Chapter 5

8088/8086 Microprocessor Programming
Introduction

5.1 Data-Transfer Instructions
5.2 Arithmetic Instructions
5.3 Logic Instructions
5.4 Shift Instructions
5.5 Rotate Instructions
5.1 Data Transfer Instructions - Move Instruction

- Move instruction
  - Used to move (copy) data between:
    - Registers
    - Register and memory
    - Immediate operand to a register or memory
  - General format:
    MOV D,S
  - Operation: Copies the content of the source to the destination
    \[(S) \rightarrow (D)\]
    - 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) \rightarrow (address SUM)
  (AH) \rightarrow (address SUM+1)
5.1 Data Transfer Instructions - 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
5.1 Data Transfer Instructions - Move Instruction

- Example (continued)
- State after execution

CS:IP = 0100:0102 = 01102H
01002H \rightarrow \text{points to next sequential instruction}

(CS) = 0100H
(DX) = 0100H \rightarrow \text{Value in CS copied into DX}

Value in CS unchanged
5.1 Data Transfer Instructions - **Move Instruction**

- **Debug execution example**
  
  ```
  MOV CX,[20]
  DS = 1A00
  (DS:20) = AA55H
  (1A00:20) → (CX)
  ```

  How could you verify loading of this data?

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

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

1. What addressing modes are in use in this program?
5.1 Data Transfer Instructions- Exchange Instruction

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

1. Why do you think this is known as a “complex instruction?”
2. How else could this operation be performed?
3. What is the benefit of using XCHG?
5.1 Data Transfer Instructions: Exchange Instruction

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

  What is the general logical address of the destination operand?

- State before fetch and execution
  
  CS:IP = 1100:0101 = 11101H
  Move instruction code = 871E3412H
  (01104H,01103H) = 1234H = SUM
  (DS) = 1200H
  (BX) = 11AA
  (DS:SUM) = (1200:1234) = 00FFH

  What is this type data organization called?
5.1 Data Transfer Instructions - Exchange Instruction

- Example (continued)
- State after execution
  
  \[ \text{CS:IP} = 1100:0105 = 11105H \]
  
  \[ 11005H \rightarrow \text{points to next sequential instruction} \]

- Register updated
  
  \( (BX) = 00FFH \)

- Memory updated
  
  \( (1200:1234) = AAH \)
  
  \( (1200:1235) = 11H \)
5.1 Data Transfer Instructions - Exchange Instruction

- Debug execution of example

C:\DOS>DEBUG
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE 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=FFFE 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 0Q
-U 101 104
1100:0101 871E3412 XCHG BX,[1234]
-T
AX=0000 BX=00FF CX=0000 DX=0000 SP=FFFE 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
-D 1234 1235
1200:1230 AA 11
-Q
C:\DOS>

Write a dump command?
5.1 Data Transfer Instructions - Translate Instruction

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:
  \((DS)0 = \text{Points to the active data segment}\)
  \((BX) = \text{Offset to the first element in the table}\)
  \((AL) = \text{Displacement to the element of the table that is to be accessed}\)*

*8-bit value limits table size to 256 elements

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLAT</td>
<td>Translate</td>
<td>XLAT</td>
<td>((AL) + (BX) + (DS)0 \rightarrow (AL))</td>
<td>None</td>
</tr>
</tbody>
</table>
5.1 Data Transfer Instructions - Translate Instruction

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

- Example
  XLAT
  (DS) = 0300H
  (BX) = 0100H
  (AL) = 3FH → 6FH = ? (Question mark)
5.1 Data Transfer Instructions - Load Effective Address and Load Full Pointer Instructions

- Load effective address instruction
  - Used to load an address pointer offset from memory into a register
  - General format:
    LEA Reg16,EA
  - Operation:
    \((EA) \rightarrow (\text{Reg16})\)
    - Source unaffected:
    - Flags unaffected

- Load full pointer
  - Used to load a full address pointer from memory into a segment register and a register
    - Segment base address
    - Offset
  - General format and operation for LDS
    LDS Reg16,EA
    \((EA) \rightarrow (\text{Reg16})\)
    \((EA+2) \rightarrow (\text{DS})\)
  - 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
5.1 Data Transfer Instructions - Load Effective Address and Load Full Pointer Instructions

- Example (continued)
- State after execution

CS:IP = 1100:0104 = 11104H

0104H → points to next sequential instruction

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

(SI) = 0020H → defines new offset into DS
5.2 Data Transfer Instructions - Load Effective Address and Load Full Pointer Instructions

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
  
  \[ (\text{INIT TABLE}, \text{INIT TABLE}+1) \rightarrow SI \]

- DI and ES are loaded with load full pointer instruction
  
  \[ (\text{INIT TABLE}+2, \text{INIT TABLE}+3) \rightarrow DI \]
  \[ (\text{INIT TABLE}+4, \text{INIT TABLE}+5) \rightarrow ES \]

- SS loaded from table via AX using move instructions
  
  \[ (\text{INIT TABLE}+6, \text{INIT TABLE}+7) \rightarrow AX \rightarrow (SS) \]

- Data registers loaded from table with move instructions
  
  \[ (\text{INIT TABLE}+8, \text{INIT TABLE}+9) \rightarrow AX \]
  \[ (\text{INIT TABLE}+A, \text{INIT TABLE}+B) \rightarrow BX \]
  \[ (\text{INIT TABLE}+C, \text{INIT TABLE}+D) \rightarrow CX \]
  \[ (\text{INIT TABLE}+E, \text{INIT TABLE}+F) \rightarrow DX \]

```assembly
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]
```
5.2 Arithmetic Instructions- 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:
  ADD D,S
  (S) + (D) → (D)
  - Add values in two registers
    ADD AX,BX
    (AX) + (BX) → (AX)
  - Add a value in memory and a value in a register
    ADD [DI],AX
    (DS:DI) + (AX) → (DS:DI)
  - Add an immediate operand to a value in a register or memory
    ADD AX,100H
    (AX) + IMM16 → (AX)
- Flags updated based on result
  - CF, OF, SF, ZF, AF, PF
5.2 Arithmetic Instructions - Addition Instructions

- Example
  ADD AX,BX
  \((AX) + (BX) \rightarrow (AX)\)
  - Word-wide register to register add
  - Half adder operation
- State before fetch and execution
  \(CS:IP = 1100:0100 = 11100H\)
  ADD machine code = 03C3H
  \((AX) = 1100H\)
  \((BX) = 0ABCH\)
  \((DS) = 1200H\)
  \((1200:0000) = 12000H = XXXX\)
5.2 Arithmetic Instructions- Addition Instructions

- Example (continued)
- State after execution

\[
\text{CS:IP = 1100:0102 = 11102H}
\]

11102H \rightarrow \text{points to next sequential instruction}

- Operation performed

\[(\text{AX}) + (\text{BX}) \rightarrow (\text{AX})\]

\[(1100H) + (0ABCH) \rightarrow 1BBCH\]

\[(\text{AX}) = 1BBCH\]

\[= 000110111011100_2\]

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

- Impact on flags

- CF = 0 (no carry resulted)
- ZF = 0 (not zero)
- SF = 0 (positive)
- PF = 0 (odd parity)—parity flag is only based on the bits of the least significant byte
5.2 Arithmetic Instructions - Addition Instructions

- Add with carry instruction — ADC
  - ADC format and operation:
    ADC D,S
  - Full-add operation
  - Used for extended addition
- Add two registers with carry
  ADC AX, BX
  \((AX) + (BX) + (CF) \rightarrow (AX)\)
- 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 (addresses)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Addition</td>
<td>ADD D, S</td>
<td>((S) + (D) \rightarrow (D))</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>ADC</td>
<td>Add with carry</td>
<td>ADC D, S</td>
<td>((S) + (D) + (CF) \rightarrow (D))</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>INC</td>
<td>Increment by 1</td>
<td>INC D</td>
<td>((D) + 1 \rightarrow (D))</td>
<td>OF, SF, ZF, AF, PF</td>
</tr>
<tr>
<td>AAA</td>
<td>ASCII adjust for addition</td>
<td>AAA</td>
<td>AF, CF</td>
<td>OF, SF, ZF, PF, PF undefined</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal adjust for addition</td>
<td>DAA</td>
<td>SF, ZF, AF, PF, CF, OF, undefined</td>
<td></td>
</tr>
</tbody>
</table>
5.2 Arithmetic Instructions - Addition Instructions

Example—Arithmetic computations

- Initial state:
  - (AX) = 1234H
  - (BL) = ABH
  - (SUM) = 00CDH
  - (CF) = 0

- Operation of first instruction
  - (DS:SUM) + (AX) \rightarrow (AX)
  - 00CDH + 1234H = 1301H
  - (AX) = 1301H
  - (CF) = unchanged

- Operation of second instruction
  - (BL) + IMM8 + (CF) \rightarrow BL
  - ABH + 05H + 0 = B0H
  - (BL) = B0H
  - (CF) = unchanged

- Operation of third instruction
  - (DS:SUM) + 1 \rightarrow (DS:SUM)
  - 00CDH + 1 = 00CEH
  - (SUM) = 00CEH
  - (CF) = unchanged

1. Does the column (SUM) stand for a value in a register code memory, or a storage location in memory?
2. Why is the operand of the INC instruction preceded by WORD PTR?
5.2 Arithmetic Instructions - Addition Instructions

Example—Execution of the arithmetic computation sequence

```
C:\DOS>DEBUG A:EX511.EXE
-U 0 12
0D03:0000 1E        PUSH DS
0D03:0001 B80000    MOV AX,0000
0D03:0004 0005      MOV AX,0D05
0D03:0005 B8050D    MOV DS,AX
0D03:0006 88D8      MOV DS,AX
0D03:000A 01060000   ADD AX,[0000]
0D03:000E 80D305    ADC BL,05
0D03:0011 FF060000   INC WORD PTR [0000]
```

1. What is the value of SUM?
2. What is logical address is accessed in memory?
5.2 Arithmetic Instructions- Subtraction Instructions

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

- **Subtract Instruction—SUB**
  - **SUB format and operation:**
    
    ```
    SUB D,S
    (D) - (S) → (D)
    ```

- Flags updated based on result
  - **CE, OF, SF, ZF, AF, PF**
5.2 Arithmetic Instructions - Subtraction Instructions

- **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?**
5.2 Arithmetic Instructions- Subtraction Instructions

- Negate instruction—NEG
  - 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

- Decrement instruction—DEC
  - DEC format and operation
    
    DEC D
    (D) - 1 \rightarrow (D)
  
  - Used to decrement pointer—addresses
  
  - Example
    
    DEC SI
    (SI) = 0FFFH
    (SI) - 1 \rightarrow SI
    
    0FFFH - 1 = 0FFEH
    (DI) = 0FFEH
5.2 Arithmetic Instructions- Multiplication Instructions

- Integer multiply instructions—MUL and IMUL
- Multiply two unsigned or signed byte or word operands
- General format and operation
  MUL S = Unsigned integer multiply
  IMUL S = Signed integer multiply
  (AL) X (S8) \rightarrow (AX) 8-bit product gives 16-bit result
  (AX) X (S16) \rightarrow (DX),(AX) 16-bit product gives 32 bit result

### Table of Arithmetic 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>MUL</td>
<td>Multiply (unsigned)</td>
<td>MUL S</td>
<td>(AL) \cdot (S8) \rightarrow (AX) (AX) \cdot (S16) \rightarrow (DX),(AX)</td>
<td>OF, CF</td>
</tr>
<tr>
<td>DIV</td>
<td>Division (unsigned)</td>
<td>DIV S</td>
<td>(1) Q(AX)(S8) \rightarrow (AL) R(AX)(S8) \rightarrow (AH)</td>
<td>SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2) Q((DX,AX)(S16)) \rightarrow (AX) R((DX,AX)(S16)) \rightarrow (DX)</td>
<td>OF, SF, ZF, AF, PF, CF undefined</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiply (signed)</td>
<td>IMUL S</td>
<td>(AL) \cdot (S8) \rightarrow (AX) (AX) \cdot (S16) \rightarrow (DX),(AX)</td>
<td>SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer divide (signed)</td>
<td>IDIV S</td>
<td>(1) Q((AX)(S8)) \rightarrow (AL) R((AX)(S8)) \rightarrow (AH)</td>
<td>OF, SF, ZF, AF, PF, CF undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2) Q((DX,AX)(S16)) \rightarrow (AX) R((DX,AX)(S16)) \rightarrow (DX)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Q is FF16 in case (1) or FFF16 in case (2), then type 0 interrupt occurs</td>
<td></td>
</tr>
<tr>
<td>AAM</td>
<td>Adjust AL for multiplication</td>
<td>AAM</td>
<td>Q((AL)/10) \rightarrow (AH) R((AL)/10) \rightarrow (AL)</td>
<td>SF, ZF, PF</td>
</tr>
<tr>
<td>AAD</td>
<td>Adjust AX for division</td>
<td>AAD</td>
<td>(AH) \cdot 10 + (AL) \rightarrow (AL) 00 \rightarrow (AH)</td>
<td>OF, AF, CF undefined</td>
</tr>
<tr>
<td>CBW</td>
<td>Convert byte to word</td>
<td>CBW</td>
<td>(MSB of AL) \rightarrow (All bits of AH)</td>
<td>SF, ZF, PF</td>
</tr>
<tr>
<td>CWD</td>
<td>Convert word to double word</td>
<td>CWD</td>
<td>(MSB of AX) \rightarrow (All bits of DX)</td>
<td>OF, AF, CF undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

- 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

- Example:

MUL CL

(AL) = \(-1_{10}\)

(CL) = \(-2_{10}\)

Expressing in 2's complement

(AL) = \(-1 = 11111111_2 = FFH\)

(CL) = \(-2 = 11111110_2 = FEH\)

Operation:

(AL) \times (CL) \rightarrow (AX)

11111111_2 \times 11111110_2 = 111110100000010

(AX) = FD02H
5.2 Arithmetic Instructions - 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>MUL</td>
<td>Multiply (unsigned)</td>
<td>MUL S</td>
<td>(AL) · (S8) → (AX)</td>
<td>OF, CF</td>
</tr>
<tr>
<td>DIV</td>
<td>Division (unsigned)</td>
<td>DIV S</td>
<td>(AX) · (S16) → (DX), (AX)</td>
<td>SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiply (signed)</td>
<td>IMUL S</td>
<td>(AX) · (S16) → (AX)</td>
<td>OF, CF</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer divide (signed)</td>
<td>IDIV S</td>
<td>(AX) · (S16) → (DX), (AX)</td>
<td>SF, ZF, AF, PF undefined</td>
</tr>
</tbody>
</table>

- **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**
5.2 Arithmetic Instructions - 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)
  - CWD = convert word to double word (MSB of AX) → (all bits of DX)

- **Application:**
  - To divide two signed 8-bit numbers, the value of the dividend must be sign extended in AX
    - Load into AL
    - Use CBW to sign extend to 16 bits

- **Example**
  - A1H → AL
    - CBW sign extends to give FFA1H → AX
    - CWD sign extends to give FFFFH → DX
5.3 Logic Instructions—Available Instructions

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:**
  
  \[
  \text{AND D,S} \quad \text{(S)} \land (D) \rightarrow (D)
  \]
  
  - Logical AND of values in two registers
    \[
    \text{AND AX,BX} \quad (AX) \land (BX) \rightarrow (AX)
    \]
  - Logical AND of a value in memory and a value in a register
    \[
    \text{AND [DI],AX} \quad (DS:DI) \land (AX) \rightarrow (DS:DI)
    \]
  - Logical AND of an immediate operand with a value in a register or memory
    \[
    \text{AND AX,100H} \quad (AX) \land \text{IMM16} \rightarrow (AX)
    \]

- Flags updated based on result
  - CF, OF, SF, ZF, PF
  - AF undefined

---

1. Describe the AND operations.
2. Describe the OR operations.
3. Describe the XOR operations.
4. Describe the NOT operations.
5.3 Logic Instructions - Example

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV AL,01010101B</td>
<td>01010101</td>
</tr>
<tr>
<td>AND AL,00011111B</td>
<td>00010101</td>
</tr>
<tr>
<td>OR AL,11000000B</td>
<td>11010101</td>
</tr>
<tr>
<td>XOR AL,00001111B</td>
<td>11011010</td>
</tr>
<tr>
<td>NOT AL</td>
<td>00100101</td>
</tr>
</tbody>
</table>

AX=0055 BX=0000 CX=0000 DX=0000 SP=FTEE 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=FTEE 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 00c0 OR AL, CO
-T

AX=00d5 BX=0000 CX=0000 DX=0000 SP=FTEE 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=FTEE 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 06d0 NOT AL
-T

AX=0025 BX=0000 CX=0000 DX=0000 SP=FTEE 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]
-T

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
  - $1 \text{ AND } 0 \rightarrow 0$; $0 \text{ and } 0 \rightarrow 0$
  - A 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
  
  \[
  \text{AND AX,000FH} \\
  (AX) = FFFFFFFF \\
  \text{IMM16 AND (AX) } \rightarrow (AX) \\
  000FH \text{ AND FFFFH} = 0000000000001111_2 \text{ AND } 1111111111111111_2 \\
  = 0000000000001111_2 \\
  = 000FH
  \]

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

\[
(\text{AL}) = b7b6b5b4b3b2b1b0?
\]
5.3 Logic Instructions - Mask Application

- OR operation can be used to set a bit or bits of a byte or word to 1
  - X OR 0 $\rightarrow$ X; result is unchanged
  - X 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

General Operation:
(AL) = XXXXXXXX$_2$ OR 00010000$_2$ = XXX1XXXX$_2$

1. What is CONTROL_FLAGS?
2. What is it relative to?
5.4 Shift Instructions - Available Instructions

- Variety of shift instructions provided
  - **SAL/SHL** → Shift arithmetic left/shift logical left
  - **SHR** → Shift logical right
  - **SAR** → Shift arithmetic right

- Perform a variety of shift left and shift right operations on the bits of a destination data operand

- Basic shift instructions—SAL/SHL, SHR, SAR
  - Destination may be in either a register or a storage location in memory

- Shift count may be:
  1= one bit shift
  CL = 1 to 255 bit shift

- Flags updated based on result
  - CF, SF, ZF, PF
  - AF undefined
  - OF undefined if Count ≠ 1

### 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 if count ≠ 1
SHR | Shift logical right | SHRD, 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 if count ≠ 1
SAR | Shift arithmetic right | SAR D, Count | Shift the (D) right by the number of bit positions equal to Count and fill the vacated bit positions on the left with the original most significant bit | SF, ZF, PF, AF undefined if count ≠ 1

### Destination | Count
---|---
Register | 1
Register | CL
Memory | 1
Memory | CL
5.4 Shift Instructions- Operation of the SAL/SHL Instruction

- SAL/SHL instruction operation
  - Typical instruction—count of 1
    SHL AX,1
  - Before execution
    Dest = (AX) = 1234H
    = 0001001000110100₂
    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
    = 0010010001101000₂
    CF = 0
  - Conclusion
    - MSB has been isolated in CF and can be acted upon by control flow instruction—conditional jump
    - Result has been multiplied by 2

Note: Signed or unsigned data answer is the same
5.4 Shift Instructions- Operation of the SHR Instruction

SHR instruction operation
- Typical instruction—count in CL
  SHR AX,CL
- Before execution
  Dest  = (AX) = 1234H = 4660_{10}
  = 0001001000110100_{2}
  Count = (CL) = 02H
  CF = X
- Operation
  - The value in all bits of AX are shifted right two bit positions
  - Emptied MSBs are filled with 0s
  - Value shifted out of LSB goes to carry flag
- After execution
  Dest  = (AX) = 048DH = 1165_{10}
  = 0000010010001101_{2}
  CF = 0
- Conclusion
  - Bit 1 has been isolated in CF and can be acted upon by control flow instruction—conditional jump
  - Result has been divided by 4
    - 4 X 1165 = 4660

Note: processes unsigned data
5.4 Shift Instructions- Operation of the SAR Instruction

- SAR instruction operation
  - Typical instruction—count in CL
    SAR AX, CL
  - Before execution—arithmetic implies signed numbers
    Dest = (AX) = 091AH = 0000100100011010₂ = +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 = 0000001001000110₂ = +582
  CF = 1

- Conclusion
  - Bit 1 has been isolated in CF and can be acted upon by control flow instruction— conditional jump
  - Result has been signed extended
  - Result value has been divided by 4 and rounded to integer
    - 4 X +582 = +2328

Note: processed data treated as signed number
5.4 Shift Instructions- SAR Instruction Execution

- Debug execution of example

```
C:\DOS>DEBUG
-A
1342:0100 SAR AX,CL
1342:0102
-R AX
AX 0000
:091A
-R CX
CX 0000
:2
-R F
NV UP EI PL NZ NA PO NC -
-T
AX=0246 BX=0000 CX=0002 DX=0000 SP=FEEE BP=0000 SI=0000 DI=0000
DS=1342 ES=1342 SS=1342 CS=1342 IP=0102 NV UP EI PL NZ AC PO CY
1342:0102 B98AFF MOV CX,FF8A
-Q
C:\DOS>
```
5.4 Shift Instructions - Application

- Application – Isolating a bit of a byte of data in memory in the carry flag
  - Example:
    - Instruction sequence
      MOV AL, [CONTROL_FLAGS]
      MOV CL, 04H
      SHR AL, CL
    - Before execution
      (CONTROL_FLAGS) = B7B6B5B4B3B2B1B0
    - After execution
      (AL) = 0000B7B6B5B4
      (CF) = B3
5.5 Rotate Instructions - Available Instructions

- 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

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROL</td>
<td>Rotate left</td>
<td>ROL D,Count</td>
<td>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.</td>
<td>CF OF undefined if count $\neq 1$</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate right</td>
<td>ROR D,Count</td>
<td>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.</td>
<td>CF OF undefined if count $\neq 1$</td>
</tr>
<tr>
<td>RCL</td>
<td>Rotate left through carry</td>
<td>RCL D,Count</td>
<td>Same as ROL except carry is attached to (D) for rotation.</td>
<td>CF OF undefined if count $\neq 1$</td>
</tr>
<tr>
<td>RCR</td>
<td>Rotate right through carry</td>
<td>RCR D,Count</td>
<td>Same as ROR except carry is attached to (D) for rotation.</td>
<td>CF OF undefined if count $\neq 1$</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Destination</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>1</td>
</tr>
<tr>
<td>Register</td>
<td>CL</td>
</tr>
<tr>
<td>Memory</td>
<td>1</td>
</tr>
<tr>
<td>Memory</td>
<td>CL</td>
</tr>
</tbody>
</table>

(b)
5.5 Rotate Instructions- Operation of the ROL Instruction

- ROL instruction operation
- Typical instruction—count of 1
  ROL AX,1
- Before execution
  Dest = (AX) = 1234H
  = 0001 0010 0011 0100₂
  Count = 1
  CF = 0
- Operation
  - The value in all bits of AX are rotated left one bit position
  - Value rotated out of the MSB is reloaded at LSB
  - Value rotated out of MSB is copied to carry flag
- After execution
  Dest = (AX) = 2468H
  = 0010 0100 0110 1000₂
  CF = 0
5.5 Rotate Instructions- Operation of the ROR Instruction

- **ROR instruction operation**
  - **Typical instruction**—count in CL
    ROR AX, CL
  - **Before execution**
    Dest = (AX) = 1234H = 0001001000110100
    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
    CF = 0
  - **Conclusion:**
    - Note that the position of hex characters in AX have been rearranged
5.5 Rotate Instructions - Operation of the RCL Instruction

- RCL instruction operation
- Typical instruction—count in CL
  RCL BX,CL
- Before execution
  Dest = (BX) = 1234H = 0001001000110100
  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
  CF = 1
5.5 Rotate Instructions- RCR Example

• RCR instruction debug execution example
  • Instruction—count in CL
    RCR BX,CL
  • Before execution
    Dest = (BX) = 1234H = 0001001000110100<sub>2</sub>
    Count = 04H
    CF = 0
  • After execution
    Dest = (BX) = 8123H = 1000000100100011<sub>2</sub>
    CF = 0
5.5 Rotate Instructions- Application

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

MOV AL,[HEX\_DIGITS]
MOV BL,AL
MOV CL,04H
ROR BL,CL
AND AL,0FH
AND BL,0FH
ADD AL,BL