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

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

---

5.1 Data-Transfer Instructions

- The MOVE Instruction
  The move (MOV) instruction is used to transfer a byte or a word of data from a source operand to a destination operand.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>Move</td>
<td>MOV D, S</td>
<td>(S) → (D)</td>
<td>None</td>
</tr>
</tbody>
</table>

e.g.      MOV DX, CS
MOV [SUM], AX
5.1 Data-Transfer Instructions

The MOVE Instruction
Note that the MOV instruction cannot transfer data directly between external memory.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Accumulator</td>
</tr>
<tr>
<td>Accumulator</td>
<td>Memory</td>
</tr>
<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>Register</td>
<td>Immediate</td>
</tr>
<tr>
<td>Memory</td>
<td>Immediate</td>
</tr>
<tr>
<td>Seg-reg</td>
<td>Reg16</td>
</tr>
<tr>
<td>Seg-reg</td>
<td>Mem16</td>
</tr>
<tr>
<td>Reg16</td>
<td>Seg-reg</td>
</tr>
<tr>
<td>Memory</td>
<td>Seg-reg</td>
</tr>
</tbody>
</table>

Allowed operands for MOV instruction

5.1 Data-Transfer Instructions

The MOVE Instruction

MOV DX, CS

Before execution

After execution
5.1 Data-Transfer Instructions

The MOVE Instruction

MOV DX, CS

Address | Memory Content | Instruction
---|---|---
01100 | 8C | MOV DX, CS
01101 | CA | Next Instruction
01102 | XX | 
02000 | XX | 
02001 | XX | 

EXAMPLE

What is the effect of executing the instruction

MOV CX, [SOURCE_MEM]

Where SOURCE_MEM equal to 2016 is a memory location offset relative to the current data segment starting at 1A00016.

Solution:

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

Therefore CL is loaded with the contents held at memory address

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

and CH is loaded with the contents of memory address

$$1A000_{16} + 20_{16} + 1_{16} = 1A021_{16}$$
### 5.1 Data-Transfer Instructions

#### EXAMPLE

Use the DEBUG to verify the previous example.

Solution:

![DEBUG output]

---

#### 5.1 Data-Transfer Instructions

- **The XCHG Instruction**

The exchange (XCHG) instruction can be used to swap data between two general-purpose registers or between a general-purpose register and a storage location in memory.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCHG</td>
<td>Exchange</td>
<td>XCHG, D, S</td>
<td>(D) ↔ (S)</td>
<td>None</td>
</tr>
</tbody>
</table>

**e.g.**

XCHG AX, DX

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Reg16</td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Memory</td>
</tr>
</tbody>
</table>

Allowed operands for XCHG instruction
5.1 Data-Transfer Instructions

EXAMPLE

What is the result of executing the following instruction?

XCHG [SUM], BX

Where SUM = 1234_{16}, (DS)=1200_{16}

Solution:

\[(\text{DS})0+\text{SUM} \leftrightarrow \text{BX}\]

\[\text{PA} = 12000_{16} + 1234_{16} = 13234_{16}\]

Execution of the instruction performs the following 16-bit swap:

\[(13234_{16}) \leftrightarrow (\text{BL})\]
\[(13235_{16}) \leftrightarrow (\text{BH})\]

So we get \((\text{BX}) = 00FF_{16}\)
\((\text{SUM}) = 11AA_{16}\)

5.1 Data-Transfer Instructions

- The XCHG Instruction

<table>
<thead>
<tr>
<th>XCHG [SUM], BX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
</tr>
<tr>
<td>11101</td>
</tr>
<tr>
<td>11102</td>
</tr>
<tr>
<td>11103</td>
</tr>
<tr>
<td>11104</td>
</tr>
<tr>
<td>11105</td>
</tr>
<tr>
<td>12000</td>
</tr>
<tr>
<td>12001</td>
</tr>
<tr>
<td>13234</td>
</tr>
<tr>
<td>13235</td>
</tr>
</tbody>
</table>

Before execution

8086/8088 MPU
5.1 Data-Transfer Instructions

The XCHG Instruction

XCHG [SUM], BX

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

Variable “SUM”

EXAMPLE

Use the DEBUG program to verify the previous example.

Solution:

After execution
5.1 Data-Transfer Instructions

EXAMPLE

Use the DEBUG program to verify the previous example.

Solution:

The XLAT Instruction

The translate (XLAT) instruction is used to simplify implementation of the lookup-table operation. Execution of the XLAT replaces the contents of AL by the contents of the accessed lookup-table location.

\[
\text{PA} = (DS)_0 + (BX) + (AL)
\]

\[
= 03000_{16} + 0100_{16} + 0D_{16} = 0310D_{16}
\]

\[
(0310D_{16}) \rightarrow (AL)
\]
5.1 Data-Transfer Instructions

The LEA, LDS, and LES Instructions

The LEA, LDS, LES instructions provide the ability to manipulate memory addresses by loading either a 16-bit offset address into a general-purpose register or a register together with a segment address into either DS or ES.

<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 →(Reg16)</td>
<td>None</td>
</tr>
<tr>
<td>LDS</td>
<td>Load register and DS</td>
<td>LDS Reg16, Mem32</td>
<td>(Mem32) →(Reg16)</td>
<td>None</td>
</tr>
<tr>
<td>LES</td>
<td>Load register and ES</td>
<td>LES Reg16, Mem32</td>
<td>(Mem32) →(Reg16)</td>
<td>None</td>
</tr>
</tbody>
</table>

e.g. LEA SI, [DI+BX+5H]

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Content</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>C5</td>
<td>LDS SI, [200H]</td>
</tr>
<tr>
<td>1111</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>1112</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>1113</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>1114</td>
<td>XX</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>XX</td>
<td></td>
</tr>
<tr>
<td>12001</td>
<td>XX</td>
<td></td>
</tr>
<tr>
<td>12002</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>12003</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>12200</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12201</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>12202</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>12203</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Before execution

611 37100 微處理機原理與應用  Lecture 05-18
5.1 Data-Transfer Instructions

The LEA, LDS, and LES Instructions

LDS SI, [200H]

Address | Memory Content | Instruction
---------|----------------|------------------------
11100 | C5 | LDS SI, [200H] |
11101 | 36 |
11102 | 00 |
11103 | 02 |
11104 | XX |
12000 | XX |
12001 | XX |
12020 | 00 |
12202 | 00 |
12203 | 13 |
13000 | XX |
13001 | XX |

New data segment

EXAMPLE

Verify the following instruction using DEBUG program.

LDS SI, [200H]
5.1 Data-Transfer Instructions

EXAMPLE

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

Solution:

```
MOV AX, [INIT_TABLE]
MOV SS, AX
LDS SI, [INIT_TABLE+02H]
LES DI, [INIT_TABLE+06H]
MOV AX, [INIT_TABLE+0AH]
MOV BX, [INIT_TABLE+0CH]
MOV CX, [INIT_TABLE+0EH]
MOV DX, [INIT_TABLE+10H]
```

5.2 Arithmetic Instructions

- The arithmetic instructions include
  - Addition
  - Subtraction
  - Multiplication
  - Division
- Data formats
  - Unsigned binary bytes
  - Signed binary bytes
  - Unsigned binary words
  - Signed binary words
  - Unpacked decimal bytes
  - Packed decimal bytes
  - ASCII numbers
5.2 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><strong>ADD</strong></td>
<td>Addition</td>
<td>ADD D, S</td>
<td>(S) + (D) → (D) Carry → (CF)</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td><strong>ADC</strong></td>
<td>Add with carry</td>
<td>ADC D, S</td>
<td>(S) + (D) + (CF) → (D) Carry → (CF)</td>
<td>OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>Increment by 1</td>
<td>INC D</td>
<td>(D) +1 → (D)</td>
<td>OF, SF, ZF, AF, PF</td>
</tr>
<tr>
<td><strong>AAA</strong></td>
<td>ASCII adjust for addition</td>
<td>AAA</td>
<td></td>
<td>AF, CF OF, SF, ZF, PF undefined</td>
</tr>
<tr>
<td><strong>DAA</strong></td>
<td>Decimal adjust for addition</td>
<td>DAA</td>
<td></td>
<td>SF, ZF, AF, PF, CF, OF undefined</td>
</tr>
</tbody>
</table>
5.2 Arithmetic Instructions

Addition Instructions: ADD, ADC, INC, AAA, DAA

<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>Register</td>
<td>Immediate</td>
</tr>
<tr>
<td>Memory</td>
<td>Immediate</td>
</tr>
<tr>
<td>Accumulator</td>
<td>Immediate</td>
</tr>
</tbody>
</table>

Allowed operands for ADD and ADC instructions

<table>
<thead>
<tr>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg16</td>
</tr>
<tr>
<td>Reg8</td>
</tr>
<tr>
<td>Memory</td>
</tr>
</tbody>
</table>

5.2 Arithmetic Instructions

EXAMPLE

Assume that the AX and BX registers contain 1100\textsubscript{16} and 0ABC\textsubscript{16}, respectively. What is the result of executing the instruction ADD AX, BX?

Solution:

\[(BX)+(AX) = 0ABC\textsubscript{16} + 1100\textsubscript{16} = 1BBC\textsubscript{16}\]

The sum ends up in destination register AX. That is

\[(AX) = 1BBC\textsubscript{16}\]
5.2 Arithmetic Instructions

- Addition Instructions: ADD, ADC, INC, AAA, DAA

Addition Instructions:
- ADD AX, BX

Before execution:
- Address: 11100
- Memory Content: 03 C3 XX
- Instruction: ADD AX, BX
- Next instruction:
- Address: 12000
- Memory Content: XX
- Instruction: ADD AX, BX

After execution:
- Address: 11100
- Memory Content: 03 C3 XX
- Instruction: ADD AX, BX
- Next instruction:
- Address: 12000
- Memory Content: XX
5.2 Arithmetic Instructions

EXAMPLE

Verify the previous example using DEBUG program.

Solution:

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

```
ADD AX, [SUM]
ADC BL, 05H
INC WORD PTR [SUM]
```

Solution:

\[
\begin{align*}
(AX) & \leftarrow (AX) + \text{(SUM)} = 1234_{16} + 00CD_{16} = 1301_{16} \\
(BL) & \leftarrow (BL) + \text{imm8} + \text{(CF)} = AB_{16} + 5_{16} + 0_{16} = B0_{16} \\
\text{(SUM)} & \leftarrow \text{(SUM)} + 1_{16} = 00CD_{16} + 1_{16} = 00CE_{16}
\end{align*}
\]
5.2 Arithmetic Instructions

EXAMPLE

What is the result of executing the following instruction sequence?

ADD AL, BL
AAA

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

Solution:

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

The result after the AAA instruction is

$(AL) = 06_{16}$
$(AH) = 00_{16}$

with both AF and CF remain cleared

EXAMPLE

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

Solution:

$(DX,CX) \leftarrow (DX,CX)+(BX,AX)$
$(DX,CX) = FEDCBA98_{16}$
$(BX,AX) = 01234567_{16}$

MOV DX, 0FEDCH
MOV CX, 0BA98H
MOV BX, 01234H
MOV AX, 04567H
ADD CX, AX
ADC DX, BX ; Add with carry
5.2 Arithmetic Instructions

- Subtraction Instructions:
  SUB, SBB, DEC, AAS, DAS, and NEG

<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, CF</td>
</tr>
<tr>
<td>DAS</td>
<td>Decimal adjust for subtraction</td>
<td>DAS</td>
<td>SF, ZF, AF, PF, CF</td>
<td>OF undefined</td>
</tr>
<tr>
<td>AAS</td>
<td>ASCII adjust for subtraction</td>
<td>AAS</td>
<td>AF, CF</td>
<td>OF, SF, ZF, PF undefined</td>
</tr>
</tbody>
</table>

- Allowed operands for SUB and SBB instructions:
  Register, Memory

- Allowed operands for DEC instruction:
  Register, Immediate

- Allowed operands for NEG instruction:
  Register, Memory
5.2 Arithmetic Instructions

EXAMPLE

Assuming that the contents of register BX and CX are $1234_{16}$ and $0123_{16}$, respectively, and the carry flag is 0, what is the result of executing the instruction SBB BX, CX?

Solution:

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

We get

$$(BX) = 1234_{16} - 0123_{16} - 0_{16} = 1111_{16}$$

the carry flag remains cleared.

EXAMPLE

Verify the previous example using DEBUG program.

Solution:
5.2 Arithmetic Instructions

EXAMPLE

Assuming that the register BX contains 003A16, what is the result of executing the following instruction?

NEG BX

Solution:

\[(BX) = 0000_{16} - (BX) = 0000_{16} + 2 \text{ complement of } 003A_{16}
\]

\[= 0000_{16} + FFC6_{16}
\]

\[= FFC6_{16}
\]

Since no carry is generated in this add operation, the carry flag is complemented to give

\[(CF) = 1
\]

EXAMPLE

Verify the previous example using DEBUG program.

Solution:

```
CONSOLE MODE

<table>
<thead>
<tr>
<th>AX</th>
<th>BX</th>
<th>CX</th>
<th>DX</th>
<th>SP</th>
<th>BP</th>
<th>SI</th>
<th>DI</th>
<th>CF</th>
<th>AF</th>
<th>PF</th>
<th>ZF</th>
<th>SF</th>
<th>OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>003A</td>
<td>FFC6</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
```
5.2 Arithmetic Instructions

EXAMPLE

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

Solution:

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

5.2 Arithmetic Instructions

Multiplication Instructions:

MUL, DIV, IMUL, IDIV, AAM, AAD, CBW, and CWD

<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</td>
<td>MUL S</td>
<td>(AL)-(S8)→(AX) (AX)-(S16)→(DX)(AX)</td>
<td>OF, CF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>DIV</td>
<td>Division</td>
<td>DIV S</td>
<td>(AL)-(S8)→(AH) (AX)-(S16)→(DX)(AX)</td>
<td>OF, SF, ZF, AF, PF, CF undefined</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiply</td>
<td>IMUL S</td>
<td>(AL)-(S8)→(AX) (AX)-(S16)→(DX)(AX)</td>
<td>OF, CF, SF, ZF, AF, PF undefined</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer divide</td>
<td>IDIV S</td>
<td>(AL)-(S8)→(AX) (AX)(S16)→(DX)(AX)</td>
<td>OF, SF, ZF, AF, PF, CF undefined</td>
</tr>
</tbody>
</table>

Flags affected:

- OF: Overflow flag
- CF: Carry flag
- SF: Sign flag
- ZF: Zero flag
- AF: Auxiliary carry flag
- PF: Parity flag
5.2 Arithmetic Instructions

Multiplication Instructions:
MUL, DIV, IMUL, IDIV, AAM, AAD, CBW, and CWD

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAM</td>
<td>Adjust AL for multiplication</td>
<td>AAM</td>
<td>Q((AL)/10)→(AH)</td>
<td>SF,ZF,PF OF,AF,CF undefined</td>
</tr>
<tr>
<td>AAD</td>
<td>Adjust AX for division</td>
<td>AAD</td>
<td>(AH)·10+(AL)→(AH)</td>
<td>SF,ZF,PF OF,AF,CF undefined</td>
</tr>
<tr>
<td>CBW</td>
<td>Convert byte to word</td>
<td>CBW</td>
<td>(MSB of AL)→(All bits of AH)</td>
<td>None</td>
</tr>
<tr>
<td>CWD</td>
<td>Convert word to double word</td>
<td>CWD</td>
<td>(MSB of AX)→(All bits of DX)</td>
<td>None</td>
</tr>
</tbody>
</table>

5.2 Arithmetic Instructions

EXAMPLE

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

MUL CL and IMUL CL

Solution:

(AL) = -1 (as 2’s complement) = 11111111₂ = FF₁₆
(CL) = -2 (as 2’s complement) = 11111110₂ = FE₁₆

Executing the MUL instruction gives

(AX) = 11111111₁₂×11111110₁₂=111110100000010₂=FD₀₂₁₆

Executing the IMUL instruction gives

(AX) = -1₁₀ × -2₁₀ = 2₁₀ = 0002₁₆
5.2 Arithmetic Instructions

EXAMPLE

Verify the previous example using DEBUG program.

Solution:

What is the result of executing the following instructions?

MOV AL, 0A1H
CBW
CWD

Solution:

(\text{AL}) = A_{16} = 10100001_2

Executing the CBW instruction extends the MSB of AL

(\text{AH}) = 11111111_2 = \text{FF}_{16} \quad \text{or} \quad (\text{AX}) = 111111110100001_2

Executing the CWD instruction, we get

(\text{DX}) = 1111111111111111_2 = \text{FFFF}_{16}

That is, \quad (\text{AX}) = \text{FFA1}_{16} \quad (\text{DX}) = \text{FFFF}_{16}
5.3 Logic Instructions

The logic instructions include:
- AND
- OR
- XOR (Exclusive-OR)
- NOT

### Logic Instructions 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>AND</td>
<td>Logical AND</td>
<td>AND D, S</td>
<td>(S) → (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)</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) ⊕ (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>(NOT D) → (D)</td>
<td>None</td>
</tr>
</tbody>
</table>

### Allowed Operands

- **Source**
  - Register
  - Memory
  - Immediate
  - Accumulator

- **Destination**
  - Register
  - Memory

Allowed operands for AND, OR, and XOR instructions
Allowed operands for NOT instruction
5.3 Logic Instructions

EXAMPLE
Describe the results of executing the following instructions?

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

Solution:

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

Executing the OR instruction, we get

\[ (AL) = 00010101_2 + 11000000_2 = 11010101_2 = D5_{16} \]

Executing the XOR instruction, we get

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

Executing the NOT instruction, we get

\[ (AL) = (NOT)11011010_2 = 00100101_2 = 25_{16} \]

---

5.3 Logic Instructions

EXAMPLE

Masking and setting bits in a register.

Solution:

Mask off the upper 12 bits of the word of data in AX

```
AND AX, 000F_16
```

Setting B4 of the byte at the offset address CONTROL_FLAGS

```
MOV AL, [CONTROL_FLAGS]
OR AL, 10H
MOV [CONTROL_FLAGS], AL
```

Executing the above instructions, we get

\[ (AL) = \text{XXXXXXXX}_2 + 00010000_2 = \text{XXX1XXXX}_2 \]
### 5.4 Shift Instructions

- **Shift instructions:** SHL, SHR, SAL, SAR

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAL/SHL</td>
<td>Shift arithmetic left / Shift logical left</td>
<td>SAL, D, Count SHL, D, Count</td>
<td>Shift the (D) left by the number of bit positions equal to Count and fill the vacated bits positions on the right with zeros</td>
<td>CF, PF, SF, Z, AF undefined if count ≠ 1</td>
</tr>
<tr>
<td>SHR</td>
<td>Shift logical right</td>
<td>SHR, D, Count</td>
<td>Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with zeros</td>
<td>CF, PF, SF, Z, AF undefined if count ≠ 1</td>
</tr>
<tr>
<td>SAR</td>
<td>Shift arithmetic right</td>
<td>SAR, D, Count</td>
<td>Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with the original most significant bits</td>
<td>CF, PF, SF, Z, AF undefined if count ≠ 1</td>
</tr>
</tbody>
</table>

**Allowed operands for shift instructions**

- **Destination**
  - Register
  - Memory
  - CL

<table>
<thead>
<tr>
<th>Destination</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>CL</td>
</tr>
<tr>
<td>Memory</td>
<td>CL</td>
</tr>
<tr>
<td>CL</td>
<td></td>
</tr>
</tbody>
</table>
5.4 Shift Instructions

- Shift instructions: SHL, SHR, SAL, SAR

**EXAMPLE**

Assume that CL contains $02_{16}$ and AX contains $091A_{16}$. Determine the new contents of AX and the carry flag after the instruction SAR AX, CL is executed.

Solution:

$$(AX) = 0000001001000110_2 = 0246_{16}$$

and the carry flag is (CF) = 1
5.4 Shift Instructions

EXAMPLE

Verify the previous example using DEBUG program.

Solution:

![Debug program screenshot]

5.4 Shift Instructions

EXAMPLE

Isolate the bit B₃ of the byte at the offset address CONTROL_FLAGS.

Solution:

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

Executing the instructions, we get

(AL)=0000B₇B₆B₅B₄

and

(CF)=B₃
### 5.5 Rotate Instructions

#### Rotate instructions: ROL, ROR, RCL, RCR

<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, CL</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 undefined if count ≠ 1</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate right</td>
<td>ROR, CL</td>
<td>Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position.</td>
<td>CF undefined if count ≠ 1</td>
</tr>
<tr>
<td>RCL</td>
<td>Rotate left through carry</td>
<td>RCL, CL</td>
<td>Same as ROL except carry is attached to (D) for rotation.</td>
<td>CF undefined if count ≠ 1</td>
</tr>
<tr>
<td>RCR</td>
<td>Rotate right through carry</td>
<td>RCR, CL</td>
<td>Same as ROL except carry is attached to (D) for rotation.</td>
<td>CF undefined if count ≠ 1</td>
</tr>
</tbody>
</table>

Allowed operands for rotate instructions:

<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>

Flags affected:
- CF: undefined if count ≠ 1
5.5 Rotate Instructions
- Rotate instructions: ROL, ROR, RCL, RCR

For RCL, RCR, the bits are rotate through the carry flag

Figure 5-30  Results of executing ROL AX, 1. (b) Results of executing ROR AX, CL with (CL)=4.

Figure 5-31  Rotation caused by execution of the RCL instruction.
5.5 Rotate Instructions

EXAMPLE

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

RCR BX, CL

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

Solution:

The original contents of BX are

\[(BX) = 0001001000110100_{2} = 1234_{16}\]

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

\[(BX) = 1000000100100011_{2} = 8123_{16}\]

\[(CF) = 0_{2}\]

EXAMPLE

Verify the previous example using DEBUG program.

Solution:
5.5 Rotate Instructions

EXAMPLE

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

Solution:

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