Chapter 5

Real-Mode 80386DX Microprocessor Programming 1
Part 2
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
Logic Instructions

- AND → Logical AND
- OR → Logical inclusive-OR
- XOR → Logical exclusive-OR
- NOT → Logical NOT

*Logical AND Instruction—AND*

**AND format and operation:**

- **AND D,S**
  
  
  \((S) \text{ AND } (D) \rightarrow (D)\)

  - Logical AND of values in two registers
    
    AND AX,BX
    
    \((AX) \text{ AND } (BX) \rightarrow (AX)\)

  - Logical AND of a value in memory and a
    
    AND [DI],AX
    
    \((DS:DI) \text{ AND } (AX) \rightarrow (DS:DI)\)

  - Logical AND of an immediate operand with a value in a register or memory
    
    AND AX,100H
    
    \((AX) \text{ AND IMM16} \rightarrow (AX)\)

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

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical AND</td>
<td>AND D,S</td>
<td>((S) \cdot (D) \rightarrow (D))</td>
<td>OF, SF, ZF, PF, CF, AF undefined</td>
</tr>
<tr>
<td>OR</td>
<td>Logical Inclusive-OR</td>
<td>OR D,S</td>
<td>((S) + (D) \rightarrow (D))</td>
<td>OF, SF, ZF, PF, CF, AF undefined</td>
</tr>
<tr>
<td>XOR</td>
<td>Logical Exclusive-OR</td>
<td>XOR D,S</td>
<td>((S) \oplus (D) \rightarrow (D))</td>
<td>OF, SF, ZF, PF, CF, AF undefined</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT</td>
<td>NOT D</td>
<td>((\bar{D}) \rightarrow (D))</td>
<td>None</td>
</tr>
</tbody>
</table>

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

The 80386, 80486, and Prentium Processors, Triebel

Prof. Yan Luo, UMass Lowell
### Logic Instructions - Example

<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>

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

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 0CC0 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 F6D0 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] DS:0000=20CD
-Q
Logic Instructions- Mask Application

Mask—to clear a bit or bits of a byte, word, or double word to 0.

- AND operation can be used to perform the mask operation
- 1 AND 0 \( \rightarrow \) 0; 0 and 0 \( \rightarrow \) 0
  - A bit or bits are masked by ANDing with 0
- 1 AND 1 \( \rightarrow \) 1; 0 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
  \[
  \text{AND AX,000FH} \\
  (AX) = \text{FFFF} \\
  \text{IMM16 AND (AX) \rightarrow (AX)} \\
  000FH \text{ AND } FFFFH = 0000000000001111_2 \text{ AND } 1111111111111111_2 \\
  = 0000000000001111_2 \\
  = 000FH
  \]

- OR operation can be used to set a bit or bits of a byte, word, or double 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
  \[
  \text{MOV AL,[CONTROL_FLAGS]} \\
  \text{OR AL, 10H ;00010000 sets fifth bit—b4} \\
  \text{MOV [CONTROL_FLAGS],AL}
  \]
Shift Instructions

- SAL/SHL → Shift arithmetic left/shift logical left
- SHR → Shift logical right
- SAR → Shift arithmetic right
- SHLD → Double precision shift left
- SHRD → Double precision shift 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
    IMM8 = 1 to 255 bit shift
- Flags updated based on result
  - CF, SF, ZF, PF
  - AF undefined
  - OF undefined if Count ≠ 1
Operation of the SAL/SHL Instruction

- Typical instruction—count of 1
  SHL AX,1
- Before execution
  Dest = (AX) = 1234H = 0001 0010 0011 0100₂, 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 = 0010 0100 0110 1000₂, 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
Operation of the SHR Instruction

- Typical instruction—count in CL
  SHR AX, CL
- Before execution
  Dest = (AX) = 1234H = 4660_{10} = 0001 00100011 0100_2
  Count = 02H , CF = X
- Operation
  - The value in all bits of AX are shifted right two bit positions
  - Emptied MSBs are filled with 0s
  - Values shifted out of LSBs go to carry flag
- After execution
  Dest = (AX) = 048DH = 1165_{10} = 0000 0100 1000 1101_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
    - $4 \times 1164 = 4660$
Operation of the SAR Instruction

- Typical instruction—count in CL
  SAR AX,CL
- Before execution
  Dest = (AX) = 091AH = 0000100100011010₂ = +2330, Count = 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—sign maintained
  - Values shifted out of LSBs 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
SAR Instruction Execution

- Debug execution of SAR example

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

C:\DOS>
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 executing 1st instruction
    - (AL) = B7B6B5B4B3B2B1B0
  - After executing 2nd instruction
    - (CL) = 04H
  - After executing 3rd instruction
    - (AL) = 0000B7B6B5B4
    - (CF) = B3
Rotate Instructions

- Variety of rotate instruction 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
    - IMM8 = 1 to 255 bit rotate
- Flags updated based on result
  - CF
  - OF undefined if Count ≠ 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 OR undefined if Count ≠ 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 OR undefined if Count ≠ 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 OR undefined if Count ≠ 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 OR undefined if Count ≠ 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>1 CL</td>
</tr>
<tr>
<td>Register</td>
<td>1 IMM8</td>
</tr>
<tr>
<td>Memory</td>
<td>1 CL</td>
</tr>
<tr>
<td>Memory</td>
<td>1 IMM8</td>
</tr>
</tbody>
</table>

The 80386, 80486, and Pentium Processors, Triebel
Prof. Yan Luo, UMass Lowell
Operation of the ROL Instruction

- 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 copied to carry flag
- After execution
  Dest = (AX) = 2468H = 0010010011010000₂
  CF = 0
Operation of the ROR Instruction

- Typical instruction—count in CL
  ROR AX, CL
- Before execution
  Dest = (AX) = 1234H
  = 0001 0010 0011 0100
  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 be rearranged
Operation of the RCL Instruction

- RCL instruction operation
  - Typical instruction—count in CL
    RCL BX,CL
  - Before execution
    Dest = (BX) = 1234H = 0001 0010 0011 0100₂
    Count = 04H, CF = 0
  - Operation
    - The value in all bits of AX are shifted left four bit positions
    - Emptied MSBs are rotated through the carry bit back into the LSB
    - Last value rotated out of MSB retained in carry flag
    - First rotate loads prior value of CF at the LSB
  - After execution
    Dest = (BX) = 2340H = 0010 0011 0100 0000₂
    CF = 1
RCR Example

- RCR instruction debug execution example
  - Instruction—count in CL
    RCR BX, CL
  - Before execution
    Dest = (BX) = 1234H = 0001 0010 0011 01002
    Count = 04H
    CF = 0
  - After execution
    Dest = (BX) = 8123H = 100000010010001101002
    CF = 0

C:\DOS> DEBUG
-A
1342:0100 RCR BX, CL
1342:0102 -R BX
  EX 0000
  :1234
  -R CX
  CX 0000
  :4
  -R F
  NV UP EI PL NZ NA PO NC -
  -T
AX=0000 BX=8123 CX=0004 DX=0000 SP=FFE6 BP=0000 SI=0000 DI=0000
DS=1342 ES=1342 SS=1342 CS=1342 IP=0102 OV UP EI PL NZ NA PO NC
1342:0102 B98AFF  MOV CX,FF8A
-Q
C:\DOS>
Rotate Instructions- Application

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

- **BT** → Bit test
- **BTR** → Bit test and reset
- **BTS** → Bit test and set
- **RTC** → Bit test and complement

**Format of bit test instruction:** \(BT(x) \ D, S\)
- \(S\) → index that selects the position of the bit tested
- \(D\) → Holds value tested

**Operation:**
- Enables the programmer to test the state of a bit in a value held in a register or memory location
- **All** → Save the value of the selected bit in the CF
- **BT** → Leaves selected bit unchanged
- **BTR** → Clears the bit
- **BTS** → Sets the bit
- **BTC** → Complements the bit

---

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>Bit test</td>
<td>BT D, S</td>
<td>Saves the value of the bit in D specified by the value in S in CF.</td>
<td>CF, OF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>BTR</td>
<td>Bit test and reset</td>
<td>BTR D, S</td>
<td>Saves the value of the bit in D specified by the value in S in CF and then resets the bit in D.</td>
<td>CF, OF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>BTS</td>
<td>Bit test and set</td>
<td>BTS D, S</td>
<td>Saves the value of the bit in D specified by the value in S in CF and then sets the bit in D.</td>
<td>CF, OF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>BTC</td>
<td>Bit test and complement</td>
<td>BTC D, S</td>
<td>Saves the value of the bit in D specified by the value in S in CF and then complements the bit in D.</td>
<td>CF, OF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>BSF</td>
<td>Bit scan forward</td>
<td>BSF D, S</td>
<td>Scan the source operand starting from bit 0, ZF = 0 if all bits are 0, else ZF = 1 and the destination operand is loaded with the bit index of the first set bit.</td>
<td>ZF, OF, SF, AF, PF, CF undefined</td>
</tr>
<tr>
<td>BSR</td>
<td>Bit scan reverse</td>
<td>BSR D, S</td>
<td>Scan the source operand starting from the MSB, ZF = 0 if all bits are 0, else ZF = 1 and the destination operand is loaded with the bit index of the first set bit.</td>
<td>ZF, OF, SF, AF, PF, CF undefined</td>
</tr>
</tbody>
</table>

**Table:**

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

**Table:**

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

---

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

- Example:
  \[\text{BTC BX,7}\]

- Before execution
  \[(BX) = 03F0H = 0000 0011 1111 0000_2\]
  \[\text{IMM8} = 7\]

- After Execution
  \[(CF) = 1\]
  \[(BX) = 0370H = 0000001110110000_2\]
Bit Scan Instructions

- **BSF** → Bit scan forward
- **BSR** → Bit scan reverse

- Format of bit scan instructions: **BS(x) D,S**
  - (S) → Holds value for which bits are tested to be 0
    - (S) = Reg16, or Reg32
  - (D) → Index of first bit that tests as non-zero
    - (D) = Reg16, Reg32

- **Operation:**
  - Enable the programmer to test a value in a register or memory location to determine if all of its bits are 0
  - **BSF** → Scans bits starting from bit 0
    - Set ZF = 0 if all bits are found to be zero
    - Sets ZF = 1 when first 1 bit detected and places index of that bit into destination
  - **BSR** → Scans bits starting from MSB
    - Set ZF = 0 if all bits are found to be zero
    - Sets ZF = 1 when first 1 bit detected and places index of that bit into destination

- **Example:**
  - BSF ESI,EDX → 32-bits of EDX scanned starting from B₀