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

- Used to move (copy) data between:
  - Registers
  - Register and memory
  - Immediate operand to a register or memory

- General format: \( \text{MOV} D,S \)
- Operation: Copies the content of the source to the destination
  \( (S) \rightarrow (D) \)
  - Source contents unchanged
  - Flags unaffected

- Allowed operands
  - Register
  - Memory
    - Accumulator (AH, AL, AX, EAX)
    - Immediate operand (S only)
    - Segment register (Seg-reg)

- Example:
  \[
  \text{MOV} [\text{SUM}], \text{AX}
  \]
  \( (\text{AL}) \rightarrow \text{(address SUM)} \)
  \( (\text{AH}) \rightarrow \text{(address SUM+1)} \)

What is the addressing mode of the destination?
Example of Move Instruction

- **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

- 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=FFFF  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=FFFF  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

- **Example**—Initialization of internal registers with immediate data and address information

```
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

- **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
    
    \[(S) \rightarrow (D); \text{source contents unchanged}\]
  - Sign bit \(\rightarrow\) extended through bit 16 or 32
  - Flags unaffected
  - Examples: `MOVSX EBX,AX`

  Where: \((AX) = FFFFH\)
  - \(S = \text{Reg16}\)
  - \(D = \text{Reg32}\)
  - Sign bit \((AX) = 1\)
  - \(FFFFFFFH \rightarrow (EBX)\)

- **Zero-Extend Move instruction—MOVZX**
  - Operates the same as MOVSX except extends with zeros
  - Example: `MOVZX CX, Byte Pointer [DATA_BYTE]`

  Where: \((DATA_{\_\_BYTE}) = \text{FFH}, S = \text{Mem8, D = Reg16}\)

  \(00FFH \rightarrow (CX)\)

Where might one use these instructions?

Why both sign extend and zero extend?
Exchange Instruction

- Used to exchange the data between two data registers or a data register and memory
- General format:  
  XCHG D,S
- Operation: Swaps the content of the source and destination  
  - Both source and destination change  
    (S) \rightarrow (D)  
    (D) \rightarrow (S)  
  - Flags unaffected
- Special accumulator destination version executes faster
- Examples:  
  XCHG AX,DX  
  (Original value in AX) \rightarrow (DX)  
  (Original value in DX) \rightarrow (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

- 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=FEEF 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=FEEF 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 871B3412 XCHG BX,[1234]
1100:0105 0F 13 34 XCHG BX,[1234]
-T
AX=0000 BX=00FF CX=0000 DX=0000 SP=FEEF 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
1234 1235
1200:1230 AA 11
-Q

c:\dos>

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

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

    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 element
Translate Instruction (example)

- **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

- **Load effective address instruction**
- Used to load the effective address of a pointer from memory into a register
- **General format:**
  \[
  \text{LEA Reg16/32}, \text{EA}
  \]
- **Operation:**
  \[
  (\text{EA}) \rightarrow (\text{Reg16/32})
  \]
  - Source unaffected:
  - Flags unaffected

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA</td>
<td>Load effective address</td>
<td>LEA Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LEA Reg32, EA</td>
<td>(EA) \rightarrow (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LDS</td>
<td>Load register and DS</td>
<td>LDS Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDS Reg32, EA</td>
<td>(EA + 2) \rightarrow (DS)</td>
<td>None</td>
</tr>
<tr>
<td>LSS</td>
<td>Load register and SS</td>
<td>LSS Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSS Reg32, EA</td>
<td>(EA + 2) \rightarrow (SS)</td>
<td>None</td>
</tr>
<tr>
<td>LES</td>
<td>Load register and ES</td>
<td>LES Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LES Reg32, EA</td>
<td>(EA + 2) \rightarrow (ES)</td>
<td>None</td>
</tr>
<tr>
<td>LFS</td>
<td>Load register and FS</td>
<td>LFS Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LFS Reg32, EA</td>
<td>(EA + 2) \rightarrow (FS)</td>
<td>None</td>
</tr>
<tr>
<td>LGS</td>
<td>Load register and GS</td>
<td>LGS Reg16, EA</td>
<td>(EA) \rightarrow (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LGS Reg32, EA</td>
<td>(EA + 2) \rightarrow (GS)</td>
<td>None</td>
</tr>
</tbody>
</table>
Load Full Pointer Instructions

- 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) $\rightarrow$ (Reg16/32)
  
  (EA+2/4) $\rightarrow$ (DS)
  
  **LSS Reg16/32,EA**
  
  (EA) $\rightarrow$ (Reg16/32)
  
  (EA+2/4) $\rightarrow$ (SS)

  - LES, LFS, and LGS operate the same
  
  **LES Reg16/32,EA**  (EA) $\rightarrow$ (Reg16/32),(ES)
  
  **LFS Reg16/32,EA**  (EA) $\rightarrow$ (Reg16/32),(FS)
  
  **LGS Reg16/32,EA**  (EA) $\rightarrow$ (Reg16/32),(GS)

---

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA</td>
<td>Load effective address</td>
<td>LEA Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LEA Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LDS</td>
<td>Load register and DS</td>
<td>LDS Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDS Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LSS</td>
<td>Load register and SS</td>
<td>LSS Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSS Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LES</td>
<td>Load register and ES</td>
<td>LES Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LES Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LFS</td>
<td>Load register and FS</td>
<td>LFS Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LFS Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
<tr>
<td>LGS</td>
<td>Load register and GS</td>
<td>LGS Reg16, EA</td>
<td>(EA) $\rightarrow$ (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LGS Reg32, EA</td>
<td>(EA) $\rightarrow$ (Reg32)</td>
<td>None</td>
</tr>
</tbody>
</table>
Load Full Pointer Instructions (example)

- **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)

- Example (continued)
- State after execution
  
  \[ \text{CS:IP} = 1100:0104 = 11104H \]
  
  \[ 01004H \rightarrow \text{points to next sequential instruction} \]
  
  \[ (\text{DS}) = 1300H \rightarrow \text{defines new data segment} \]
  
  \[ (\text{SI}) = 0020H \]
  
  - Rest of the bits in ESI unaffected
Example—Initialization of internal registers from memory with data and address information

- DS loaded via AX with immediate value using move instructions
  \[
  \text{DATA SEG ADDR} \rightarrow (AX) \rightarrow (DS)
  \]
- Index register SI loaded with move from table
  \[
  (\text{INIT \_TABLE,INIT \_TABLE}) \rightarrow SI
  \]
- DI and ES are loaded with load full pointer instruction
  \[
  (\text{INIT \_TABLE+2,INIT \_TABLE+3}) \rightarrow DI
  (\text{INIT \_TABLE+4,INIT \_TABLE+5}) \rightarrow ES
  \]
- SS loaded from table via AX using move instructions
  \[
  (\text{INIT \_TABLE+6,INIT \_TABLE+7}) \rightarrow AX \rightarrow (SS)
  \]
- Data registers loaded from table with move instructions
  \[
  (\text{INIT \_TABLE+8,INIT \_TABLE+9}) \rightarrow AX
  (\text{INIT \_TABLE+A,INIT \_TABLE+B}) \rightarrow BX
  (\text{INIT \_TABLE+C,INIT \_TABLE+D}) \rightarrow CX
  (\text{INIT \_TABLE+E,INIT \_TABLE+F}) \rightarrow DX
  \]
Addition Instructions.

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:
  \[
  \text{ADD D, S} \\
  (S) + (D) \rightarrow (D) \\
  \text{carry} \rightarrow \text{CF}
  \]
  - Add values in two registers
  - **ADD AX, BX**
    - \((AX) + (BX) \rightarrow (AX) \& \text{CF}\)
  - 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**
    - \((AX) + \text{IMM16} \rightarrow (AX)\)

- **Flags updated based on result**
  - CF, OF, SF, ZF, AF, PF
Addition Instructions (example)

- **Example**
  - ADD AX,BX
  - 
  - \((AX) + (BX) \rightarrow (AX)\)
  - 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 \rightarrow \text{points to next sequential instruction}

- Operation performed
  \((AX) + (BX) = (AX)\)
  \((1100H) + (0ABCH) = 1BBCH\)
  \(= 0011011101111002\)

  \((AX) = 1BBCH\)

  Upper bits of \((AX)\) unchanged

  \((BX) = \text{unchanged}\)

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

- **Add with carry instruction**—ADC
  - ADC format and operation:
    
    ADC D,S
    
    \[(S) + (D) + (CF) \rightarrow (D)\]
  - Full-add operation
  - Add two registers with carry
    
    ADC AX,BX
    
    \[(AX) + (BX) + (CF) \rightarrow (AX) \& CF\]
  - Add register and memory with carry
    
    ADC [DI],AX
    
    \[(DS:DI) + (AX) + (CF) \rightarrow (DS:DI)\]
  - Add immediate operand to a value in a register or memory
    
    ADC AX,100H
    
    \[(AX) + IMM16 + (CF) \rightarrow (AX)\]
  - Same flags updated as ADD

- **Increment instruction**—INC
  - INC format and operation
    
    INC D
    
    \[(D) + 1 \rightarrow (D)\]
  - 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
    - \((DS:SUM) + (AX) \rightarrow (AX)\)
    - \(00CDH + 1234H = 1301H\)
    - \((AX) = 1301H\)
    - \((CF) = \text{unchanged}\)
  - Operation of second instruction
    - \((BL) + IMM8 + (CF) \rightarrow BL\)
    - \(ABH + 05H + 0 = B0H\)
    - \((BL) = B0H\)
    - \((CF) = \text{unchanged}\)
  - Operation of third instruction
    - \((DS:SUM) + 1 \rightarrow (DS:SUM)\)
    - \(00CDH + 1 = 00CEH\)
    - \((SUM) = 00CEH\)
    - \((CF) = \text{unchanged}\)
Examples of Addition Instructions

- Example—Execution of the arithmetic computation

C:\DOS>DEBUG A:EX511.EXE
-U 0 12
OD03:0000 1E  push DS
OD03:0001 B60000 mov AX, 0000
OD03:0004 50      push AX
OD03:0005 B8050D mov AX, OD05
OD03:0008 88DB    mov DS, AX
OD03:000A 030E0000 add AX, [0000]
OD03:000E 80D305 adc BL, 05
OD03:0011 FP060000 inc word ptr [0000]
-G A

AX=OD03 BX=0000 CX=0000 DX=0000 SP=003C BP=0000 SI=0000 DI=0000
DS=OD05 ES=0CF3 SS=0D06 CS=0D03 IP=000A NV UP EI PL NZ NA FO NC
OD03:000A 030E0000 add AX, [0000]
-R AX
AX OD03
:1234
-R BX
BX 0000
:AB
-R F
NV UP EI PL NZ NA FO NC
-E 0 CD 00
-D 0 1
OD05:0000 CD 00
-T

AX=1301 BX=000B CX=0000 DX=0000 SP=003C BP=0000 SI=0000 DI=0000
DS=OD05 ES=0CF3 SS=0D06 CS=0D03 IP=0011 NV UP EI MG NZ AC PO NC
OD03:000E 80D305 adc BL, 05
-T

AX=1301 BX=000B CX=0000 DX=0000 SP=003C BP=0000 SI=0000 DI=0000
DS=OD05 ES=0CF3 SS=0D06 CS=0D03 IP=0011 NV UP EI MG NZ AC PO NC
OD03:0011 FP060000 inc word ptr [0000]
-T

AX=1301 BX=000B CX=0000 DX=0000 SP=003C BP=0000 SI=0000 DI=0000
DS=OD05 ES=0CF3 SS=0D06 CS=0D03 IP=001B NV UP EI PL NZ NA FO NC
OD03:0015 CB retf
-D 0 1
OD05:0000 CE 00
-G

Program terminated normally
-G
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

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>Subtract</td>
<td>SUB D,S</td>
<td>(D) - (S) → (D)</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>SBB</td>
<td>Subtract with borrow</td>
<td>SBB D,S</td>
<td>(D) - (S) - (CF) → (D)</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement by 1</td>
<td>DEC D</td>
<td>(D) - 1 → (D)</td>
<td>OF, SF, ZF, AF, PF</td>
</tr>
<tr>
<td>NEG</td>
<td>Negate</td>
<td>NEG D</td>
<td>0 - (D) → (D)</td>
<td>OF, SF, ZF, AF, PF</td>
</tr>
<tr>
<td>DAS</td>
<td>Decimal adjust for subtraction</td>
<td>DAS</td>
<td>1 → (CF)</td>
<td>OF undefined</td>
</tr>
<tr>
<td>AAS</td>
<td>ASCII adjust for subtraction</td>
<td>AAS</td>
<td></td>
<td>AF, CF, OF, SF, PF, CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Memory</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Register</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
<td></td>
</tr>
<tr>
<td>Accumulator</td>
<td>Immediate</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Immediate</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>Immediate</td>
<td></td>
</tr>
</tbody>
</table>
Subtraction Instructions

- Subtract with borrow instruction—SBB
  - SBB format and operation:
    SBB D,S
    (D) - (S) - (CF) \(\rightarrow\) (D)
  - 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
Subtraction Instructions

- **Negate instruction**—**NEG**
  - **NEG format and operation**
    
    $$\text{NEG D}$$
    
    $$\begin{align*}
    (0) \cdot (D) & \rightarrow (D) \\
    (1) & \rightarrow (CF)
    \end{align*}$$
  
  - **Example**:
    
    $$\text{NEG BX}$$
    
    $$(0) \cdot (BX) \rightarrow (BX)$$
    
    $$0000H - 003AH = 0000H + FFC6H \quad (2\text{'s complement}) = FFC6H$$
    
    $$(BX) = FFC6H; \, CF = 1$$

- **Decrement instruction**—**DEC**
  - **DEC format and operation**
    
    $$\text{DEC D}$$
    
    $$(D) - 1 \rightarrow (D)$$
  
  - **Used to decrement pointers**
  
  - **Example**
    
    $$\text{DEC SI}$$
    
    $$(SI) = 0FFFH$$
    
    $$(SI) - 1 \rightarrow SI$$
    
    $$0FFFH - 1 = 0FEFH$$
    
    $$(DI) = 0FFEH$$
### Multiplication and Division Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MUL</strong></td>
<td>Multiply (unsigned)</td>
<td>MUL S</td>
<td>`(AL)</td>
<td><code>(AX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AX)</code></td>
<td><code>(DAX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(EAX)</code></td>
<td><code>(EDX)</code></td>
</tr>
<tr>
<td><strong>DIV</strong></td>
<td>Division (unsigned)</td>
<td>DIV S</td>
<td>(1) Q</td>
<td><code>(AX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2) Q</td>
<td><code>(AX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(3) Q</td>
<td><code>(EAX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
<tr>
<td><strong>IMUL</strong></td>
<td>Integer multiply (signed)</td>
<td>IMUL S</td>
<td><code>(AL)</code></td>
<td><code>(SX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AX)</code></td>
<td><code>(DAX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(EAX)</code></td>
<td><code>(EDX)</code></td>
</tr>
<tr>
<td><strong>IDIV</strong></td>
<td>Integer divide (signed)</td>
<td>IDIV S</td>
<td>(1) Q</td>
<td><code>(AX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2) Q</td>
<td><code>(AX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(3) Q</td>
<td><code>(EAX)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(AL)</code></td>
<td><code>(AL)</code></td>
</tr>
</tbody>
</table>

**Source**

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg8</td>
<td>Reg16</td>
</tr>
<tr>
<td>Reg16</td>
<td>Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg16</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg16</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg8</td>
<td>Reg16</td>
</tr>
<tr>
<td>Reg16</td>
<td>Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg16</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg16</td>
<td>Mem16, Imm8</td>
</tr>
<tr>
<td>Reg32</td>
<td>Reg32</td>
</tr>
<tr>
<td>Reg32</td>
<td>Reg32</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem32</td>
</tr>
<tr>
<td>Reg16</td>
<td>Mem32</td>
</tr>
<tr>
<td>Reg32</td>
<td>Reg32</td>
</tr>
<tr>
<td>Reg32</td>
<td>Mem32</td>
</tr>
</tbody>
</table>

**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
    \[
    \text{MUL } S = \text{Unsigned integer multiply}
    \]
    \[
    \text{IMUL } S = \text{Signed integer multiply}
    \]
    \[
    (AL) \times (S8) \rightarrow (AX) \quad 8\text{-bit product gives 16 bit result}
    \]
    \[
    (AX) \times (S16) \rightarrow (DX), (AX) \quad 16\text{-bit product gives 32 bit result}
    \]
    \[
    (EAX) \times (S32) \rightarrow (EDX), (EAX) \quad 32\text{-bit product gives 64 bit result}
    \]
  - 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:  
  
  \((R_{16}) \times IMM_{8} \rightarrow (R_{16})\)

  IMUL R,S,I = Source in a register or memory times immediate operand; result in the register
  
  Typical operation:  
  
  \((S_{32}) \times IMM_{8} \rightarrow (R_{32})\)

  IMUL R,S = Source times register; result in the register
  
  Typical operation:  
  
  \((R_{32}) \times (S_{32}) \rightarrow (R_{32})\)
Multiplication Instruction Example

Example: unsigned multiply

MUL CL

(AL) = -1<sub>10</sub>

(CL) = -2<sub>10</sub>

Expressing in 2's complement

(AL) = -1 = 11111111₂ = FFH

(CL) = -2 = 11111110₂ = FEH

Operation: numbers are treated as unsigned integers

(AL) X (CL) → (AX)

255 X 254 = ?

11111111₂ X 11111110₂ = ?

= 1111 1101 0000 0010

(AX) = FD02H

(CF) = CY → carry from AL to AH
Multiplication Instructions Example

- Example: multiplying as signed numbers
  \[ (AL) = -1_{10} \]
  \[ (CL) = -2_{10} \]
  Result
  \[ (-1) \times (-2) = +2 \]
### Division Instruction

**Integer divide instructions—DIV and IDIV**

#### Divide unsigned—DIV S

- **Operations:**
  - $(AX) / (S8) \rightarrow (AL) =$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

- $(DX,AX) / (S16) \rightarrow (AX) =$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

- $(EDX,EAX) / (S32) \rightarrow (EAX) =$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

**Table:**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL</td>
<td>Multiply (unsigned)</td>
<td>MUL S</td>
<td>$(AX) / (S8) \rightarrow (AL)$</td>
<td>OF, CF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>DIV</td>
<td>Division (unsigned)</td>
<td>DIV S</td>
<td>$(AX) / (S16) \rightarrow (DX,AX)$</td>
<td>All flags undefined</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiply (signed)</td>
<td>IMUL S</td>
<td>$(AX) * (S16) \rightarrow (DX,AX)$</td>
<td>OF, CF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer divide (signed)</td>
<td>IDIV S</td>
<td>$(AX) / (S32) \rightarrow (EDX,EAX)$</td>
<td>All flags undefined</td>
</tr>
<tr>
<td>AAM</td>
<td>Adjust AL after multiplication</td>
<td>AAM</td>
<td>$(AL) / (S8) \rightarrow (AH)$</td>
<td>OF, CF, SF, ZF, PF, AF, OF undefined</td>
</tr>
<tr>
<td>AAD</td>
<td>Adjust AX before division</td>
<td>AAD</td>
<td>$(AL) / (S16) \rightarrow (AH)$</td>
<td>SF, ZF, PF, OF, AF, OF undefined</td>
</tr>
<tr>
<td>CBW</td>
<td>Convert byte to word</td>
<td>CBW</td>
<td>$(AL) / (S16) \rightarrow (AH)$</td>
<td>None</td>
</tr>
<tr>
<td>CWDE</td>
<td>Convert word to double word</td>
<td>CWDE</td>
<td>$(MSB of AL) / (S16) \rightarrow (AH)$</td>
<td>None</td>
</tr>
<tr>
<td>CWD</td>
<td>Convert word to double word</td>
<td>CWD</td>
<td>$(MSB of AX) / (S16) \rightarrow (AH)$</td>
<td>None</td>
</tr>
<tr>
<td>CDQ</td>
<td>Convert double word to quad word</td>
<td>CDQ</td>
<td>$(MSB of AX) / (S16) \rightarrow (AH)$</td>
<td>None</td>
</tr>
</tbody>
</table>
Convert Instructions

- **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
    - FFA1H → AX
      CWD sign extends to give FFFFH
    - FFFFH → DX