MACHINE LANGUAGE CODING AND THE DEBUG SOFTWARE DEVELOPMENT PROGRAM OF THE PC

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

**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 = 10001000₂ = 88₁₆

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

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

Therefore, BYTE 2 = 11000011₂ = C3₁₆

MOV BL, AL = 88C3₁₆

---

**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 = 00000001₂ = 01₁₆

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

REG = 000, MOD = 10, R/M = 010

Therefore, BYTE 2 = 10000011₂ = 81₁₆

ADD AX, [SI] = 0304₁₆
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)(RM) = 100000112 = 8316
BYTE 3 = 3416
BYTE 4 = 1216

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

4.2 Encoding a Complete Program in Machine Code

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
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

4.1 Converting Assembly Language Instructions to Machine Code

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

EXAMPLE
Encode the assembly 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
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
4.3 The PC and Its DEBUG Program

Using DEBUG, the programmer can issue commands to the microcomputer.

Loading the DEBUG program

Using `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

### Using DEBUG Commands

- **Loading the DEBUG program**
  
  ```
  C:DEBUG
  ```

- **Six kinds of information** entered as part of a command:
  1. A command letter
  2. An address
  3. A register name
  4. A file name
  5. A drive name
  6. Data

### Syntax for the REGISTER (R) command

- **R [REGISTER NAME]**
  
  - e.g. `-R AX (-)`
    
    ```
    AX 0000
    -
    -00FF (-) : This alters the content of AX
    ```

### 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>Control 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>BP</td>
<td>Base pointer register</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer register</td>
</tr>
<tr>
<td>SS</td>
<td>Segment register</td>
</tr>
<tr>
<td>DS</td>
<td>Data segment register</td>
</tr>
<tr>
<td>ES</td>
<td>Extra segment register</td>
</tr>
<tr>
<td>CS</td>
<td>Code segment register</td>
</tr>
<tr>
<td>FS</td>
<td>File segment register</td>
</tr>
<tr>
<td>GS</td>
<td>Global segment 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>CF</td>
<td>Carry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF</td>
<td>Parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>Auxiliary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZF</td>
<td>Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>Sign</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OF</td>
<td>Overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>Interrupt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DF</td>
<td>Direction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TF</td>
<td>Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>No parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Carry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Auxiliary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
- SERACH

EXAMPLE

Issue commands to the DEBUG program on the PC that causes the value in BX to be modified to FF0016 and then verify that this new value is loaded into BX.

Solution:

```
-R BX (J)
BX 0000
-R FF00 (J)
-R BX (J)
BX FF00
```

EXAMPLE

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

Solution:

```
-R F (J)
NV UP EI PL NZ NA PO NC -PE (J)
-R F (J)
NV UP EI PL NZ NA PE NC - (J)
```

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 (J)
-D 1342:100 (J)
-D DS:100 (J)
-D 100 (J)
```

4.4 Examining and Modifying the Contents of Memory

DUMP Command (D)

```
DUMP Command (D)
```

Address of the first byte of data

ASCII version of the memory data

16 bytes of data per line,
128 bytes per dump
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 0300 through 031F 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 (\( \_ \))

ENTER Command (E)

- E ADDRESS [LIST]

 e.g.

- E DS:100 FF FF FF FF (\( \_ \))
- E DS:100 (\( \_ \)) -1342:0100 FF (\( \_ \)) (Return to end)
- E DS:100 (\( \_ \)) -1342:0100 FF (\( \_ \)) (Space bar to continue)
- 1342:0100 FF (\( \_ \))

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

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 FF (\( \_ \))

Entering "-" causes the display of previous byte storage location.

Enter ASCII data to the memory.

Solution:

- E DS:200 "ASCII" (\( \_ \))
- E DS:200 ‘ASCII’ (\( \_ \))
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 33H, and the block of storage locations from DS:140 to DS:15F with the value 44H.

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.

M START_ADDRESS END_ADDRESS DEST_ADDRESS

e.g. -M 100 11F 200 (-)

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 (-)

4.4 Examining and Modifying the Contents of Memory

- 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.

S START_ADDRESS END_ADDRESS LIST

e.g. -S 100 11F 33 (-)
4.4 Examining and Modifying the Contents of Memory

- **SEARCH Command (S)**

4.6 Hexadecimal Addition and Subtraction

- **HEXADECIMAL Command (H)**
  
  The HEXADECIMAL command provides the ability to add and subtract hexadecimal numbers.

  \[
  H \text{ NUM1 NUM2}
  \]

  e.g.
  
  - \(-H \text{ ABCD 0FFF (..)}\)
  
  \(-H \text{ BBBF A (..)}\)

  *Both number and results are limited to four hexadecimal digits.

4.5 Input and Output of Data

- **INPUT Command (I)**
  
  The INPUT command read data from an input port of the 64K byte-wide ports of 8088 I/O.

  \[
  I \text{ ADDRESS}
  \]

  e.g.
  
  - \(-I \text{ 61 (..)}\)
  
  The contents of the port ant I/O address 0061 are 4D.

- **OUTPUT Command (O)**
  
  The OUTPUT command write data to an input port of the 64K byte-wide ports of 8088 I/O.

  \[
  O \text{ ADDRESS BYTE}
  \]

  e.g.
  
  - \(-O \text{ 61 4F (..)}\)

  This command causes the value 4F to be written into the byte-wide output port at address 0061.

4.6 Hexadecimal Addition and Subtraction

**EXAMPLE**

Use the H command to find the negative of the number 0009H.

Solution:

\(-H \text{ 0 9 (..)}\)

\(0009 \text{ FFF7H}\)

\(\text{FFF7H is the negative of 9H expressed in 2's complement form.}\)
4.7 Loading, Verifying and Saving Machine Language Program

An example to load an instruction

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

4.7 Loading, Verifying and Saving Machine Language Program

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

W [START_ADDRESS [DRIVE START_SECTOR NUM_SECTOR] ]
  e.g. -W CS:200 1 10 1 (\ldots)
  -W 200 1 10 1 (\ldots)

Drive B
1 Sector = 512 Byte

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

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] ]
  e.g. -U CS:100 101 (\ldots)
  \ldots1342:0100 88C3 MOV BL, AL

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 (\ldots)
  \ldotsThe reloading of the instruction can be verified by U command
  e.g. -U CS:300 301 (\ldots)
  \ldots1342:300 301 ADD AX, [SI]

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 (\ldots)
-D CS:200 201 (\ldots)
-U CS:200 201 (\ldots)
ADD AX, [SI]

4.7 Loading, Verifying and Saving Machine Language Program

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 BE 00 01 BF 20 01 B9 10 00 8A 24 88 25 46 (\ldots)
-D CS:100 117(\ldots)
-U CS:100 117(\ldots)
-W CS:100 1 100 1 (\ldots)
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.

\[ \text{N FILE NAME} \]

- 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

\[ \text{N FILE NAME L [STARTING ADDRESS]} \]

EXAMPLE

Save a machine code program into a file.
Solution:

\[ -N \text{A:BLK.1 (..)} \quad \text{; Give a file name in disk A} \]
\[ -R \text{CX (..)} \quad \text{; Give a program size of 1816 bytes} \]
\[ \text{CX XXXX :18} \]
\[ -R \text{BX (..)} \quad \text{; BX XXXX} \]
\[ :0 (..) \quad \text{W CS:100 (..)} \quad \text{; Save the program in disk A} \]

4.8 Assembling Instructions with the Assemble Command

ASSEMBLE Command (A)

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

\[ \text{A [STARTING_ADDRESS]} \]

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

EXAMPLE

Assemble a complete program with the ASSEMBLE command.
Solution:

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

EXAMPLE

Reload a program into memory.
Solution:

\[ -N \text{A:BLK.1 (..)} \quad \text{; Give a file name in disk A} \]
\[ -L \text{CS:100 (..)} \quad \text{; Load the program name BLK.1 in disk A} \]
\[ \text{C:\DOS>REN A:BLK.1 BLK.EXE (..)} \quad \text{; Rename the file} \]
\[ \text{C:\DOS>DEBUG A:BLK.EXE (..)} \quad \text{; Load the program directly} \]
\[ \text{C:\DOS>A:BLK.EXE (..)} \quad \text{; Run the program} \]
4.8 Assembling Instructions with the Assemble Command

- Assemble a program with ASSEMBLE command

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.

  \[ T \Rightarrow \text{STARTING_ADDRESS} \ [ \text{NUMBER} ] \]

  e.g.
  
  \- T \Rightarrow \text{CS:100} (~)
  
  \- T (~)
  
  \- T \Rightarrow \text{CS:100 3} (~)

- UNASSEMBLE 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 or debugging.

  \[ G \Rightarrow \text{STARTING_ADDRESS} \ [ \text{BREAKPOINT_ADDRESS_LIST} ] \]

  e.g.
  
  \- G \Rightarrow \text{CS:200 217} (~)
  
  \- G \Rightarrow \text{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 (.): Define the data segment address
- D:100 10F FF (.): Fill memory with FF
- F D:120 12F 00 (.): Fill memory with 00
- R DS (.): Store data segment with 134216

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

Solution: (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
- D DS:100 10F (.): Display memory at DS:100
- D DS:120 12F (.): Display memory at DS:120

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>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>DUMPFILE</td>
<td>Displays the contents of a file at the current address.</td>
</tr>
<tr>
<td>g</td>
<td>GOTO</td>
<td>Goto a specified address.</td>
</tr>
<tr>
<td>h</td>
<td>HELP</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>n</td>
<td>NEXT</td>
<td>Execute the next line of code.</td>
</tr>
<tr>
<td>p</td>
<td>PRINT</td>
<td>Prints the value of a specified register or memory location.</td>
</tr>
<tr>
<td>q</td>
<td>QUIET</td>
<td>Disables the display of output.</td>
</tr>
<tr>
<td>r</td>
<td>RUN</td>
<td>Continues program execution.</td>
</tr>
<tr>
<td>s</td>
<td>STEP</td>
<td>Executes a single step.</td>
</tr>
<tr>
<td>u</td>
<td>UNDUMPFILE</td>
<td>Removes the contents of a file at the current address.</td>
</tr>
<tr>
<td>x</td>
<td>EXIT</td>
<td>Exits from DEBUG.</td>
</tr>
</tbody>
</table>

Other commands may be used to control program execution and display program memory.