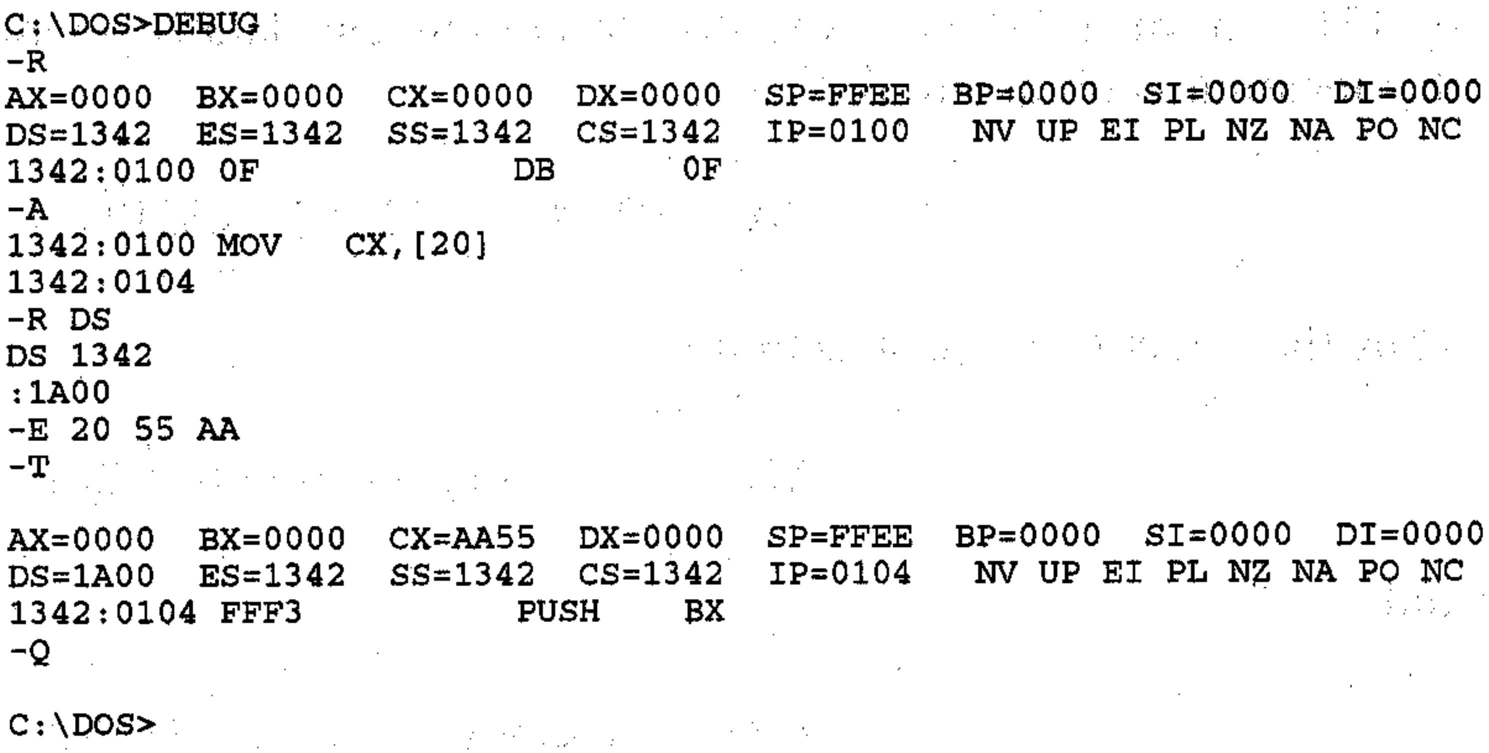
- Value in CS unchanged

How are the flags affected?

Exeuction of Move Instruction

- **Debug execution example**

  MOV CX,[20]
  DS = 1A00
  (DS:20) = AA55H
  (1A00:20) → (CX)

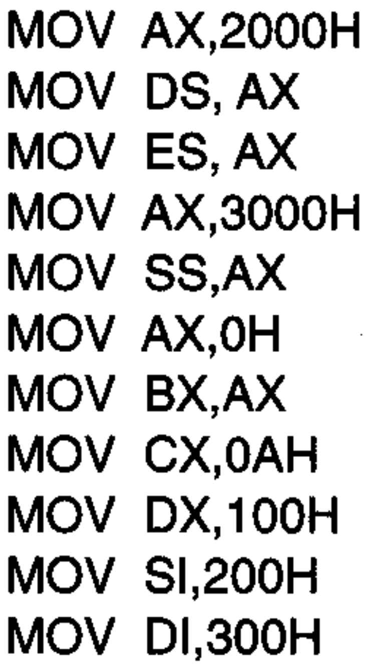


```
C:\DOS>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 0F            DB      0F
-A
1342:0100 MOV   CX,[20]
1342:0104
-R DS
DS 1342
:1A00
-E 20 55 AA
-T

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>
```

Usage of Move Instruction



```
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
```

- **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**

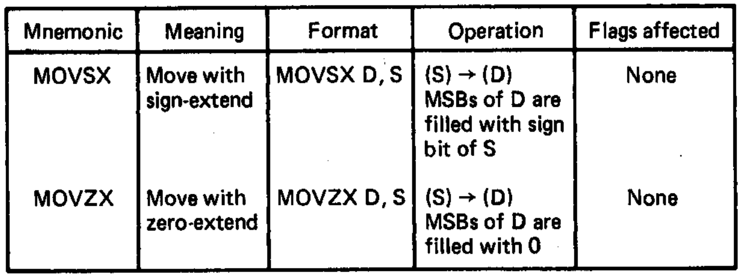
    IMM16 → (AX) =0000H

    (AX) → (BX) =0000H

    IMM16 → (CX) = 000AH and (DX) = 0100H
  - **Index register initialized from immediate operands**

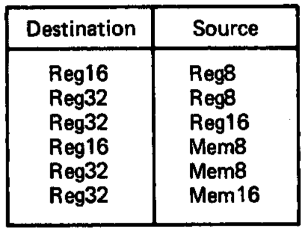
    IMM16 → (SI) = 0200H and (DI) = 0300H

Sign-Extend and Zero-Extend Move Instruction



Mnemonic	Meaning	Format	Operation	Flags affected
MOVSX	Move with sign-extend	MOVSX D, S	(S) → (D) MSBs of D are filled with sign bit of S	None
MOVZX	Move with zero-extend	MOVZX D, S	(S) → (D) MSBs of D are filled with 0	None

(a)



Destination	Source
Reg16	Reg8
Reg32	Reg8
Reg32	Reg16
Reg16	Mem8
Reg32	Mem8
Reg32	Mem16

(b)

Where might one use these instructions?

Why both sign extend and zero extend?

- **Sign-Extend Move instruction**
  - **Used to move (copy) data between two registers or memory and a register and extend the value with the value of the sign bit**
  - **General format:**

    MOVSX D,S
  - **Operation: Copies the content of the source to the destination**

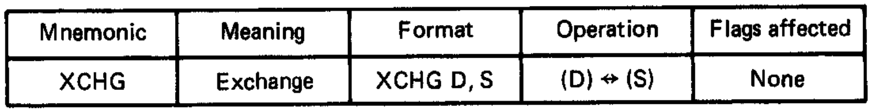
    (S) → (D); source contents unchanged
    - **Sign bit → extended through bit 16 or 32**
    - **Flags unaffected**
  - **Examples: MOVSX EBX,AX**

    Where: (AX) = FFFFH
    S = Reg16
    D = Reg32
    Sign bit (AX) = 1
    FFFFFFFFH → (EBX)
- **Zero-Extend Move instruction—MOVZX**
  - **Operates the same as MOVSX except extends with zeros**
  - **Example:MOVZX CX, Byte Pointer [DATA_BYTE]**

    Where: (DATA_BYTE) = FFH,S = Mem8,D = Reg16

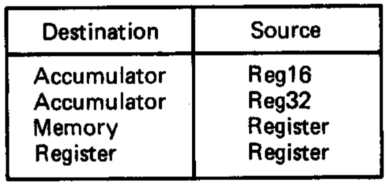
    00FFH → (CX)

Exchange Instruction



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

(a)



Destination	Source
Accumulator	Reg16
Accumulator	Reg32
Memory	Register
Register	Register

(b)

- **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**

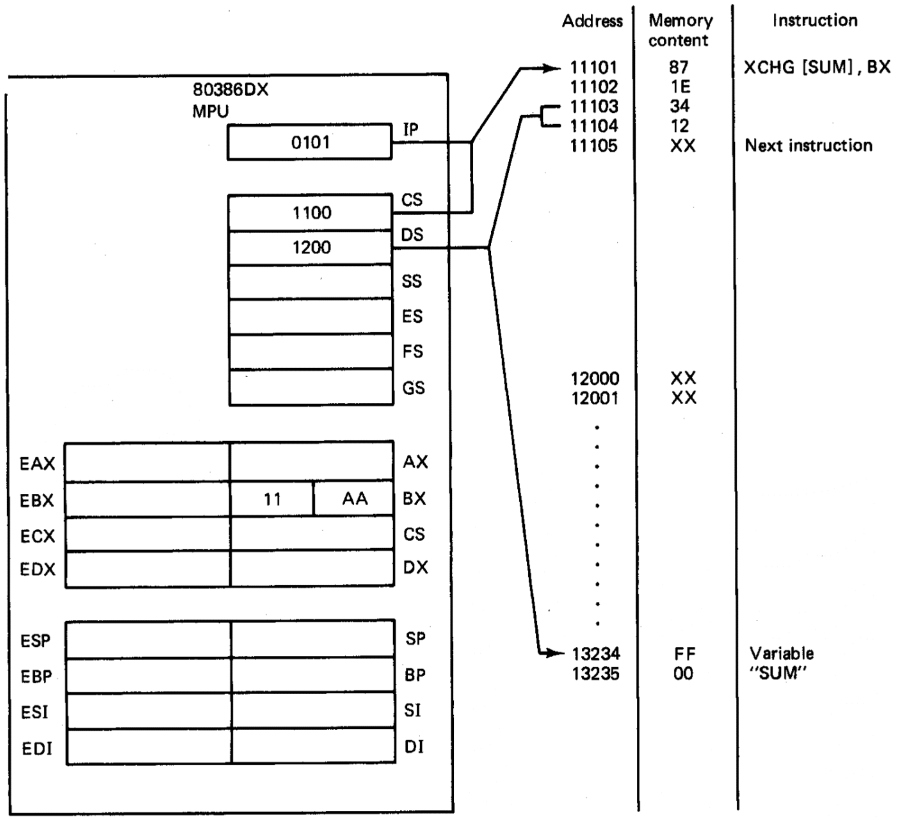
    (S) → (D)

    (D) → (S)
  - **Flags unaffected**
- **Special accumulator destination version executes faster**
- **Examples: XCHG AX,DX**

  (Original value in AX) → (DX)

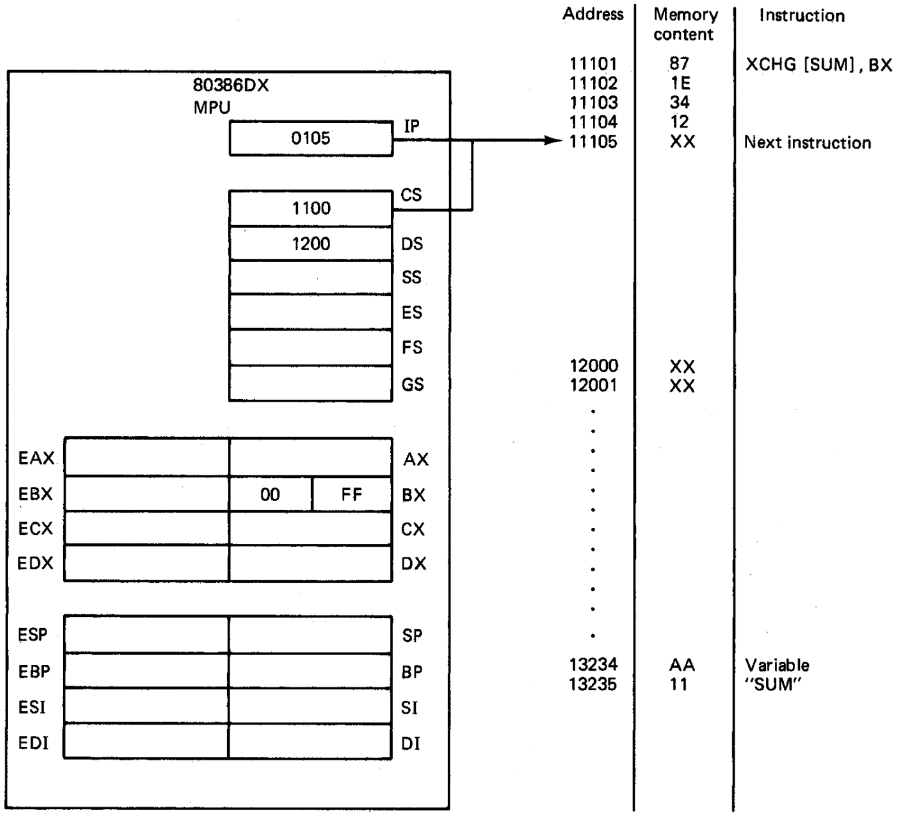
  (Original value in DX) → (AX)

Example of Exchange Instruction



- **Example**
  XCHG [SUM],BX
  Source = BX → word data
  Destination = memory address
  SUM → word data
  Operation:
  (SUM) → (BX)
  (BX) → (SUM)
- **State before fetch and execution**
  CS:IP = 1100:0101 = 11101H
  XCHG instruction code = 871E3412H
  (01104H,01103H) = 1234H = SUM
  (DS) = 1200H
  (BX) = 11AA
  (DS:SUM) = (1200:1234) = 00FFH

Example of Exchange Instruction



Address	Memory content	Instruction
11101	87	XCHG [SUM], BX
11102	1E	
11103	34	
11104	12	
11105	XX	Next instruction
12000	XX	
12001	XX	
⋮		
13234	AA	Variable "SUM"
13235	11	

- Example (continued)
- State after execution

CS:IP = 1100:0105 = 11105H

11105H → points to next sequential instruction

(BX) = 00FFH

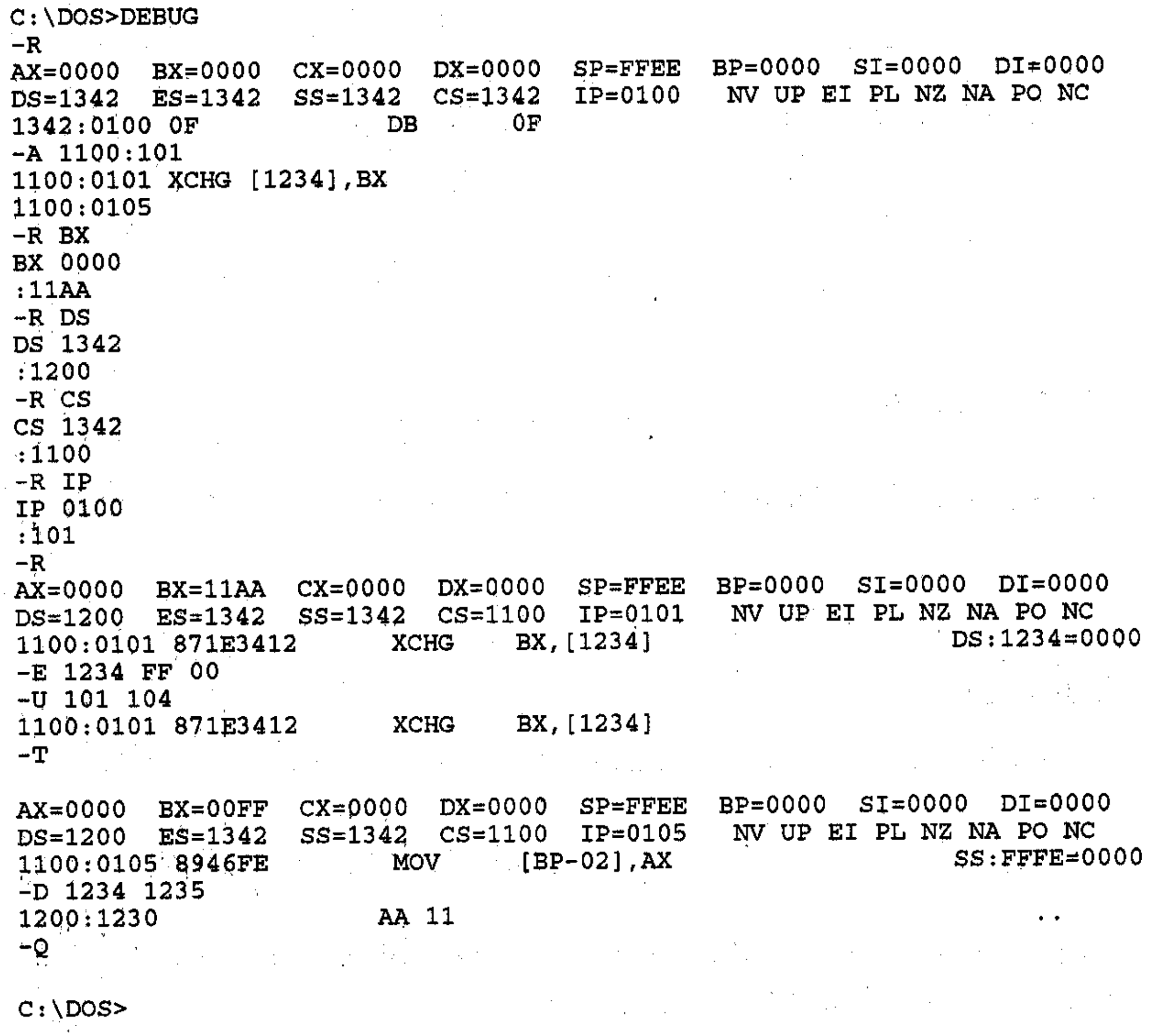
- Rest of the bits in EBX unaffected

- Memory updated

(1200:1234) = AAH

(1200:1235) = 11H

Execution of Exchange Instruction

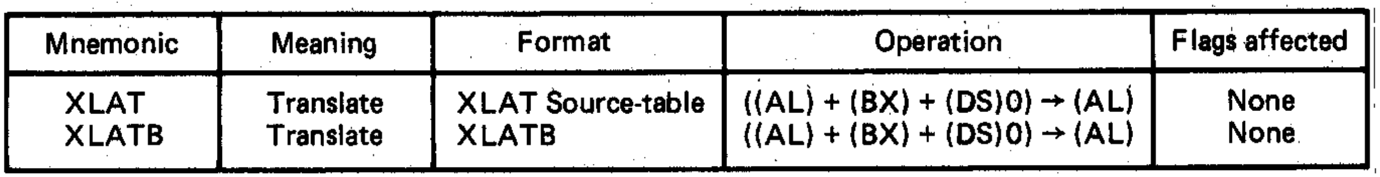


```
C:\DOS>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 0F            DB      0F
-A 1100:101
1100:0101 XCHG [1234],BX
1100:0105
-R BX
BX 0000
:11AA
-R DS
DS 1342
:1200
-R CS
CS 1342
:1100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

Translate Instruction



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

- Translate instruction
  - Used to look up a byte-wide value in a table in memory and copy that value into the AL register
  - General format: XLAT
    - Operands are said to be “implicit”
  - 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:

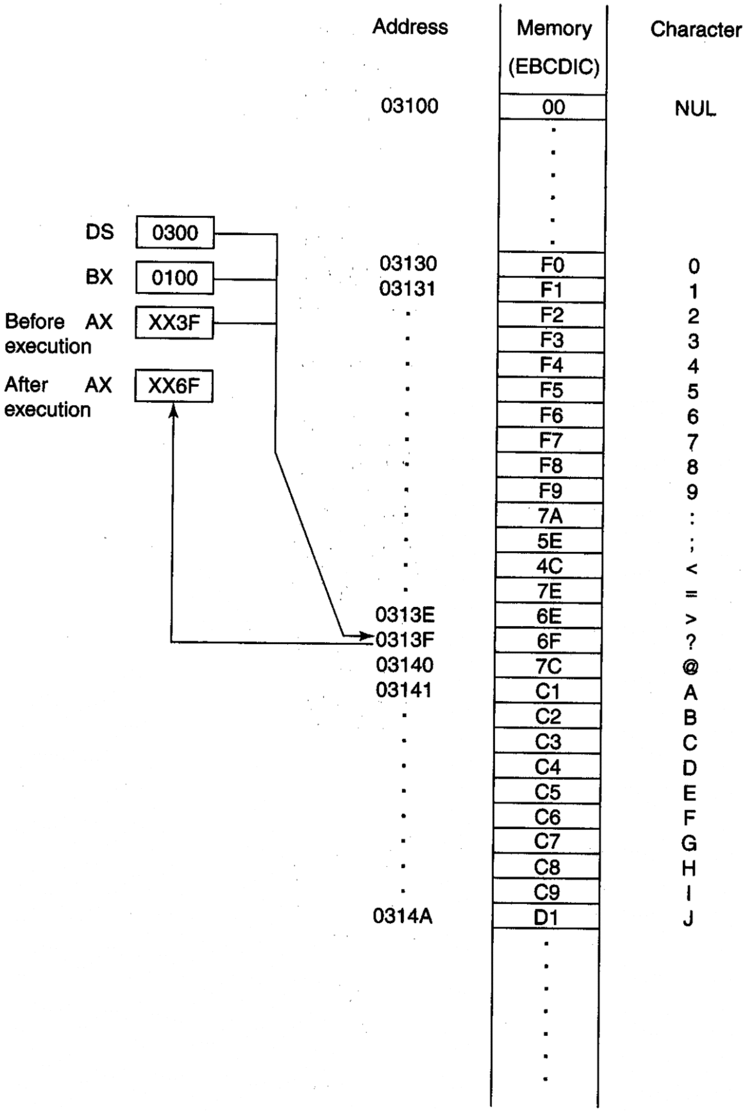
    (DS)0 = Points to the active data segment

    (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 element

Translate Instruction (example)



DS	0300
BX	0100
Before execution AX	XX3F
After execution AX	XX6F

Address	Memory (EBCDIC)	Character
03100	00	NUL
.	.	
03130	F0	0
03131	F1	1
	F2	2
	F3	3
	F4	4
	F5	5
	F6	6
	F7	7
	F8	8
	F9	9
	7A	:
	5E	;
	4C	<
	7E	=
0313E	6E	>
0313F	6F	?
03140	7C	@
03141	C1	A
	C2	B
	C3	C
	C4	D
	C5	E
	C6	F
	C7	G
	C8	H
	C9	I
0314A	D1	J
	.	

- 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

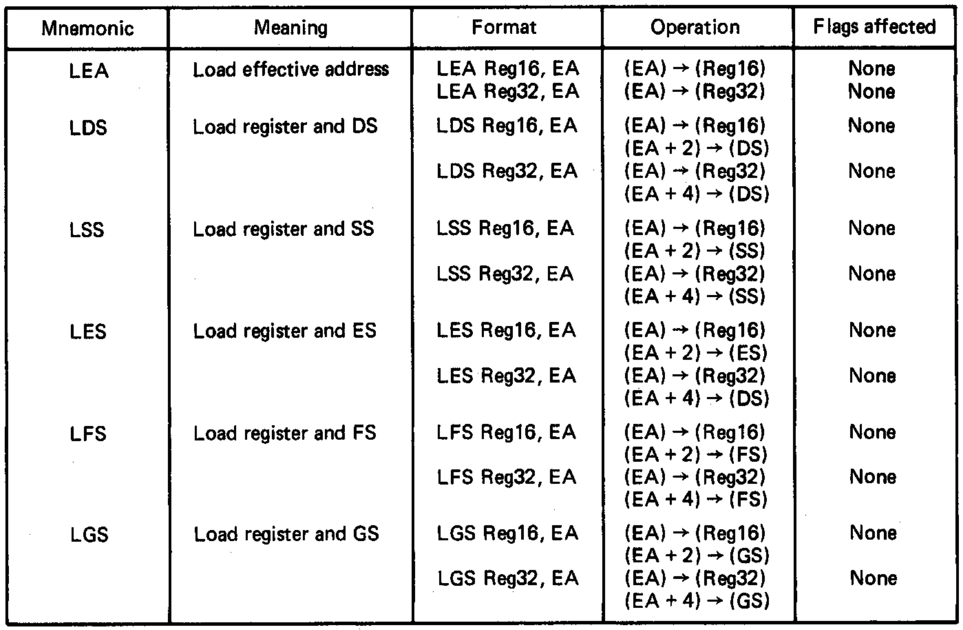
  (DS) = 0300H

  (BX) = 0100H

  (AL) = 3FH → 6FH = ? (Question mark)

* Figure not in textbook

Load Effective Address



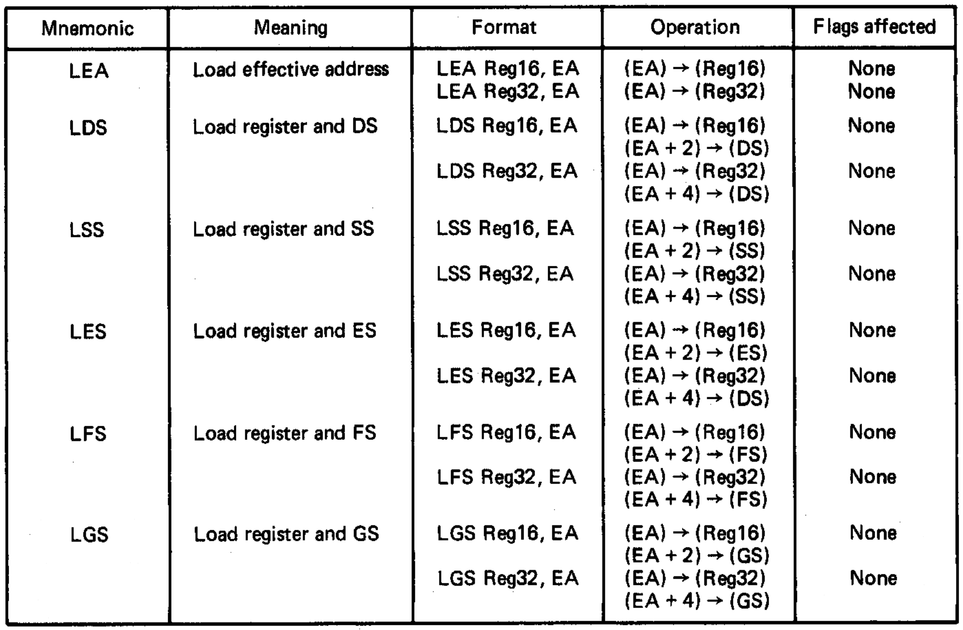
Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg16, EA LEA Reg32, EA	(EA) → (Reg16) (EA) → (Reg32)	None None
LDS	Load register and DS	LDS Reg16, EA LDS Reg32, EA	(EA) → (Reg16) (EA + 2) → (DS) (EA) → (Reg32) (EA + 4) → (DS)	None None
LSS	Load register and SS	LSS Reg16, EA LSS Reg32, EA	(EA) → (Reg16) (EA + 2) → (SS) (EA) → (Reg32) (EA + 4) → (SS)	None None
LES	Load register and ES	LES Reg16, EA LES Reg32, EA	(EA) → (Reg16) (EA + 2) → (ES) (EA) → (Reg32) (EA + 4) → (DS)	None None
LFS	Load register and FS	LFS Reg16, EA LFS Reg32, EA	(EA) → (Reg16) (EA + 2) → (FS) (EA) → (Reg32) (EA + 4) → (FS)	None None
LGS	Load register and GS	LGS Reg16, EA LGS Reg32, EA	(EA) → (Reg16) (EA + 2) → (GS) (EA) → (Reg32) (EA + 4) → (GS)	None None

- **Load effective address instruction**
  - **Used to load the effective address of a pointer from memory into a register**
  - **General format:**

    LEA Reg16/32,EA
  - **Operation:**

    (EA) → (Reg16/32)
    - **Source unaffected:**
    - **Flags unaffected**

Load Full Pointer Instructions



Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg16, EA LEA Reg32, EA	(EA) → (Reg16) (EA) → (Reg32)	None None
LDS	Load register and DS	LDS Reg16, EA LDS Reg32, EA	(EA) → (Reg16) (EA + 2) → (DS) (EA) → (Reg32) (EA + 4) → (DS)	None None
LSS	Load register and SS	LSS Reg16, EA LSS Reg32, EA	(EA) → (Reg16) (EA + 2) → (SS) (EA) → (Reg32) (EA + 4) → (SS)	None None
LES	Load register and ES	LES Reg16, EA LES Reg32, EA	(EA) → (Reg16) (EA + 2) → (ES) (EA) → (Reg32) (EA + 4) → (DS)	None None
LFS	Load register and FS	LFS Reg16, EA LFS Reg32, EA	(EA) → (Reg16) (EA + 2) → (FS) (EA) → (Reg32) (EA + 4) → (FS)	None None
LGS	Load register and GS	LGS Reg16, EA LGS Reg32, EA	(EA) → (Reg16) (EA + 2) → (GS) (EA) → (Reg32) (EA + 4) → (GS)	None None

- **Used to load a full address pointer from memory into a segment register and register**
- **General formats and operation for LDS and LSS**

  LDS Reg16/32,EA

  (EA) → (Reg16/32)

  (EA+2/4) →(DS)

  LSS Reg16/32,EA

  (EA) → (Reg16/32)

  (EA+2/4) → (SS)
- **LES, LFS, and LGS operate the same**

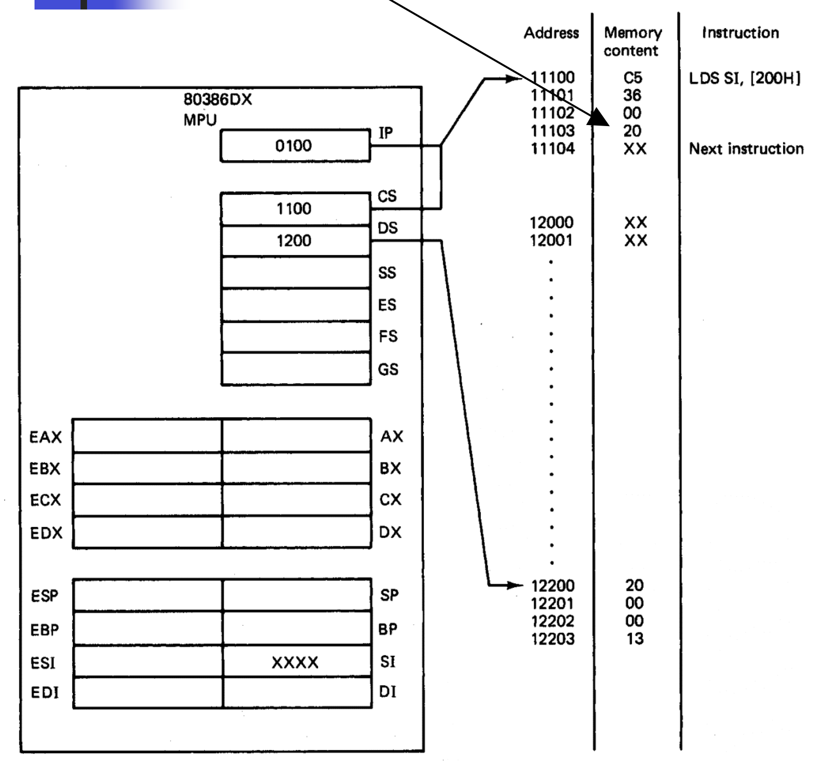
LES Reg16/32,EA (EA) → (Reg16/32),(ES)

LFS Reg16/32,EA (EA) → (Reg16/32),(FS)

LGS Reg16/32,EA (EA) → (Reg16/32),(GS)

Load Full Pointer Instructions (example)

Error 02 not 20



- Example

  LDS SI,[200H]

  Source = pointer at DS:200H→ 32 bits

  Destination = SI → offset of pointer

  DS → sba of pointer

  Operation: (DS:200H) → (SI)

  (DS:202H) → (DS)

- State before fetch and execution

  CS:IP = 1100:0100 = 11100H

  LDS instruction code = C5360002H

  (11102H,11103H) = (EA) = 2000H

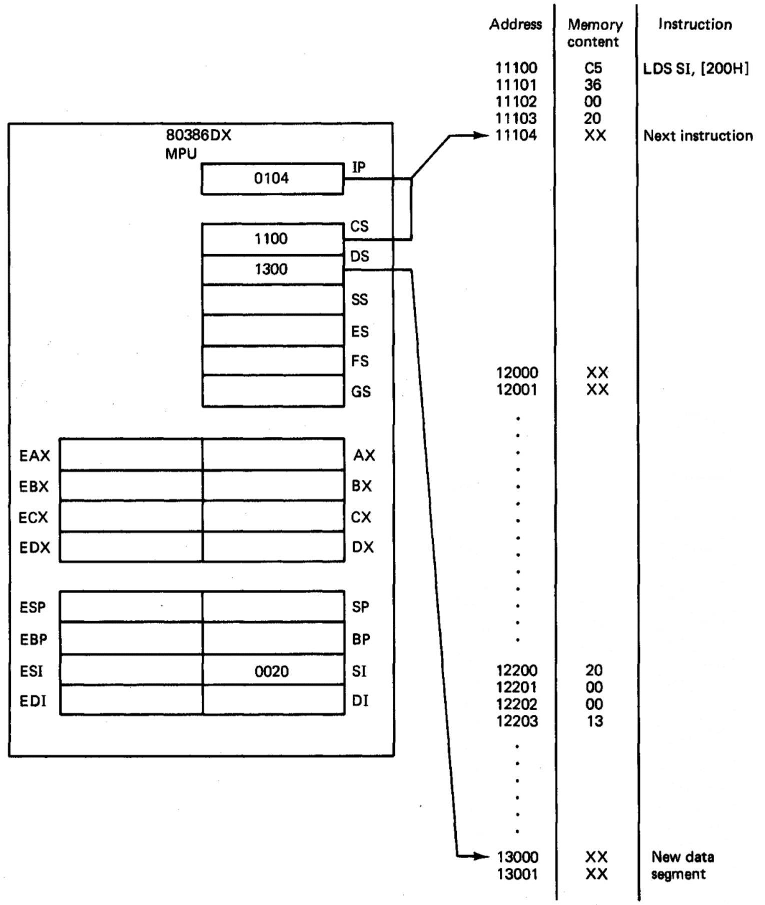
  (DS) = 1200H

  (SI) = XXXX → don’t care state

  (DS:EA) = 12200H = 0020H

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

Load Full Pointer Instructions (example)



Address	Memory content	Instruction
11100	C5	LDS SI, [200H]
11101	36	
11102	00	
11103	20	
11104	XX	Next instruction
12000	XX	
12001	XX	
12200	20	
12201	00	
12202	00	
12203	13	
13000	XX	New data segment
13001	XX	

- Example (continued)
- State after execution

  CS:IP = 1100:0104 = 11104H

  01004H → points to next

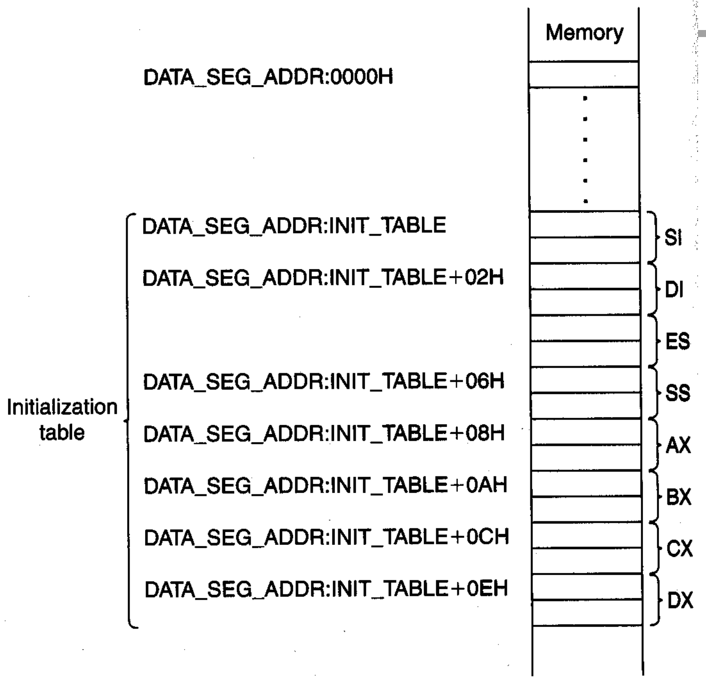
  sequential instruction

  (DS) = 1300H → defines new data segment

  (SI) = 0020H

  - Rest of the bits in ESI unaffected

Example


Memory
DATA_SEG_ADDR:0000H
DATA_SEG_ADDR:INIT_TABLE
DATA_SEG_ADDR:INIT_TABLE+02H
DATA_SEG_ADDR:INIT_TABLE+06H
DATA_SEG_ADDR:INIT_TABLE+08H
DATA_SEG_ADDR:INIT_TABLE+0AH
DATA_SEG_ADDR:INIT_TABLE+0CH
DATA_SEG_ADDR:INIT_TABLE+0EH
Initialization table
SI
DI
ES
SS
AX
BX
CX
DX

```
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) → 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**

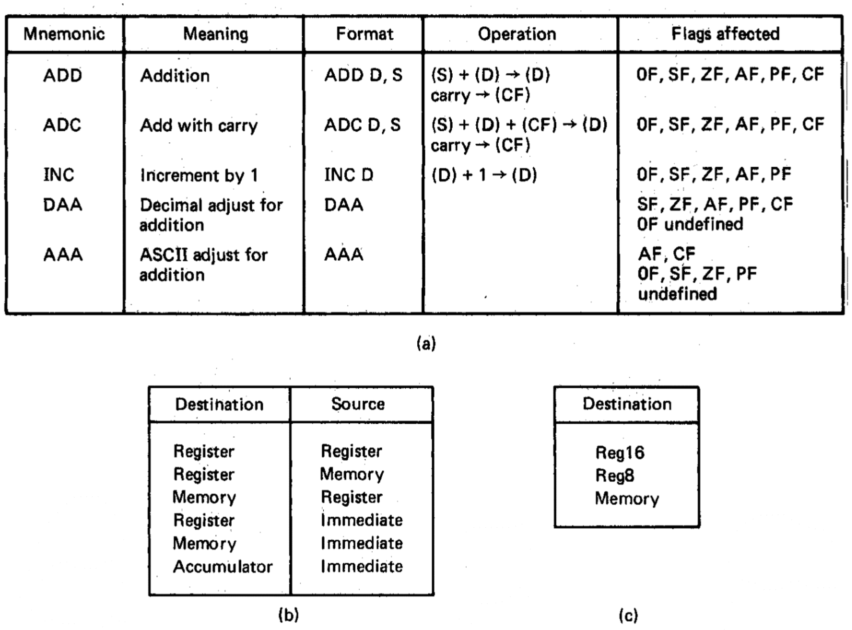
    (INIT_TABLE+8,INIT_TABLE+9) → AX

    (INIT_TABLE+A,INIT_TABLE+B) → BX

    (INIT_TABLE+C,INIT_TABLE+D) → CX

    (INIT_TABLE+E,INIT_TABLE+F) → DX

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
DAA	Decimal adjust for addition	DAA		SF, ZF, AF, PF, CF OF undefined
AAA	ASCII adjust for addition	AAA		AF, CF OF, SF, ZF, PF 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) & CF
    - Add a value in memory and a value in a register

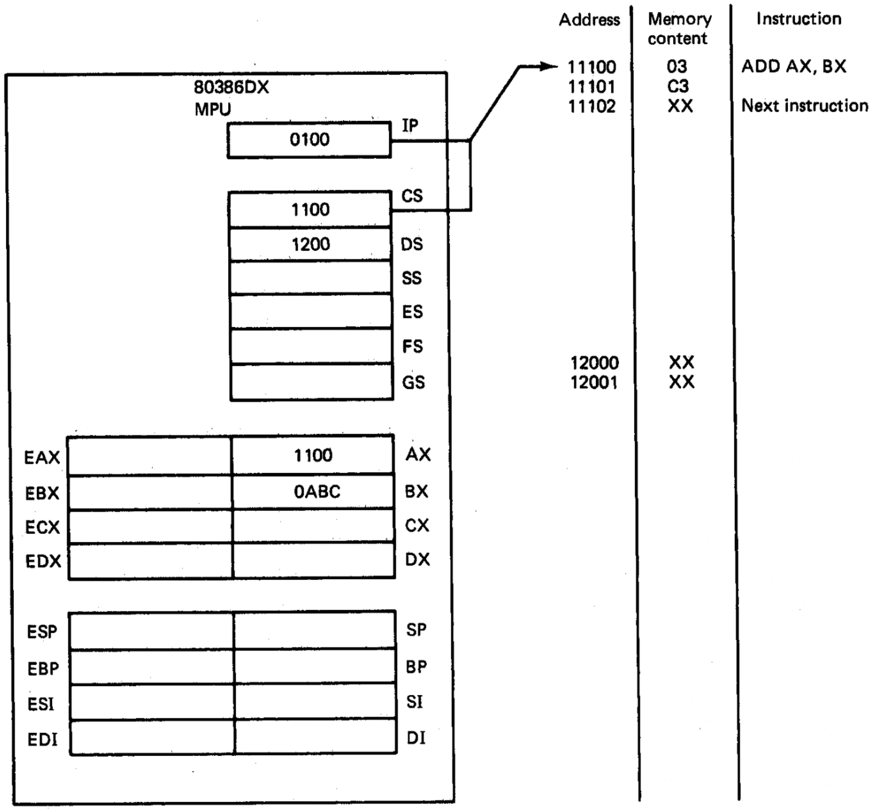
      ADD [DI],AX

      (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

Addition Instructions (example)



Address	Memory content	Instruction
11100	03	ADD AX, BX
11101	C3	
11102	XX	Next instruction
12000	XX	
12001	XX	

- 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 instruction code = 03C3H

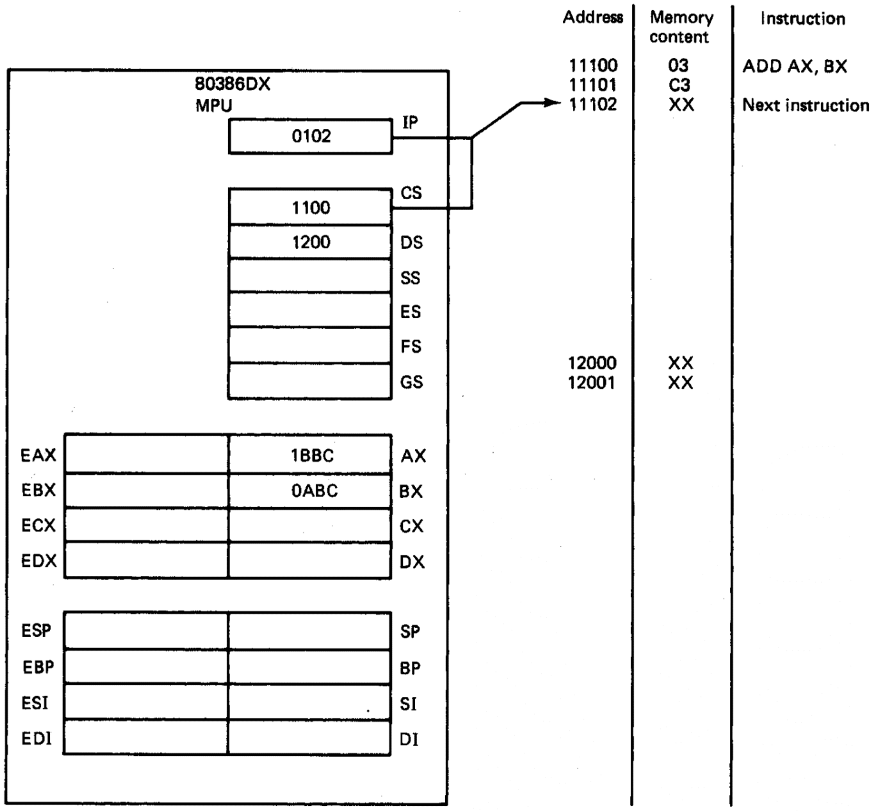
  (AX) = 1100H

  (BX) = 0ABCH

  (DS) = 1200H

  (1200:0000) = 12000H = XXXX

Addition Instructions (example)



- **Example (continued)**
- **State after execution**

  CS:IP = 1100:0102 = 11102H

  01002H → points to next sequential instruction

  - **Operation performed**

    (AX) + (BX) = (AX)

    (1100H) + (0ABCH) = 1BBCH

    = 0001101110111100_2

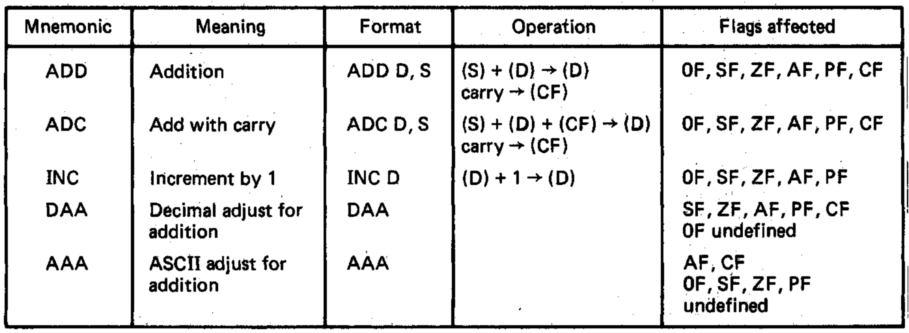
    (AX) = 1BBCH

    Upper bits of (AX) unchanged

    (BX) = unchanged

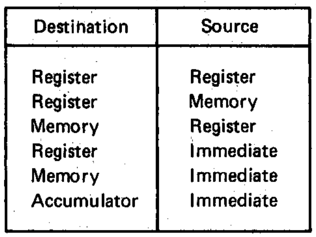
  - **Impact on flags**
    - **CF =0 (no carry resulted)**
    - **ZF = 0 (not zero)**
    - **SF = 0 (positive)**
    - **PF = 0 (odd parity)**

Other Addition Instructions



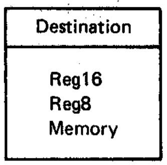
Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D, S	(S) + (D) → (D) carry → (CF)	OF, SF, ZF, AF, PF, CF
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) → (D) carry → (CF)	OF, SF, ZF, AF, PF, CF
INC	Increment by 1	INC D	(D) + 1 → (D)	OF, SF, ZF, AF, PF
DAA	Decimal adjust for addition	DAA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

    ADC AX,BX

    (AX) + (BX) + (CF) → (AX) & CF
  - 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

Examples of Addition Instructions

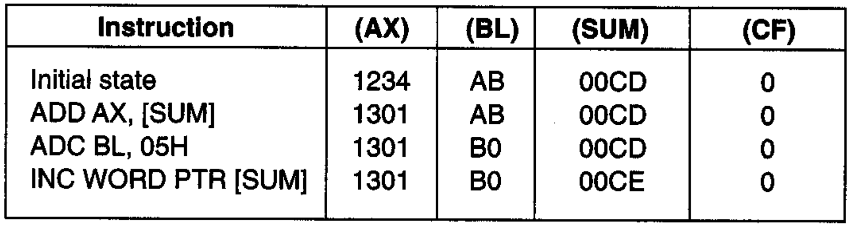
- **Example—Arithmetic computations**
  - **Initial state:**

    (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**

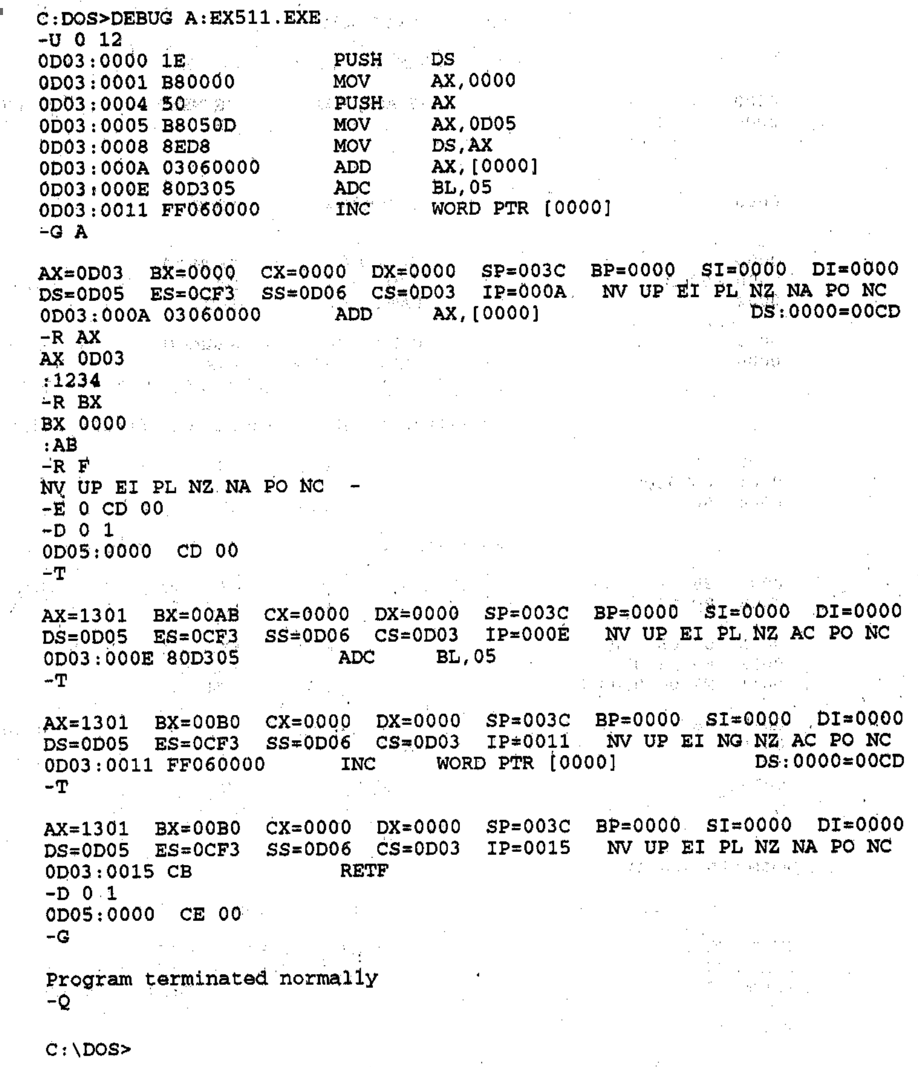
    (DS:SUM) + 1 → (DS:SUM)
    00CDH + 1 = 00CEH
    (SUM) = 00CEH
    (CF) = unchanged



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

Examples of Addition Instructions

- Example—Execution of the arithmetic computation



```
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>
```

Subtraction Instructions

- Variety of arithmetic instructions 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

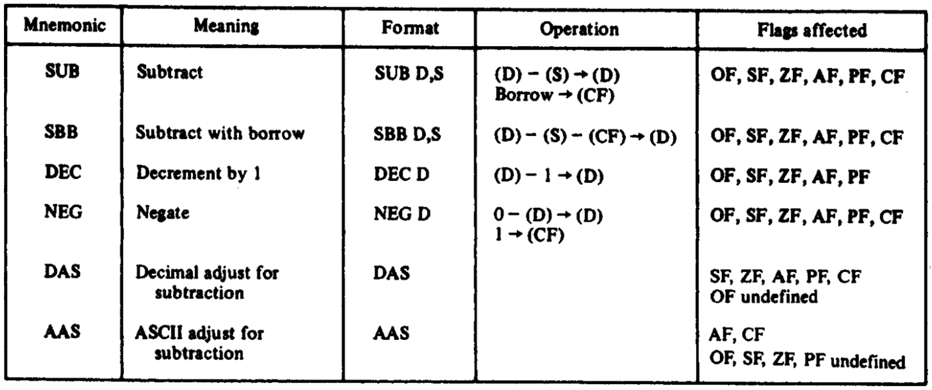
      (AX) - (BX) → (AX)
    - Subtract a value in memory and a value in a register

      SUB [DI],AX

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

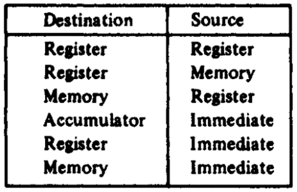
      SUB AX,100H

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



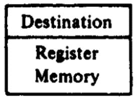
Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) − (S) → (D) Borrow → (CF)	OF, SF, ZF, AF, PF, CF
SBB	Subtract with borrow	SBB D,S	(D) − (S) − (CF) → (D)	OF, SF, ZF, AF, PF, CF
DEC	Decrement by 1	DEC D	(D) − 1 → (D)	OF, SF, ZF, AF, PF
NEG	Negate	NEG D	0 − (D) → (D) 1 → (CF)	OF,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

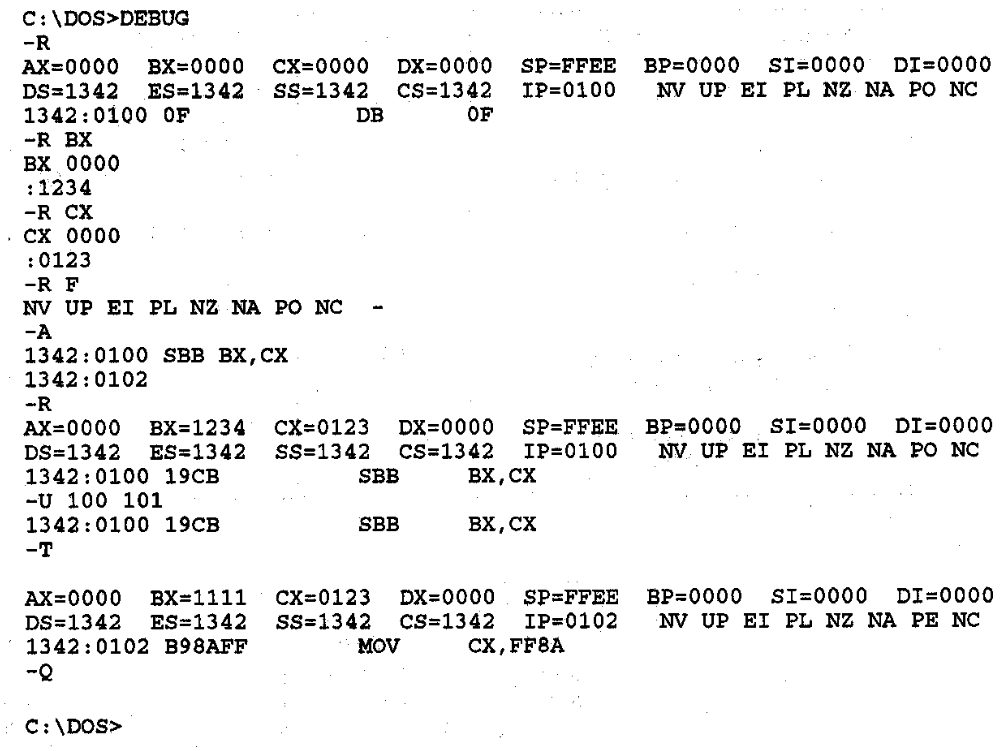
(b)



Destination
Register
Memory

(c)

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)
  - Subtracts two registers and carry (borrow)

    SBB AX,BX
  - Example:

    SBB BX,CX

    (BX) = 1234H

    (CX) = 0123H

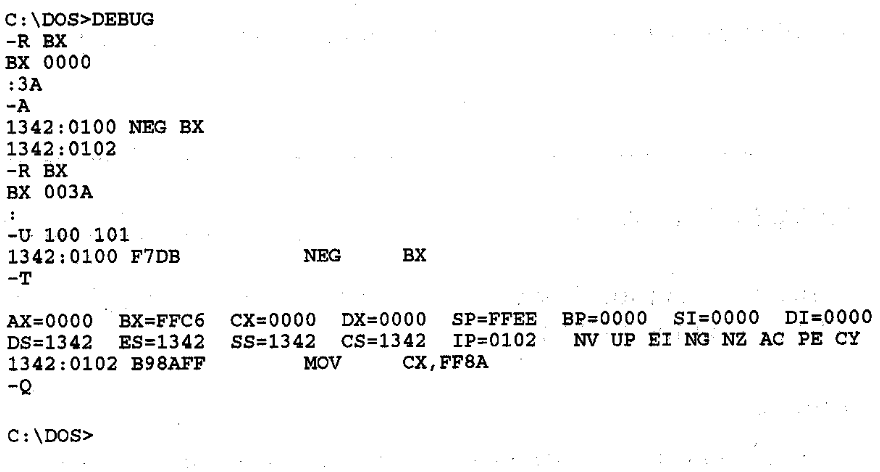
    (CF) = 0

    (BX) - (CX) - (CF) → (BX)

    1234H - 01234H - 0H = 1111H

    (BX) = 1111H

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 pointers
  - Example

    DEC SI

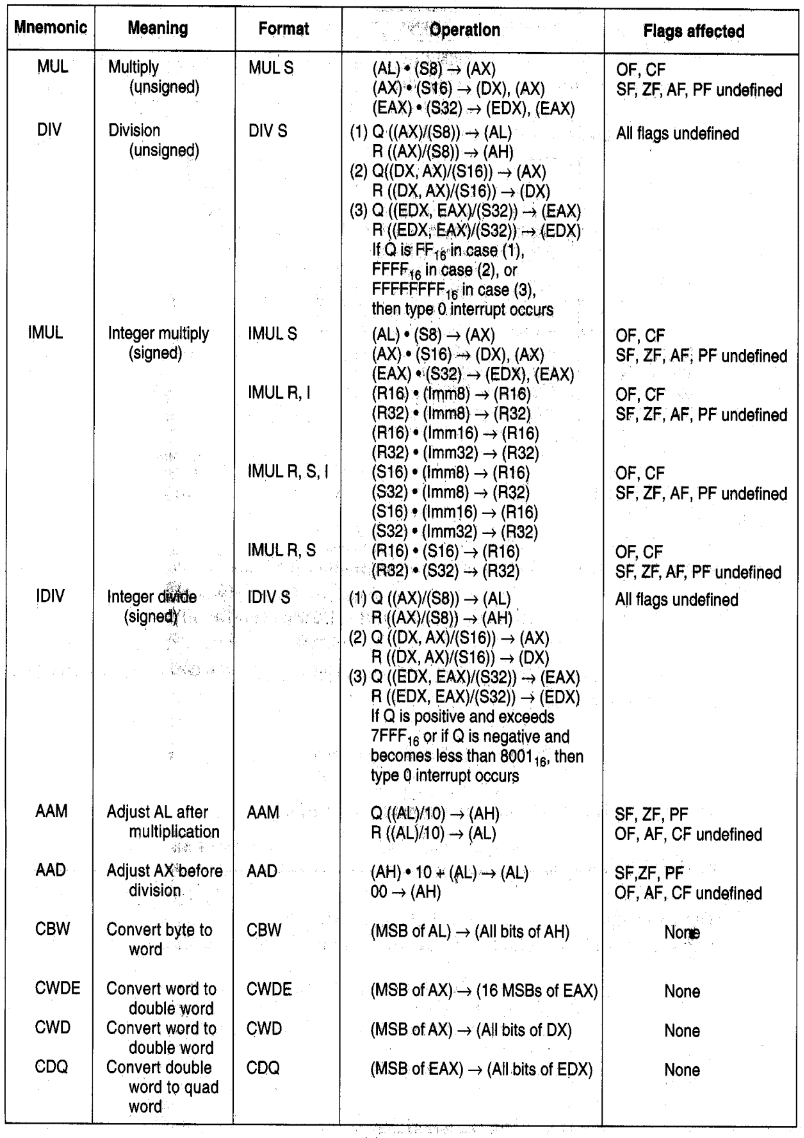
    (SI) = 0FFFH

    (SI) - 1 → SI

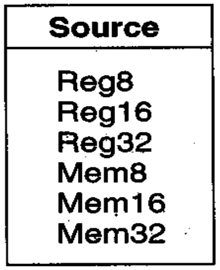
    0FFFFH - 1 = 0FFEH

    (DI) = 0FFEH

Multiplication and Division Instructions

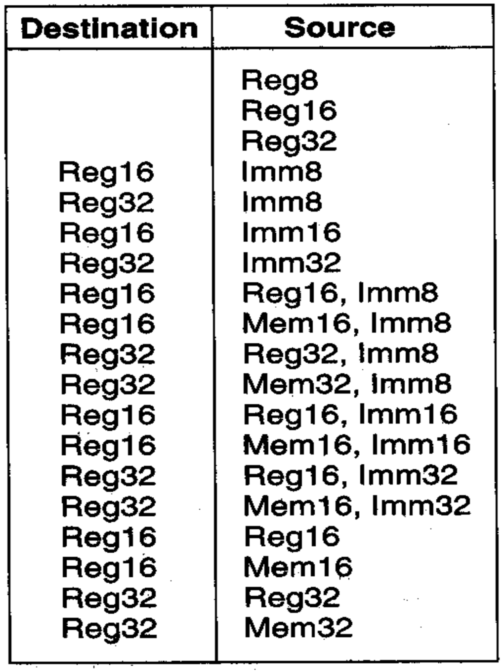


Mnemonic	Meaning	Format	Operation	Flags affected
MUL	Multiply (unsigned)	MUL S	(AL) • (S8) → (AX) (AX) • (S16) → (DX), (AX) (EAX) • (S32) → (EDX), (EAX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q ((AX)/(S8)) → (AL) R ((AX)/(S8)) → (AH) (2) Q((DX, AX)/(S16)) → (AX) R ((DX, AX)/(S16)) → (DX) (3) Q ((EDX, EAX)/(S32)) → (EAX) R ((EDX, EAX)/(S32)) → (EDX) If Q is FF_{16} in case (1), $FFFF_{16}$ in case (2), or $FFFFFFFF_{16}$ in case (3), then type 0 interrupt occurs	All flags undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) • (S8) → (AX) (AX) • (S16) → (DX), (AX) (EAX) • (S32) → (EDX), (EAX)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, I	(R16) • (Imm8) → (R16) (R32) • (Imm8) → (R32) (R16) • (Imm16) → (R16) (R32) • (Imm32) → (R32)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, S, I	(S16) • (Imm8) → (R16) (S32) • (Imm8) → (R32) (S16) • (Imm16) → (R16) (S32) • (Imm32) → (R32)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, S	(R16) • (S16) → (R16) (R32) • (S32) → (R32)	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) (3) Q ((EDX, EAX)/(S32)) → (EAX) R ((EDX, EAX)/(S32)) → (EDX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interrupt occurs	All flags undefined
AAM	Adjust AL after multiplication	AAM	Q ((AL)/10) → (AH) R ((AL)/10) → (AL)	SF, ZF, PF OF, AF, CF undefined
AAD	Adjust AX before 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
CWDE	Convert word to double word	CWDE	(MSB of AX) → (16 MSBs of EAX)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None
CDQ	Convert double word to quad word	CDQ	(MSB of EAX) → (All bits of EDX)	None



Source
Reg8
Reg16
Reg32
Mem8
Mem16
Mem32

(b)



Destination	Source
	Reg8
	Reg16
	Reg32
Reg16	Imm8
Reg32	Imm8
Reg16	Imm16
Reg32	Imm32
Reg16	Reg16, Imm8
Reg16	Mem16, Imm8
Reg32	Reg32, Imm8
Reg32	Mem32, Imm8
Reg16	Reg16, Imm16
Reg16	Mem16, Imm16
Reg32	Reg16, Imm32
Reg32	Mem16, Imm32
Reg16	Reg16
Reg16	Mem16
Reg32	Reg32
Reg32	Mem32

(c)

The 80386, 80486, and Prentium Processors,Triebel
Prof. Yan Luo, UMass Lowell

Multiplication Instructions

- Integer multiply instructions—MUL and IMUL
  - Multiply two unsigned or signed byte, word, or double 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

      (EAX) X (S32)→ (EDX), (EAX) 32-bit product gives 64 bit result

    - Source operand (S) can be an 8-bit, 16-bit, or 32-bit value in a register or memory
      - Other source operand is "implicit" and is AL, AX, or EAX
    - Destination in "implicit"
      - AX assumed to be destination for 16 bit result
      - DX,AX assumed destination for 32 bit result
      - EDX,EAX assumed destination for 64 bit result
    - Only CF and OF flags updated; other undefined

Multiplication Instructions

- Integer multiply instructions—MUL and IMUL
  - Other formats of the signed multiply instruction

IMUL R,I = Register operand times immediate operand; result in the register

Typical operation: (R16) X IMM8 → (R16)

IMUL R,S,I = Source in a register or memory times immediate operand; result in the register

Typical operation: (S32) X IMM8 → (R32)

IMUL R,S = Source time register; result in the register

Typical operation: (R32) X (S32) → (R32)

Multiplication Instruction Example

```
C:\WINDOWS>debug
-a cs:100
106D:0100 mul cl
106D:0102
-r ax
AX 0000
:ff
-r cx
CX 0000
:fe
-u cs:100 101
106D:0100 F6E1          MUL     CL
-r
AX=00FF  BX=0000  CX=00FE  DX=0000  SP=FFEE  BP=0000  SI=0000
    DI=0000
DS=106D  ES=106D  SS=106D  CS=106D  IP=0100   NV UP EI PL NZ
    NA PO NC
106D:0100 F6E1          MUL     CL
-t =cs:100

AX=FD02  BX=0000  CX=00FE  DX=0000  SP=FFEE  BP=0000  SI=0000
    DI=0000
DS=106D  ES=106D  SS=106D  CS=106D  IP=0102   OV UP EI PL NZ
    NA PO CY
106D:0102 86E9          XCHG    CH,CL
```

- **Example: unsigned multiply**

  MUL CL

  (AL) = -1_{10}

  (CL) = -2_{10}

  Expressing in 2's complement

  (AL) = -1 = 11111111_2 = FFH

  (CL) = -2 = 11111110_2 = FEH

  Operation: numbers are treated as unsigned integers

  (AL) X (CL) → (AX)

  255 X 254 = ?

  11111111_2 X 11111110_2 = ?

  = 1111 1101 0000 0010

  (AX) = FD02H

  (CF) = CY → carry from AL to AH

Multiplication Instructions Example

```
C:\WINDOWS>debug
-a cs:100
106D:0100 imul cl
106D:0102
-r ax
AX 0000
:ff
-r cx
CX 0000
:fe
-r
AX=00FF  BX=0000  CX=00FE  DX=0000  SP=FFEE  BP=0000  SI=0000
    DI=0000
DS=106D  ES=106D  SS=106D  CS=106D  IP=0100   NV UP EI PL NZ NA
    PO NC
106D:0100 F6E9          IMUL    CL
-u cs:100 101
106D:0100 F6E9          IMUL    CL
-t =cs:100

AX=0002  BX=0000  CX=00FE  DX=0000  SP=FFEE  BP=0000  SI=0000
    DI=0000
DS=106D  ES=106D  SS=106D  CS=106D  IP=0102   NV UP EI PL NZ NA
    PO NC
106D:0102 86E9          XCHG    CH,CL
-
```

- **Example: multiplying as signed numbers**

  IMUL CL

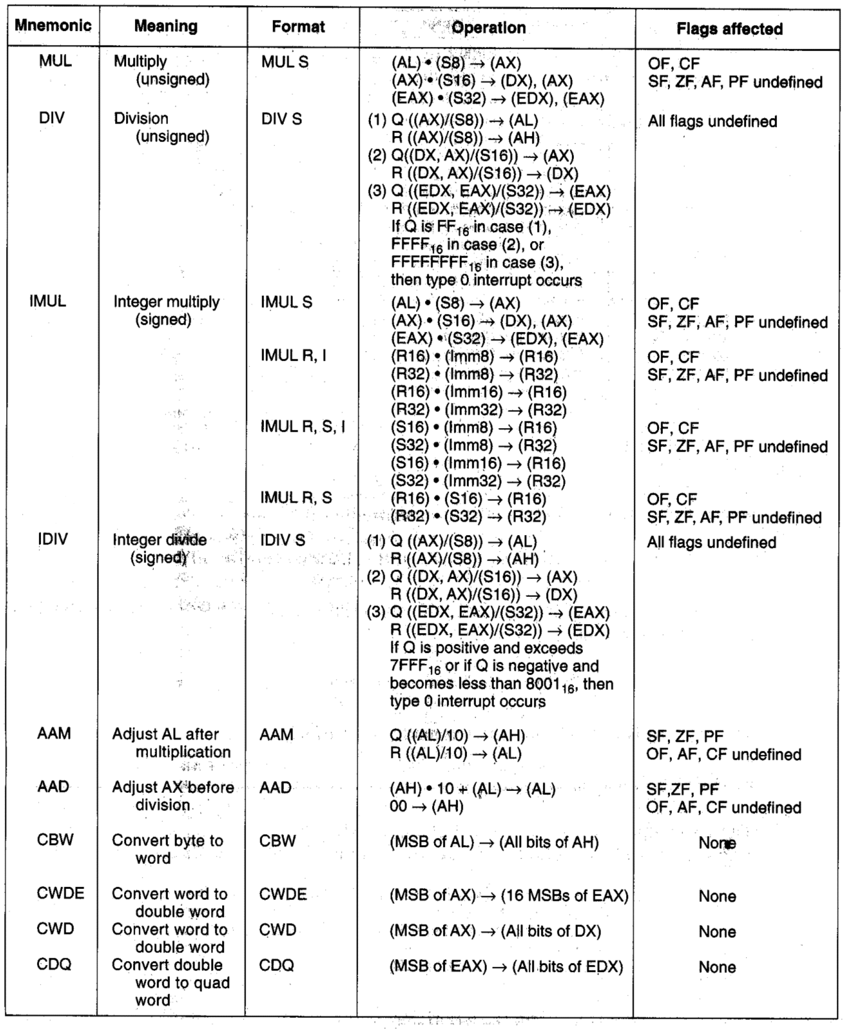
  (AL) = -1_{10}

  (CL) = -2_{10}

  Result

  (-1) X (-2) = +2

Division Instruction



Mnemonic	Meaning	Format	Operation	Flags affected
MUL	Multiply (unsigned)	MUL S	(AL) • (S8) → (AX) (AX) • (S16) → (DX), (AX) (EAX) • (S32) → (EDX), (EAX)	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) (3) Q ((EDX, EAX)/(S32)) → (EAX) R ((EDX, EAX)/(S32)) → (EDX) If Q is FF$_{16}$ in case (1), FFFF$_{16}$ in case (2), or FFFFFFFF$_{16}$ in case (3), then type 0 interrupt occurs	All flags undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) • (S8) → (AX) (AX) • (S16) → (DX), (AX) (EAX) • (S32) → (EDX), (EAX)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, I	(R16) • (Imm8) → (R16) (R32) • (Imm8) → (R32) (R16) • (Imm16) → (R16) (R32) • (Imm32) → (R32)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, S, I	(S16) • (Imm8) → (R16) (S32) • (Imm8) → (R32) (S16) • (Imm16) → (R16) (S32) • (Imm32) → (R32)	OF, CF SF, ZF, AF, PF undefined
		IMUL R, S	(R16) • (S16) → (R16) (R32) • (S32) → (R32)	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) (3) Q ((EDX, EAX)/(S32)) → (EAX) R ((EDX, EAX)/(S32)) → (EDX) If Q is positive and exceeds 7FFF$_{16}$ or if Q is negative and becomes less than 8001$_{16}$, then type 0 interrupt occurs	All flags undefined
AAM	Adjust AL after multiplication	AAM	Q ((AL)/10) → (AH) R ((AL)/10) → (AL)	SF, ZF, PF OF, AF, CF undefined
AAD	Adjust AX before 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
CWDE	Convert word to double word	CWDE	(MSB of AX) → (16 MSBs of EAX)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None
CDQ	Convert double word to quad word	CDQ	(MSB of EAX) → (All bits of EDX)	None

- **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**

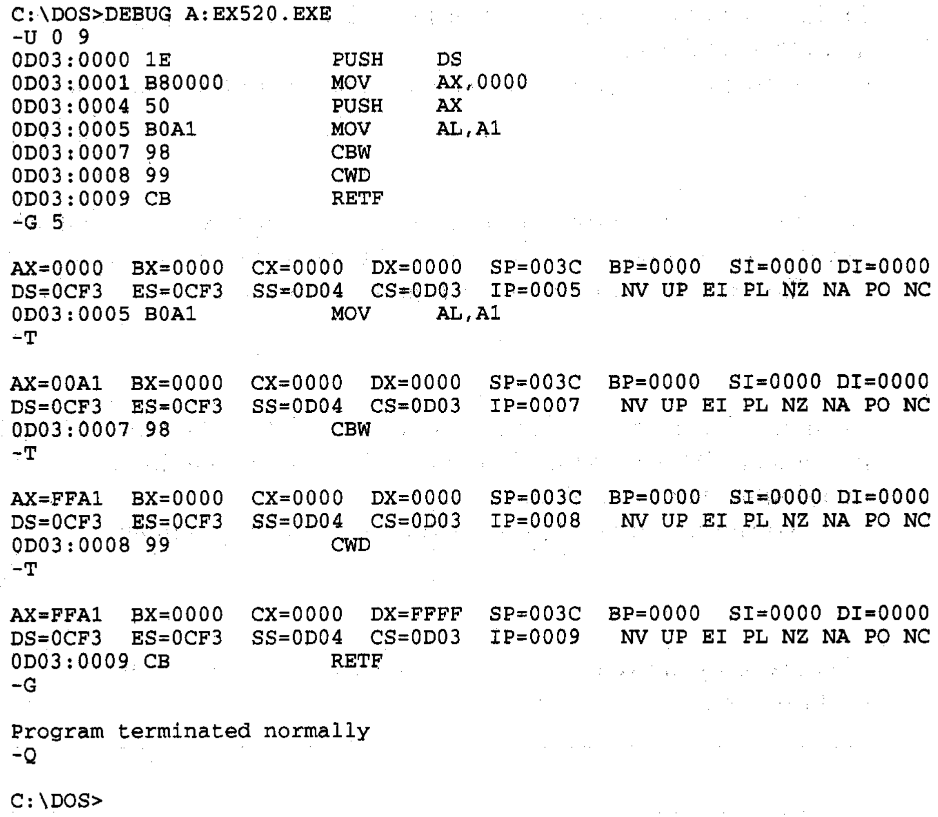
(EDX,EAX) / (S32) → (EAX) =quotient

(EDX) = remainder

- **64 bit dividend in EDX,EAX divided by 32-bit divisor in a register or memory**
- **Quotient of result in EAX**
- **Remainder of result in EDX**

- **Divide error (Type 0) interrupt may occur**

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)
    - **CWDE = convert word to double word**
      (MSB of AX) → (16 MSBs of EAX)
    - **CWD = convert word to double word**
      (MSB of AX) → (all bits of DX)
    - **CDQ = convert word to quad word**
      (MSB of EAX) → (all bits of EDX)
  - **Application:**
    - **To divide two signed 8-bit numbers, the value of the dividend must be sign extended in AX– load into AL and then 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