4.1 Converting Assembly Language Instructions to Machine Code
4.2 Encoding a Complete Program in Machine Code
4.3 The PC and Its DEBUG Program
4.4 Examining and Modifying the Contents of Memory
4.5 Input and Output of Data
THE DEBUG, A SOFTWARE DEVELOPMENT PROGRAM FOR THE PC

4.6 Hexadecimal Addition and Subtraction
4.7 Loading, Verifying and Saving Machine Language Program
4.8 Assembling Instructions with the Assemble Command
4.9 Executing Instructions and Programs with the TRACE and GO command
4.10 Debugging a Program

4.1 Converting Assembly Language Instructions to Machine Code

- General instruction format for machine code
4.1 Converting Assembly Language Instructions to Machine Code

- Byte 1 specification
  - Opcode field (6-bits)
    Specifies the operation to be performed
  - Register direction bit (D-bit)
    - 1 – the register operand is a destination operand
    - 0 – the register operand is a source operand
  - Data size bit (W-bit)
    - 1 – 16-bit data size
    - 0 – 8-bit data size

- Byte 2 specification
  - Mode (MOD) field (2-bits)
    Together with R/M field to specify the second operand
  - Register (REG) field (3-bit)
    Identifies the register for the first operand
  - Register/Memory (R/M) field (3-bit)

<table>
<thead>
<tr>
<th>REG</th>
<th>W=0</th>
<th>W=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>AL</td>
<td>AX</td>
</tr>
<tr>
<td>001</td>
<td>CL</td>
<td>CX</td>
</tr>
<tr>
<td>010</td>
<td>DL</td>
<td>DX</td>
</tr>
<tr>
<td>011</td>
<td>BL</td>
<td>BX</td>
</tr>
<tr>
<td>100</td>
<td>AH</td>
<td>SP</td>
</tr>
<tr>
<td>101</td>
<td>CH</td>
<td>BP</td>
</tr>
<tr>
<td>110</td>
<td>DH</td>
<td>SI</td>
</tr>
<tr>
<td>111</td>
<td>BH</td>
<td>DI</td>
</tr>
</tbody>
</table>
### 4.1 Converting Assembly Language Instructions to Machine Code

#### Byte 2 specification

<table>
<thead>
<tr>
<th>CODE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Memory Mode, no displacement follows*</td>
</tr>
<tr>
<td>01</td>
<td>Memory Mode, 8-bit displacement follows</td>
</tr>
<tr>
<td>10</td>
<td>Memory Mode, 16-bit displacement follows</td>
</tr>
<tr>
<td>11</td>
<td>Register Mode (no displacement)</td>
</tr>
</tbody>
</table>

* Except when R/M = 11, then 16-bit displacement follows

<table>
<thead>
<tr>
<th>MOD</th>
<th>R/M</th>
<th>W = 0</th>
<th>W = 1</th>
<th>EFFECTIVE ADDRESS CALCULATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>000</td>
<td>AL</td>
<td>AX</td>
<td>(BX) + (SI) + D8</td>
</tr>
<tr>
<td>06</td>
<td>010</td>
<td>DL</td>
<td>DX</td>
<td>(BP) + (BX) + D8</td>
</tr>
<tr>
<td>07</td>
<td>011</td>
<td>BL</td>
<td>BX</td>
<td>(BP) + (BP) + D8</td>
</tr>
<tr>
<td>09</td>
<td>100</td>
<td>AH</td>
<td>SP</td>
<td>(SI) + D8</td>
</tr>
<tr>
<td>0A</td>
<td>110</td>
<td>BH</td>
<td>SI</td>
<td>DIRECT ADDRESS</td>
</tr>
<tr>
<td>0C</td>
<td>111</td>
<td>DI</td>
<td>DI</td>
<td>(BX) + D8</td>
</tr>
</tbody>
</table>

**EXAMPLE**

MOV  BL, AL  

Encode the above instruction in machine code

Solution:

OPCODE = 100010 (for MOV), D = 0 (source), W = 0 (8-bit)

This leads to BYTE 1 = 10001000 = 88₁₆

In byte 2 the source operand, specified by REG, is AL

REG = 000, MOD = 11, R/M = 011

Therefore, BYTE 2 = 11000011 = C3₁₆

**MOV BL, AL = 88C3₁₆**
4.1 Converting Assembly Language Instructions to Machine Code

EXAMPLE

ADD AX, [SI]
Encode the above instruction in machine code

Solution:

OPCODE = 000000 (for ADD),    D = 1 (dest.),   W = 1 (16-bit)
This leads to    BYTE 1 = 00000011₂ = 03₁₆
In byte 2 the destination operand, specified by REG, is AX
REG = 000,     MOD = 00,       R/M = 100
Therefore,        BYTE 2 = 00000100₂ = 04₁₆

ADD AX, [SI] = 0304₁₆

EXAMPLE

XOR CL, [1234H]
Encode the above instruction in machine code

Solution:

OPCODE = 001100 (for XOR),    D = 1 (dest.),   W = 0 (8-bit)
This leads to    BYTE 1 = 00110010₂ = 32₁₆
In byte 2 the destination operand, specified by REG, is CL
REG = 001,     MOD = 00,       R/M = 110
Therefore,        BYTE 2 = 00001110₂ = 0E₁₆
BYTE 3 = 34₁₆    BYTE 4 = 12₁₆

XOR CL, [1234H] = 320E3412₁₆
4.1 Converting Assembly Language Instructions to Machine Code

**EXAMPLE**

ADD [BX][DI]+1234H, AX

Encode the above instruction in machine code

**Solution:**

OPCODE = 000000 (for ADD), \( D = 0 \) (source), \( W = 1 \) (16-bit)

This leads to \( \text{BYTE 1} = 00000001_{16} = 01_{16} \)

In byte 2 the destination operand, specified by REG, is AX

\( \text{REG} = 000 \), \( \text{MOD} = 10 \), \( \text{R/M} = 001 \)

Therefore, \( \text{BYTE 2} = 10000001_{16} = 81_{16} \)

\( \text{BYTE 3} = 34_{16} \) \( \text{BYTE 4} = 12_{16} \)

**ADD [BX][DI]+1234H, AX = 01813412_{16}**

---

4.1 Converting Assembly Language Instructions to Machine Code

- **Additional one-bit field and their functions**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>No sign extension</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Sign extend 8-bit immediate data to 16 bits if ( W=1 )</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>Shift/rotate count is one</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Shift/rotate count is specified in CL register</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>Repeat/loop while zero flag is clear</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Repeat/loop while zero flag is set</td>
</tr>
</tbody>
</table>

- **Instructions that involve a segment register (SR-field)**

<table>
<thead>
<tr>
<th>Register</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>00</td>
</tr>
<tr>
<td>CS</td>
<td>01</td>
</tr>
<tr>
<td>SS</td>
<td>10</td>
</tr>
<tr>
<td>DS</td>
<td>11</td>
</tr>
</tbody>
</table>
4.1 Converting Assembly Language Instructions to Machine Code

EXAMPLE

MOV  WORD PTR [BP][DI]+1234H, 0ABCDH

 Encode the above instruction in machine code

Solution:

This example does not follow the general format
From Fig. 3-1  MOV -> 1100011W, and W = 1 for word-size data

BYTE 1 = 11000111₂ = C7₁₆
BYTE 2 = (MOD)000(R/M) = 10000011₂ = 83₁₆
BYTE 3 = 34₁₆  BYTE 4 = 12₁₆
BYTE 5 = CD₁₆  BYTE 6 = AB₁₆

MOV  WORD PTR [BP][DI]+1234H, 0ABCDH = C7833412CDAB₁₆

EXAMPLE

MOV [BP][DI]+1234H, DS

 Encode the above instruction in machine code

Solution:

This example does not follow the general format
From Fig. 3-6  MOV -> 10001100, and the instruction is
10001100(MOD)0(SR)(R/M)(DISP)
From Fig. 4-5 we find that for DS, the SR = 11
Therefore, the instruction is coded as

MOV [BP][DI]+1234H, DS

=10001100100110110011010000010010₂
=8C9B3412₁₆
4.2 Encoding a Complete Program in Machine Code

Steps in encoding a complete assembly program:
- Identify the general machine code format (Fig. 3-6)
- Evaluate the bit fields (Fig. 4-2, 4-3, 4-4, 4-5)
- Express the binary-code instruction in hexadecimal form

To execute the program, the machine code of the program must be stored in the code segment of memory.

The first byte of the program is stored at the lowest address.

EXAMPLE
Encode the “block move” program in Fig. 4-6(a) and show how it would be stored in memory starting at address 20016.

Solution:

```
MOV AX, 2000H ; LOAD AX REGISTER
MOV DS, AX ; LOAD DATA SEGMENT ADDRESS
MOV SI, 100H ; LOAD SOURCE BLOCK POINTER
MOV DI, 120H ; LOAD DESTINATION BLOCK POINTER
MOV CX, 10H ; LOAD REPEAT COUNTER
NXTPT:   MOV AH, [SI] ; MOVE SOURCE BLOCK ELEMENT TO AH
          MOV [DI], AH ; MOVE ELEMENT FROM AH TO DEST. BLOCK
          INC SI ; INCREMENT SOURCE BLOCK POINTER
          INC DI ; INCREMENT DESTINATION BLOCK POINTER
          DEC CX ; DECREMENT REPEAT COUNTER
          JNZ NXTPT ; JUMP TO NXTPT IF CX NOT EQUAL TO ZERO
          NOP ; NO OPERATION
```
### 4.2 Encoding a Complete Program in Machine Code

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Type of instruction</th>
<th>Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV AX,2000H</td>
<td>Move immediate data to register</td>
<td>1011010000000000000010000 = B8000H</td>
</tr>
<tr>
<td>MOV DL,AX</td>
<td>Move register to segment register</td>
<td>100111010110000 = 8E96H</td>
</tr>
<tr>
<td>MOV SI,100H</td>
<td>Move immediate data to register</td>
<td>101111100000000000000000001 = BE000H</td>
</tr>
<tr>
<td>MOV DI,120H</td>
<td>Move immediate data to register</td>
<td>101111111110000000000001 = BF200H</td>
</tr>
<tr>
<td>MOV CX,10H</td>
<td>Move immediate data to register</td>
<td>101110010110000000000000 = B91000H</td>
</tr>
<tr>
<td>MOV AH,[SI]</td>
<td>Move memory data to register</td>
<td>1000101001001000 = 8A24H</td>
</tr>
<tr>
<td>MOV [DI],AH</td>
<td>Move register data to memory</td>
<td>100010000100101 = 8825H</td>
</tr>
<tr>
<td>INC SI</td>
<td>Increment register to memory</td>
<td>1100010101 = 46H</td>
</tr>
<tr>
<td>INC DI</td>
<td>Increment register to memory</td>
<td>1100010111 = 47H</td>
</tr>
<tr>
<td>DEC CX</td>
<td>Decrement register to memory</td>
<td>1100010101 = 49H</td>
</tr>
<tr>
<td>JNZ NXTPT</td>
<td>Jump if not equal to zero</td>
<td>1110101111011011 = 7F7FH</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>10010000 = 90H</td>
</tr>
</tbody>
</table>

### 4.2 Encoding a Complete Program in Machine Code

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Contents</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>200H</td>
<td>80H</td>
<td>MOV AX,2000H</td>
</tr>
<tr>
<td>201H</td>
<td>90H</td>
<td>MOV DL,AX</td>
</tr>
<tr>
<td>202H</td>
<td>20H</td>
<td>MOV SI,100H</td>
</tr>
<tr>
<td>203H</td>
<td>80H</td>
<td>MOV DI,120H</td>
</tr>
<tr>
<td>204H</td>
<td>00H</td>
<td>MOV CX,10H</td>
</tr>
<tr>
<td>205H</td>
<td>10H</td>
<td>MOV AH,[SI]</td>
</tr>
<tr>
<td>206H</td>
<td>5AH</td>
<td>MOV [DI],AH</td>
</tr>
<tr>
<td>207H</td>
<td>20H</td>
<td>INC SI</td>
</tr>
<tr>
<td>208H</td>
<td>40H</td>
<td>INC DI</td>
</tr>
<tr>
<td>209H</td>
<td>20H</td>
<td>DEC CX</td>
</tr>
<tr>
<td>210H</td>
<td>70H</td>
<td>JNZ NXTPT</td>
</tr>
<tr>
<td>211H</td>
<td>60H</td>
<td>NOP</td>
</tr>
</tbody>
</table>
4.3 The PC and Its DEBUG Program

- Using DEBUG, the programmer can issue commands to the microcomputer.
- Loading the DEBUG program
  
  C:\DEBUG

- Six kinds of information are entered as part of a command:
  - A command letter
  - An address
  - A register name
  - A file name
  - A drive name
  - Data

4.3 The PC and Its DEBUG Program

- The DEBUG program command set

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>(REGISTER NAME)</td>
<td>Execute or modify the contents of an internal register</td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td>End use of the DEBUG program</td>
</tr>
<tr>
<td>D</td>
<td>(ADDRESS)</td>
<td>Dump the contents of memory to the display</td>
</tr>
<tr>
<td>S</td>
<td>(ADDRESS, ADDR)</td>
<td>Display or modify the contents of memory</td>
</tr>
<tr>
<td>F</td>
<td>STARTING ADDRESS ENDING ADDRESS LIST</td>
<td>Fill a block in memory with the data in list</td>
</tr>
<tr>
<td>M</td>
<td>STARTING ADDRESS ENDING ADDRESS DESTINATION ADDRESS</td>
<td>Move a block of data from a source location to memory as a destination</td>
</tr>
<tr>
<td>C</td>
<td>STARTING ADDRESS ENDING ADDRESS DESTINATION ADDRESS</td>
<td>Compare two blocks of data in memory and display the differences</td>
</tr>
<tr>
<td>I</td>
<td>ADDRESS</td>
<td>Search through a block of data in memory and display all locations that match the data in list</td>
</tr>
<tr>
<td>A</td>
<td>ADDRESS, DPT, RPT</td>
<td>Print the data to the output port</td>
</tr>
<tr>
<td>N</td>
<td>(ADDRESS, NUMBER, ADDRESS, NUMBER)</td>
<td>Generate hexadecimtal sum and difference of the two numbers</td>
</tr>
<tr>
<td>V</td>
<td>(STARTING ADDRESS, NUMBER, STARTING SECTION NUMBER OF SECTIONS)</td>
<td>Perform a count of the specified number of Instructions</td>
</tr>
<tr>
<td>H</td>
<td>(STARTING ADDRESS)</td>
<td>Perform a count of the specified number of Instructions</td>
</tr>
<tr>
<td>T</td>
<td>(ADDRESS, NUMBER)</td>
<td>Clear data from memory to the disk</td>
</tr>
<tr>
<td>G</td>
<td>(STARTING ADDRESS, STARTING ADDRESS, NUMBER, ADDRESS)</td>
<td>Clear data from memory to the disk</td>
</tr>
<tr>
<td>W</td>
<td>(ADDRESS, NUMBER)</td>
<td>Clear data from memory to the disk</td>
</tr>
</tbody>
</table>
### 4.3 The PC and Its DEBUG Program

#### An initial state when with the loading of DEBUG

- **IP**: 0000<sub>16</sub>
- **CS**: 1342<sub>16</sub>
- **DS**: 1342<sub>16</sub>
- **SS**: 1342<sub>16</sub>
- **ES**: 1342<sub>16</sub>
- **AX**: 0000<sub>16</sub>
- **BX**: 0000<sub>16</sub>
- **CX**: 0000<sub>16</sub>
- **DX**: 0000<sub>16</sub>
- **SP**: FFEE<sub>16</sub>
- **BP**: 0000<sub>16</sub>
- **SI**: 0000<sub>16</sub>
- **DI**: 0000<sub>16</sub>
- **Flags**: 0040<sub>16</sub>

#### Memory
- Active code segment, data segment, stack segment, and extra segment (64 K bytes)
- Memory: 00000<sub>16</sub> to FFFFF<sub>16</sub>

---

#### Syntax for the REGISTER (R) command

**R [REGISTER NAME]**

- **e.g.**
  - `-R AX` (←)

  - `AX 0000`
  - `:00FF` (←) ; This alters the content of AX
4.3 The PC and Its DEBUG Program

EXAMPLE

Verify the initialized state of the 8088 by examining the contents of its registers with the Register command.

Solution:

-R (..)

Register mnemonics for the R command:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>Accumulator register</td>
</tr>
<tr>
<td>BX</td>
<td>Base register</td>
</tr>
<tr>
<td>CX</td>
<td>Count register</td>
</tr>
<tr>
<td>DX</td>
<td>Data register</td>
</tr>
<tr>
<td>SI</td>
<td>Source index register</td>
</tr>
<tr>
<td>DI</td>
<td>Destination index register</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer register</td>
</tr>
<tr>
<td>BP</td>
<td>Base pointer register</td>
</tr>
<tr>
<td>CS</td>
<td>Code segment register</td>
</tr>
<tr>
<td>DS</td>
<td>Data segment register</td>
</tr>
<tr>
<td>SS</td>
<td>Stack segment register</td>
</tr>
<tr>
<td>ES</td>
<td>Extra segment register</td>
</tr>
<tr>
<td>F</td>
<td>Flag register</td>
</tr>
<tr>
<td>IP</td>
<td>Instruction pointer</td>
</tr>
</tbody>
</table>
4.3 The PC and Its DEBUG Program

- Status flag notations

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
<th>Set</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF</td>
<td>Overflow</td>
<td>OV</td>
<td>NV</td>
</tr>
<tr>
<td>DF</td>
<td>Direction</td>
<td>DN</td>
<td>UP</td>
</tr>
<tr>
<td>IF</td>
<td>Interrupt</td>
<td>EI</td>
<td>DI</td>
</tr>
<tr>
<td>SF</td>
<td>Sign</td>
<td>NG</td>
<td>PL</td>
</tr>
<tr>
<td>ZF</td>
<td>Zero</td>
<td>ZR</td>
<td>NZ</td>
</tr>
<tr>
<td>AF</td>
<td>Auxiliary carry</td>
<td>AC</td>
<td>NA</td>
</tr>
<tr>
<td>PF</td>
<td>Parity</td>
<td>PE</td>
<td>PO</td>
</tr>
<tr>
<td>CF</td>
<td>Carry</td>
<td>CY</td>
<td>NC</td>
</tr>
</tbody>
</table>

EXAMPLE

Issue commands to the DEBUG program on the PC that causes the value in BX to be modified to FF00\textsubscript{16} and then verify that this new value is loaded into BX.

Solution:

```
-R BX (\_\_)
BX 0000
:FF00 (\_\_)
-R BX (\_\_)
BX FF00
:_ (\_\_)
```
4.3 The PC and Its DEBUG Program

EXAMPLE
Use the Register command to set the parity flag to even parity. Verify that the flag has been changed.

Solution:

- R F (→)
  NV UP EI PL NZ NA PO NC -PE (←)
- R F (→)
  NV UP EI PL NZ NA PE NC - (←)

4.4 Examining and Modifying the Contents of Memory

- The commands provided for use in examining and modifying the memory:
  - DUMP
  - ENTER
  - FILL
  - MOVE
  - COMPARE
  - SEARCH
4.4 Examining and Modifying the Contents of Memory

- **DUMP Command (D)**

  The DUMP command allows us to examine the contents of a memory location or a block of consecutive memory location.

  **D [ADDRESS]**

  e.g. -D \(_\downarrow\)  
  -D 1342:100 \(_\downarrow\)  
  -D DS:100 \(_\downarrow\)  
  -D 100 \(_\downarrow\)
4.4 Examining and Modifying the Contents of Memory

EXAMPLE

Issue a dump command to display the contents of the 32 bytes of memory located at offset 030016 through 031F16 in the current data segment.

Solution:

-D 300 31F (∼)

EXAMPLE

Use the Dump command to examine the 16 bytes of memory just below the top of the stack.

Solution:

-D SS:FFEE FFFD (∼)
4.4 Examining and Modifying the Contents of Memory

■ ENTER Command (E)

\[ E \ \text{ADDRESS [LIST]} \]

e.g. 

\[-E\ DS:100\ FF\ FF\ FF\ FF\ \ (\_\_\_\_) \]

\[-E\ DS:100\ \ (\_\_\_) \]

\[-1342:0100\ \text{FF.\ }\ (\_\_\_)\ (\text{Return to end}) \]

\[-E\ DS:100\ \ (\_\_\_) \]

\[-1342:0100\ \text{FF.\ }\ (\text{Space bar to continue}) \]

\[-1342:0100\ \text{FF.\ }\] (Return to end)

EXAMPLE

Start a data entry sequence by examining the contents of address DS:100 and then, without entering new data, depress the “-” key. What happen?

Solution:

\[-E\ DS:100\ \ (\_\_\_) \]

\[1342:0100\ \text{FF.\ }\]

Entering “-” causes the display of previous byte storage location.
4.4 Examining and Modifying the Contents of Memory

EXAMPLE
Enter ASCII data to the memory.
Solution:

-E DS:200 “ASCII” (→)
-E DS:200 ‘ASCII’ (→)

FILL Command (F)
The FILL command fills a block of consecutive memory locations all with the same data.

F STARTING_ADDRESS ENDING_ADDRESS LIST

e.g.  -F 100 11F 22 (→)
4.4 Examining and Modifying the Contents of Memory

EXAMPLE

Initialize all storage locations in the block of memory from DS:120 through DS:13F with the value 33\textsubscript{16} and the block of storage locations from DS:140 to DS:15F with the value 44\textsubscript{16}.

Solution:

- F 120 13F 33 (\_\_\_\_\_)
- F 140 15F 44 (\_\_\_\_\_)

---

4.4 Examining and Modifying the Contents of Memory

- MOVE Command (M)

The MOVE command allows us to copy a block of data from one part of memory to another part.

\[ \text{M START\_ADDRESS END\_ADDRESS DEST\_ADDRESS} \]

e.g. \(-M 100 11F 200 (\_\_\_\_\_))\)
4.4 Examining and Modifying the Contents of Memory

EXAMPLE

Fill each storage location in the block of memory from address DS:100 through DS:11F with the value 1116. Then copy this block of data to a destination block starting at DS:160.

Solution:  
-F 100 11F 11 (…)  
-M 100 11F 160 (…)  

4.4 Examining and Modifying the Contents of Memory

COMPARE Command (C)

The COMPARE command allows us to compare the contents of two blocks of data to determine if they are or are not the same.

C START_ADDRESS END_ADDRESS DEST_ADDRESS

e.g.    -C 100 10F 120 (…)  

611 37100 微處理機原理與應用 Lecture 04-39
### 4.4 Examining and Modifying the Contents of Memory

- **COMPARE Command (C)**

  ```
  C START_ADDRESS  END_ADDRESS  LIST
  ```

  - e.g. `-S 100 17F 33 (-)`

- **SEARCH Command (S)**

  The SEARCH command can be used to scan through a block of data in memory to determine whether or not it contains specific data.
4.4 Examining and Modifying the Contents of Memory

- SEARCH Command (S)

4.5 Input and Output of Data

- INPUT Command (I)

The INPUT command reads data from an input port of the 64K byte-wide ports of 8088 I/O.

\[
\text{I ADDRESS}
\]

e.g. 

- I 61 (\text{-})

4D

The contents of the port at I/O address \text{0061}_{16} are \text{4D}_{16}
### 4.5 Input and Output of Data

**OUTPUT Command (O)**

The OUTPUT command writes data to an input port of the 64K byte-wide ports of 8088 I/O.

\[ \text{O ADDRESS BYTE} \]

*Example:* 

```
-O 61 4F (\text{\textbackslash})
```

This command causes the value \(4F_{16}\) to be written into the byte-wide output port at address \(0061_{16}\).

---

### 4.6 Hexadecimal Addition and Subtraction

**HEXADECIMAL Command (H)**

The HEXADECIMAL command provides the ability to add and subtract hexadecimal numbers.

\[ \text{H NUM1 NUM2} \]

*Example:*

```
-H ABC0 0FFF (\text{\textbackslash})
BBBF 9BC1

-H BBBF A (\text{\textbackslash})
BBC9 BBB5
```

*Both number and results are limited to four hexadecimal digits.*
4.6 Hexadecimal Addition and Subtraction

EXAMPLE

Use the H command to find the negative of the number $0009_{16}$.

Solution:

\[-H \quad 0 \quad 9 \quad (\_\_)
\]
\[0009 \quad FFF7\]

$FFF7_{16}$ is the negative of $9_{16}$ expressed in 2’s complement form.

EXAMPLE

If a byte of data is located at physical address $02A34_{16}$ and the data segment register contains $0150_{16}$, what value must be loaded into the source index register such that DS:SI points to the byte storage location?

Solution:

\[-H \quad 2A34 \quad 1500 \quad (\_\_)
\]
\[3F34 \quad 1534\]

This shows that SI must be loaded with the value $1534_{16}$. 
4.7 Loading, Verifying and Saving Machine Language Program

- An example to load an instruction

```plaintext
MOV BL, AL
The machine code is 88C3₁₆
-E CS:100 88 C3 (⊥)
-D CS:100 101 (⊥)
1342:0100 88 C3
```

---

4.7 Loading, Verifying and Saving Machine Language Program

- UNASSEMBLE Command (U)

The UNASSEMBLE command converts machine code instructions to their equivalent assembly language source statement.

```
U [STARTING_ADDRESS [ENDING_ADDRESS]]
```

```plaintext
e.g.    -U CS:100 101 (⊥)
        1342:0100 88C3 MOV BL, AL
```
4.7 Loading, Verifying and Saving Machine Language Program

EXAMPLE

Use a sequence of commands to load, verify loading, and unassemble the machine code instruction 0304H. Load the instruction at address CS:200.

Solution:  
- E CS:200 03 04 (\text{\texttt{-E}})  
- D CS:200 201 (\text{\texttt{-D}})  
- U CS:200 201 (\text{\texttt{-U}})  
ADD AX, [SI]

4.7 Loading, Verifying and Saving Machine Language Program

\begin{itemize}
  \item WRITE Command (W)
\end{itemize}

The WRITE command gives the ability to save data stored in memory on a diskette.

\textbf{W [START\_ADDRESS [DRIVE START\_SECTOR NUM\_SECTOR] ]}

e.g.  
- W CS:200 1 10 1 (\text{\texttt{-W}})  
- W 200 1 10 1 (\text{\texttt{-W}})

* Be caution in saving program in a disk, especially the hard drive.

Drive B \hspace{1cm} 1 Sector = 512 Byte

\textbf{Drive B \hspace{1cm} 1 Sector = 512 Byte}
4.7 Loading, Verifying and Saving Machine Language Program

**LOAD Command (L)**

The LOAD command gives the ability to reload memory from a diskette.

```
L [START_ADDRESS [DRIVE START_SECTOR NUM_SECTOR] ]
```

*e.g.*

```
-L  CS:300  1  10  1  (..)
```

The reloading of the instruction can be verified by U command

*e.g.*

```
-U  CS:300  301 (..)
```

```
1342:300 301 ADD AX, [SI]
```

---

**EXAMPLE**

Enter the machine code of the block move program. The program is to be loaded into memory starting at address CS:100. Verify, unassemble, and save the code.

**Solution:**

```
-E CS:100 B8 00 20 8E D8 00 01 BF 20 01 B9 10 00 8A 24 88 25 46 (..)
-D CS:100 117(..)
-U CS:100 117(..)
-W CS:100 1 100 1 (..)
```
4.7 Loading, Verifying and Saving Machine Language Program

- NAME Command (N)

The NAME command, along with the WRITE command, gives the ability to save a program on the diskette under a file name.

```
N FILE NAME
```

- The BX, CX registers must be updated to identify the size of the program that is to be saved in the file.
- After BX, CX registers have been initialized, the write command is used to saved the program.
- To reload the program, the command sequence is

```
N FILE NAME
L [STARTING ADDRESS]
```
4.7 Loading, Verifying and Saving Machine Language Program

EXAMPLE

Save a machine code program into a file.

Solution:

-N A:BLK.1 (.J) ; Give a file name in disk A
-R CX (.J) ; Give a program size of 18,16 bytes
CX XXXX
:18
-R BX (.J)
BX XXXX
:0 (.J)
W CS:100 (.J) ; Save the program in disk A

EXAMPLE

Reload a program into memory.

Solution:

-N A:BLK.1 (.J) ; Give a file name in disk A
-L CS:100 (.J) ; Load the program name BLK.1 in disk A

C:\DOS>REN A:BLK.1 BLK.EXE (.J) ; Rename the file
C:\DOS>DEBUG A:BLK.EXE (.J) ; Load the program directly
C:\DOS>A:BLK.EXE (.J) ; Run the program
4.8 Assembling Instructions with the Assemble Command

ASSEMBLE Command (A)

The ASSEMBLE command lets us automatically assemble the instructions of a program.

A [STARTING_ADDRESS]

e.g.  
-A CS:100 (.)
1342:0100 _
1342:0100 ADD [BX+SI+1234], AX (.)
1342:0104 _
-D CS:100 103 (.)

EXAMPLE

Assemble a complete program with the ASSEMBLE command.

Solution:

-A CS:200 (.)
0B35:0200 MOV AX, 2000 (.)
0B35:0203 MOV DS, AX (.)
0B35:0205 MOB SI, 100 (.)
.
.
.
0B35:0217 NOP (.)
0B35:0218 (.)

4.8 Assembling Instructions with the Assemble Command

- Assemble a program with ASSEMBLE command

```
CONSOLE MODE - DEBUG

0035:0200  86 00 20  MOV AX,2000
0035:0203  86 00 08  MOV DS,AX
0035:0206  86 01 00  MOV SI,100
0035:0209  86 01 10  MOV DI,120
0035:020C  86 01 10  MOV CX,10
0035:020E  86 01 00  MOV AH,[SI]
0035:0210  86 00 00  MOV [DI],AH
0035:0213  86 01 00  INC SI
0035:0214  86 01 00  INC DI
0035:0215  86 01 00  DEC CX
0035:0217  86 01 20  JNZ 20E
0035:0217  86 00 00  NOP
```

4.8 Assembling Instructions with the Assemble Command

- Unassemble a program with UNASSEMBLE command

```
CONSOLE MODE - DEBUG

0035:0200  86 00 20  MOV AX,2000
0035:0203  86 00 08  MOV DS,AX
0035:0206  86 01 00  MOV SI,100
0035:0209  86 01 10  MOV DI,120
0035:020C  86 01 10  MOV CX,10
0035:020E  86 01 00  MOV AH,[SI]
0035:0210  86 00 00  MOV [DI],AH
0035:0213  86 01 00  INC SI
0035:0214  86 01 00  INC DI
0035:0215  86 01 00  DEC CX
0035:0217  86 01 20  JNZ 20E
0035:0217  86 00 00  NOP
```
4.9 Executing Instructions and Programs with the TRACE and GO command

TRACE Command (T)
The TRACE command provides the programmer with the ability to execute the program one instruction at a time.

T [=STARTING_ADDRESS] [NUMBER]

e.g. -T =CS:100
- T
- T =CS:100 3

EXAMPLE
Load and trace a program.
Solution:
- L CS:100 1 10 1
- U 100 101
- R AX
AX 0000 :1111
- R SI
SI 0000 :1234
- E DS:1234 22 22
- T =CS:100
4.9 Executing Instructions and Programs with the TRACE and GO command

- GO Command (G)

The GO command is typically used to run programs that are already working or to execute programs in the later stages of debugging.

```
G [=STARTING_ADDRESS [BREAKPOINT_ADDRESS_LIST]]
```

e.g.  
- \( -G \) =CS:200  217 (\( ...) \)
- \( -G \) =CS:100 (\( ...) \)
- \( -G \) (\( ...) \)
4.9 Executing Instructions and Programs with the TRACE and GO command

EXAMPLE

Use GO command to execute a program and examine the result.

Solution:

- N A:BLK.EXE (...) ; Define the program file to be loaded
- L CS:200 (...) ; Load the program at CS:200
- R DS (...) 
  DS 1342
  :2000 (...) ; Define the data segment address
- F DS:100 10F FF (...) ; Fill memory with FF
- F DS:120 12F 00 (...) ; Fill memory with 00
- R DS (...) 
  DS 2000
  :1342 ; Store data segment with 1342_{16}

EXAMPLE (continued)

- R (...) ; Show data register status
- U CS:200 217 (...) ; Unassemble the program
- G =CS:200 20E (...) ; Execute the program to CS:20E
- G =CS:20E 215 (...) ; Execute the program to CS:215
- D DS:100 10F (...) ; Display memory at DS:100
- D DS:120 12F (...) ; Display memory at DS:120
- G =CS:215 217 (...) ; Execute the program to CS:217
- D DS:100 10F (...) ; Display memory at DS:100
- D DS:120 12F (...) ; Display memory at DS:120
4.9 Executing Instructions and Programs with the TRACE and GO command

CONSOLE MODE - DEBUG

4.9 Executing Instructions and Programs with the TRACE and GO command

CONSOLE MODE - DEBUG
4.9 Executing Instructions and Programs with the TRACE and GO command

4.10 Debugging a Program

- Errors in a program are also referred to as bugs; the process of removing them is called debugging.
- Two types of errors
  - Syntax error
  - Execution error
- A syntax error is an error caused by not following the rules for coding or entering an instruction. These types of errors are typically identified by the microcomputer and signaled to user with an error message.
- In the DEBUG environment, the TRACE command is usually used to debug execution errors.
### 4.10 Debugging a Program

**Review of the DEBUG commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>R (REGISTER NAME)</td>
<td>Examine or modify the contents of an internal register</td>
</tr>
<tr>
<td>Quit</td>
<td>Q</td>
<td>End use of the DEBUG program</td>
</tr>
<tr>
<td>U牧</td>
<td>U</td>
<td>Dump the contents of memory to the display</td>
</tr>
<tr>
<td>A牧</td>
<td>A</td>
<td>Examine or modify the contents of memory</td>
</tr>
<tr>
<td>F牧</td>
<td>F</td>
<td>Fill a block in memory with the data in list</td>
</tr>
<tr>
<td>M牧</td>
<td>M</td>
<td>Move a block of data from one location in memory to another</td>
</tr>
<tr>
<td>C牧</td>
<td>C</td>
<td>Compare the contents of memory (or the numbers in an extended data)</td>
</tr>
<tr>
<td>S牧</td>
<td>S</td>
<td>Search through a block of data in memory and display all locations that match the data in list</td>
</tr>
<tr>
<td>I牧</td>
<td>I</td>
<td>Read the input port</td>
</tr>
<tr>
<td>O牧</td>
<td>O</td>
<td>Write the data to the output port</td>
</tr>
<tr>
<td>InputHexAddress</td>
<td>M</td>
<td>Compare hexadecimal sum and difference of the two numbers</td>
</tr>
<tr>
<td>OutputHexAddress</td>
<td>W</td>
<td>Execute the instruction with hex input addresses</td>
</tr>
<tr>
<td>Word</td>
<td>W</td>
<td>Save the contents of memory in a file on a disk</td>
</tr>
<tr>
<td>Load</td>
<td>L</td>
<td>Load memory with the contents of a file on a disk</td>
</tr>
<tr>
<td>Assembly</td>
<td>A</td>
<td>Assemble the instructions into machine code and store in memory</td>
</tr>
<tr>
<td>Trace</td>
<td>T</td>
<td>Trace the execution of the specified number of instructions</td>
</tr>
<tr>
<td>Go</td>
<td>G</td>
<td>Execute the instructions stored in memory from the specified address</td>
</tr>
</tbody>
</table>

611 37100 微處理機原理與應用  Lecture 04-73