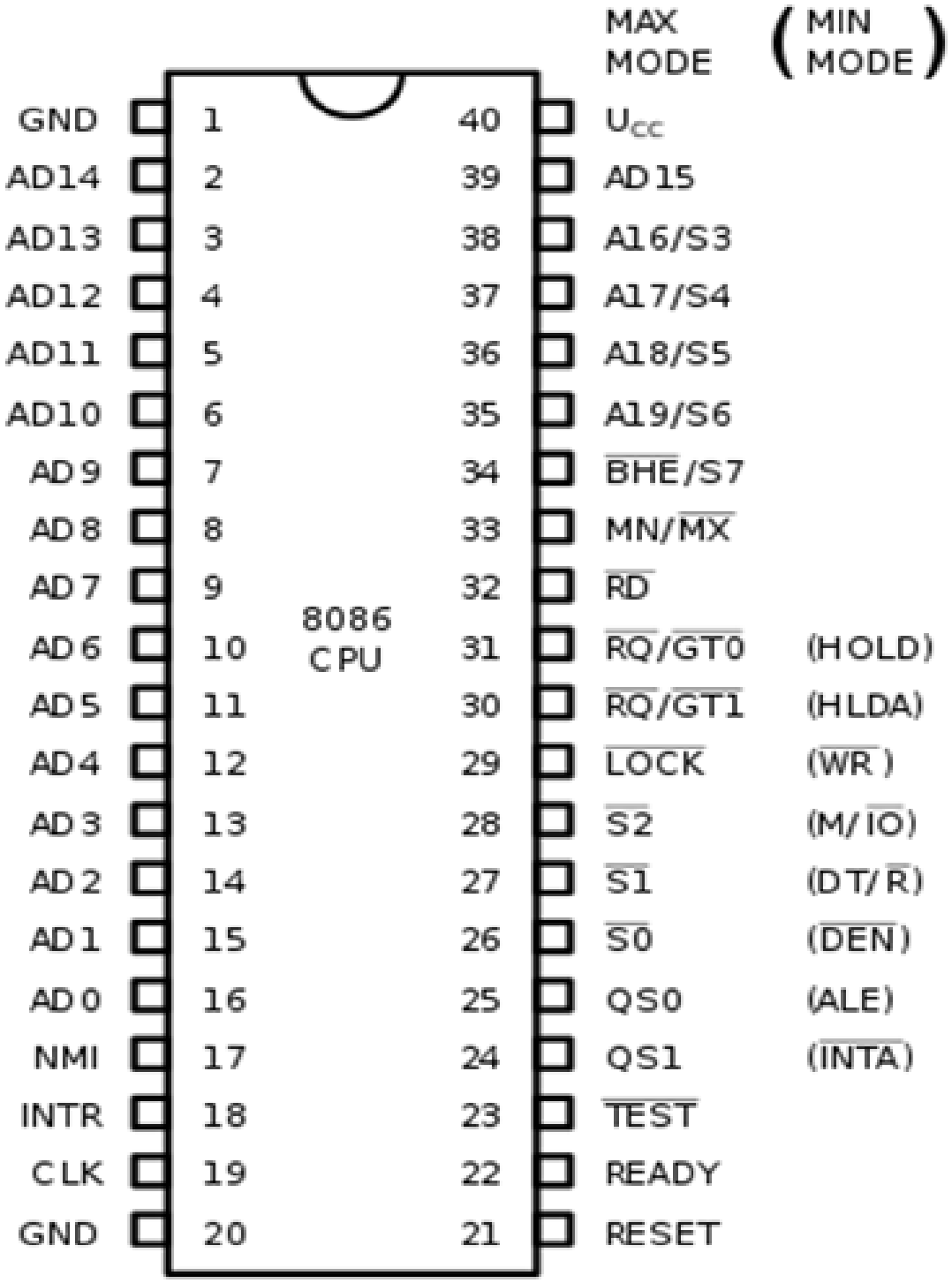
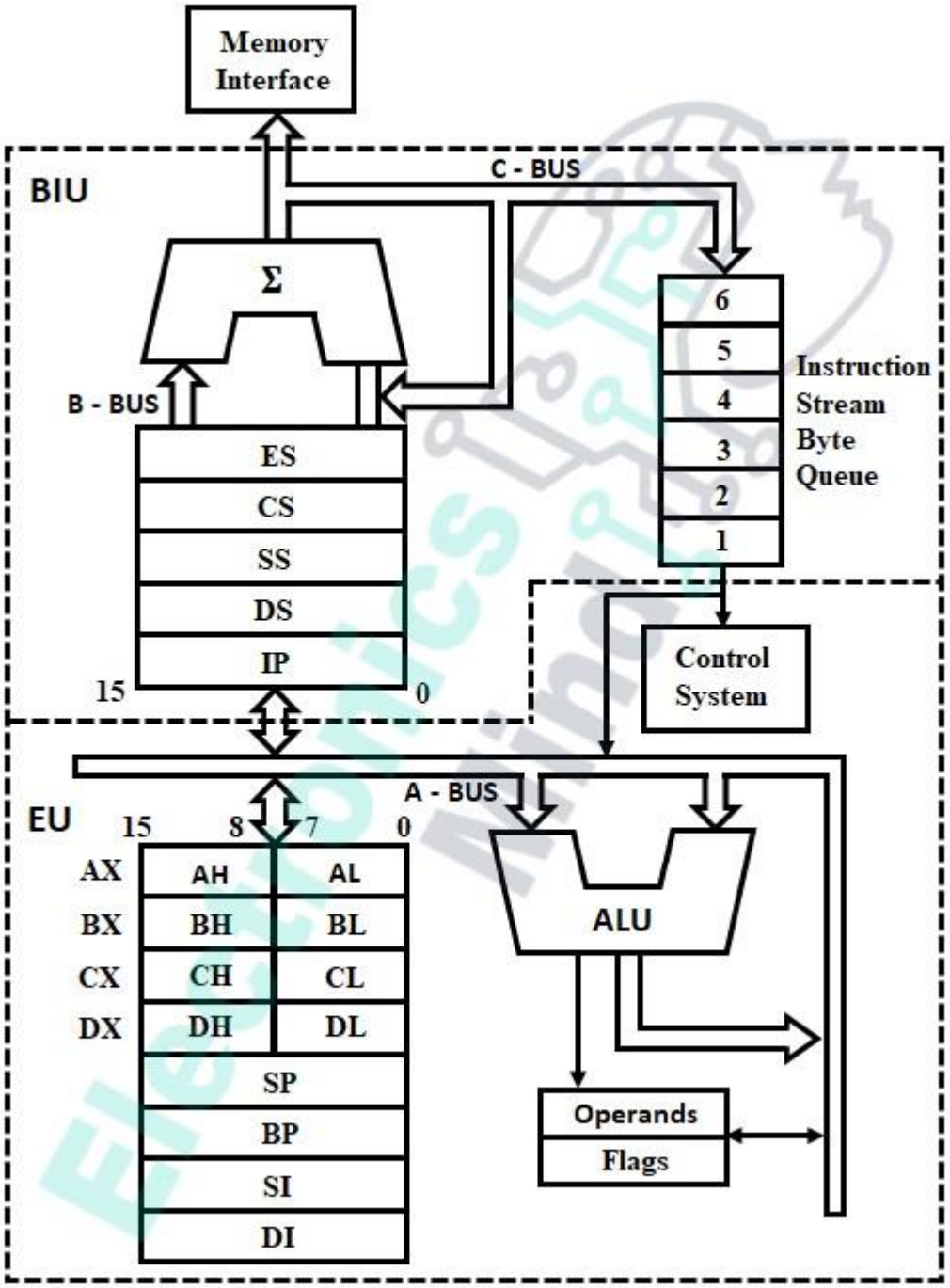
Chapter 1

8086 Architecture

1. Pin Diagram of 8086:



2. Architecture of 8086:



8086 Internal Architecture

3. Use of Status Signals in Maximum Mode:

S2	S1	S0	Status
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Passive

4. Features of 8086:

1. It is 40 pin IC.
2. HMOS technology is used.
3. 5V DC supply required.
4. 16-bit Data Lines.
5. 20-bit address line hence can address 2^{20}=1MB.
6. Operating clock frequency is 5MHz, 8MHz, 10MHz.
7. Multiplexed address and data lines (AD_0 - AD_{15}).
8. It can perform arithmetic operations like addition, subtraction, division, multiplication and logical operations like AND, OR, NOT.
9. 8086 have 6-bytes instruction queue (IQ).
10. Parallel or pipelining execution of BIU and EU.
11. It provides 256(2^8) software interrupts and two hardware interrupts (NMI and INTR).
12. It can access 2^{16} = 64 kilo I/O devices (56536 I/O Devices).
13. It supports multiuser, multiprogramming due to memory segmentation.
14. It consists of four segments as
    - Code Segment
    - Data Segment
    - Stack Segment
    - Extra Segment
15. It provides separate instruction set for string operations.
16. Powerful instruction set, can be programmed using high level language like c.

Chapter 2

Assembly Language Programming

To develop an assembly language program, we must know the program development steps such as defining and understanding the problem, and then represent it using standard way of algorithm or flowchart. Then the next step is the coding that actually develops the program.

To develop the assembly language program, we have to use different assembly language programming tools like editor, assembler, linker, and debugger.

The editor is used to create the source code which is further converted into an object code (.obj) by an assembler. This is called as assembling the program. Then from that object code, the executable code (.exe) is generated by a linker, so that the program can be run by the CPU. This is called as linking of the program. Andfinally, debug the program to find out the error, if any and troubleshoot these errors using the debugger.

For each type of language, there is a specific structure in which the source code of that particular language is to be developed and written. In the same way, to develop an assembly language program with assembler, we have to follow the specific structure of assembly language programming.

In order to edit, assemble, and link the program to create the executable code which is ready to run, we have to follow a specific sequence of steps provided by the structure of the programming language. This structure is called as model of 8086 assembly language program.

Model (Structure) of 8086 Assembly language Programs: -

1) To develop an assembly language program by following specific format and syntax, we have to use an assembler tool that may be either TASM (TurboAssembler) or MASM (Microsoft Assembler).
2) This consists of Assembler Directives, 8086 Assembly Language Instructions and Data, written in specific sequence and format.
3) There are three different models of Assembly Language Program, and theyare as follows —

   A) Assembly Language Programming Model using SEGMENT and ENDS Directives

B) Assembly Language Programming Model using . (Dot) Directive

C) Editing, Assembling, and Linking the Assembly Language Program

A) **Assembly Language Programming Model using SEGMENT and ENDS Directives: -**

1) This model uses different logical segments for data and variable declarations and initialization as well as for writing instructions.

z) We can use single or multiple code and data segment to develop the assembly language program.

3) Each logical segment starts with a SEGMENT directive and end with anENDS directive, and both are followed by the name of segment.

4) The format (structure) is as follows —

```
DATA SEGMENT

<VARIABLE DECLARATION STATEMENTS>
VAR1 DB .....
YAR1 DW .....


DATA    ENDS

CODE

SEGMENT
ASSUME  CS:CODE,
DS:DATASTART:


<ASSEMBLY LANGUAGE INSTRUCTIONS / EXECUTABLE STATME>
<PROCEDURES / MACROS>
CODE    ENDS
END START
```

5) The program starts with a declaration of logical data segment using the SEGMENT assembler directive.

6) This data segment consists of all variable declarations along with data initialization. It does not contain any assembly language instruction.

7) The data segment ends with a ENDS assembler directive.

8) Then the logical code segment is started with SEGMENT assembler directive.

9) This segment consists of ASSUME assembler directive, that indicates the name of the logical segment along with physical segment register.

10) After this, a label START is used, which indicates the entry point of the program execution. In other words, the execution of assembly language instructions starts from this label.

11) The logical code segment consists of assembly language instructions, procedures, and macros. It does not contain the any variable declarations and data initialization part.

12) This segment ends with an ENDS assembler directive.

13) After this ENDS directive, there is one more assembler directive known as END assembler directive that indicates end of the program. No other statement (instruction) is added (used) after an END directive (Except comment, if required).

B) **Assembly Language Programming Model using .(Dot) Directive: -**

1) This model is somewhat different than the SEGMENT and ENDS model.

2) It is used when the program uses a single code and single data segments inmemory which is tiny or small.

3) Here, the data segment starts with a .DATA (dot DATA) directive, and codesegment starts with a .CODE (dot CODE) directive.

11) This code segment ends with .EXIT directive. The use of .EXIT directive is optional.

12) END assembler directive is used to indicate end of the program. No other statement (instruction) is added (used) after an END directive (Except comment, if required).

Editing, Assembling, Linking, and Debugging the Assembly Language Program: -

1) For writing and executing an assembly language program as per above models, we need to use different tools for editing, assembling, linking, and debugging a program.

z) These tools are provided by a MASM or TASM assembler tools. Let us see, how the program is written and executed using these tools.

a) Editing a Program: -

1) For writing a new program, or editing an existing program, we need to use a tool called **EDITOR.**

2) We know that, an editor is a system program which allows us to create and edit a source file as per standard structure.

3) To use an editor tool in DOS system, we use an **EDIT** command with the name of program (file) as follows —

```
C:\MASM> EDIT PNAME.ASM        ;using MASM
                               OR C:\TASM> EDIT
PNAME.ASM                      ;using TASM
```

4) Using this command, an editor is opened. Then type the program and save it and then exit from the editor. Note that the program is saved in secondary memory at specified location with name PNAME and an extension .ASM.

b) Assembling a Program: -

1) After editing, the next step is to assemble the program using another toolcalled as an **ASSEMBLER.**

4) No need to use the ASSUME directive in this model, since the segments are automatically associated with their physical segment registers.

5) The format of this model is as follows —

```
.MODEL SMALL
.DATA

<VARIABLE DECLARATION STATEMENTS>YAR1 DB .....
YAR1 DW .....

.CODE
.S
T
A
R
T
U
P
S
T
A
R
T

<ASSEMBLY LANGUAGE INSTRUCTIONS / EXECUTABLE STATMENTS>
<PROCEDURES / MACROS>

E
X
I
T
E
N
D
```

6) The program starts with a declaration of logical data segment using the .DATA assembler directive.

7) This data segment consists of all variable declarations along with data initialization. It does not contain any assembly language instruction.

8) The data segment ends automatically when code segment starts using .CODE assembler directive.

9) After this, a .STARTUP is used, which indicates the entry point of the program execution. In other words, the execution of assembly language instructions starts from this label.

10) The logical code segment consists of assembly language instructions, procedures, and macros. It does not contain the any variable declarations and data initialization part.

11) An assembler is a system program that converts the source file provided by an editor into an object file. This process is known as assembling of program.

2) To do this, we use the following command for assembler —

C:\MASM> MASM PNAME.ASM ; using MASM OR

C:\TASM> TASM PNAME.ASM ; using TASM

3) If there are no errors then the object code is created by an assembler with name **PNAME.OBJ.** Then next step is to link the program using LINKER.

4) If there are errors, then first correct these errors using editor as per above mentioned editor command, save it and exit, then again assemble the program. Note that, no object code is created if the program has errors.

5) Repeat this process unless the error count becomes zero. Then process for next step i.e. linking of program. Note that, the object file is generated at the same location where the source file is stored, i.e. in secondary memory.

c) <u>Linking a Program: -</u>

1) After assembling the program, the next step is linking the program using one more tool called a **LINKER.**

2) A linker is a program that joins several object files into one large object file and converts this object file provided by an assembler into an executable file with extension .EXE. This process is called as linking of a program.

3) To use a linker (for linking), we use a command as —
C:\ MASM> LINK PNAME.OBJ ; using MASM OR
C:\TASM> TLINK PNAME.OBJ ; using TASM

4) After this command, the linker generates an executable file with extension **.EXE,** and the complete name of the file is **PNAME.EXE.**

5) Note that, the EXE file is also generated at the same location in secondary memory, where the source file and an object file is stored. The next step is to process that file with the debugger.

d) **Debugging a Program: -**

1j After generating an EXE file by the linker, the next and final step is to process that EXE file with system program known as **DEBUGGER** for execution.

2) The EXE file is stored in secondary memory, and for executi0n, that file must be loaded into main memory. This is done by a debugger.

3) A debugger is a system program that loads the program from secondary memory into main memory for the purpose of execution. It also starts the execution of the program.

4) A debugger allows us to debug a program. It also allows us check memory location contents, CPU register contents at run time. It also troubleshoots the errors, if any.

5) To use the debugger, the command used is -
C:\MASM> DEBUG PNAME.EXE ; using MASM OR
C:\TASM> TD PANEM.EXE ; using TASM

6) Using these commands, the execution of the program is started. If we use the MASM, then after this command a '—' is displayed at the next line, indicating that the DEBUG operation is successfully invoked, and further use the debug prompt for debugging commands.

7) A valid command is accepted using the enter key. Some of the commands

that we use for debugging are ’T’, 'D’, ’R', and ’q’ for tracing program execution, displaying memory contents in segment from specified offset, displaying all registers and flags, and quit the debug and return to DOS respectively.

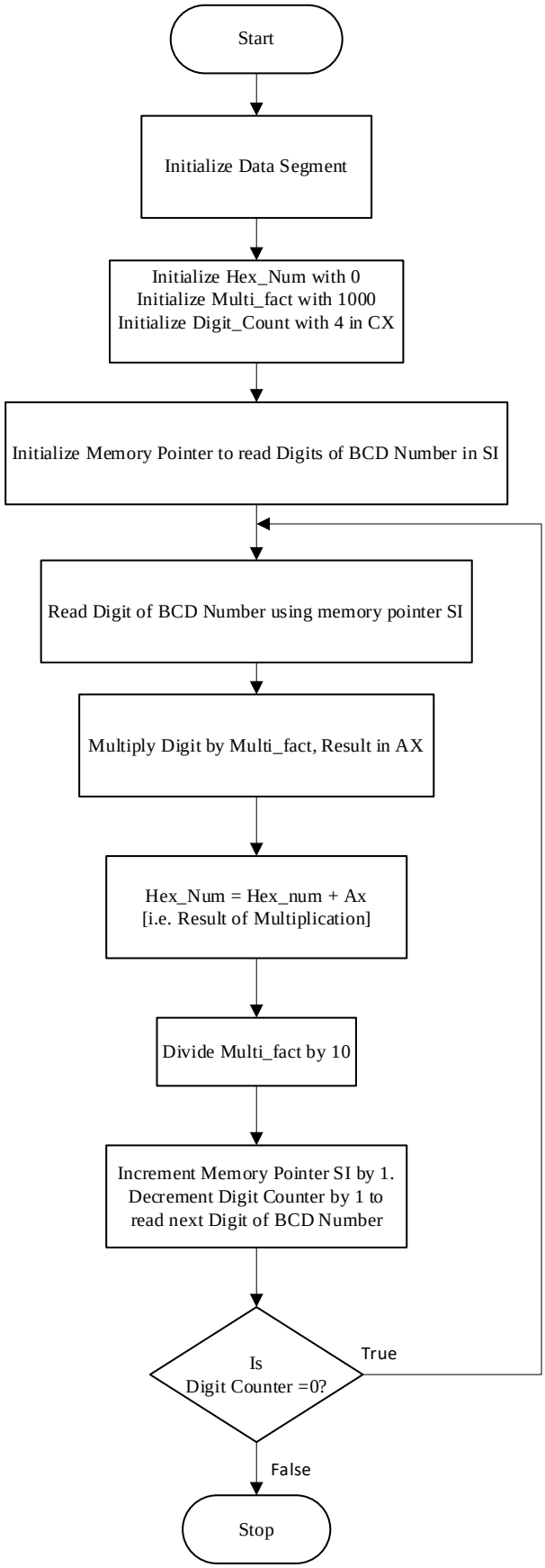
Chapter 3

Program

1. Algorithm:

Step 1: Initialize Data Segment
Step 2: Initialize Hex_Num with 0
Step 3: Initialize Multi_fact with 1000
Step 4: Initialize Digit_Count with 4 in CX
Step 5: Initialize Memory Pointer to read Digits of BCD Number in SI
Step 6: Read Digit of BCD Number using memory pointer in SI
Step 7: Multiply Digit by Multi_fact, Result in AX
Step 8: Hex_Num = Hex_Num + AX [i.e Result of Multiplication]
Step 9: Divide Multi_fact by 10
Step 10: Increment Memory Pointer SI by 1
Step 11: Decrement Digit Counter by 1 to read next Digitof BCD Number.
Step 12: Is counter =0?, if True then goto 6, if Fasle then goto 13
Step 13: Stop

2. Flowchart:

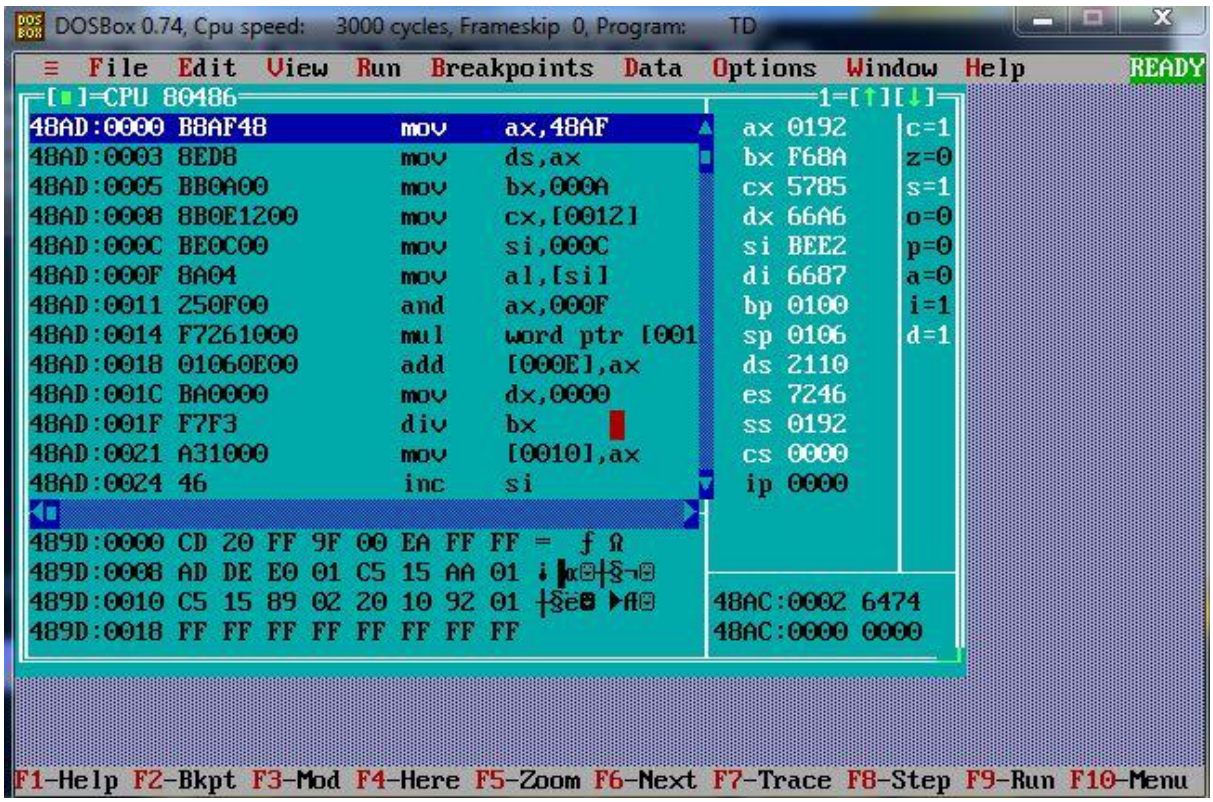

Start
Initialize Data Segment
Initialize Hex_Num with 0
Initialize Multi_fact with 1000
Initialize Digit_Count with 4 in CX
Initialize Memory Pointer to read Digits of BCD Number in SI
Read Digit of BCD Number using memory pointer SI
Multiply Digit by Multi_fact, Result in AX
Hex_Num = Hex_num + Ax
[i.e. Result of Multiplication]
Divide Multi_fact by 10
Increment Memory Pointer SI by 1.
Decrement Digit Counter by 1 to
read next Digit of BCD Number
Is
Digit Counter =0?
True
False
Stop

3. Code:

```
.MODEL SMALL
.DATA
        BINARY          DW  1234H
        HEX_NUM         DW  0
        MULT_FACT       DW  3e8h
        DIGIT_COUNT     DW  4

.CODE
        MOV AX,@DATA            ;Initialize Data Segment
        MOV DS,AX
        MOV BX,10               ;Initialize Division Factor
        MOV CX,DIGIT_COUNT      ;Digit Counter
        MOV SI,OFFSET BINARY    ;Initialize Memory Pointer
UP:
        MOV AL,[SI]             ;Read Digit of BCD Number
        AND AX,000FH            ;Mask Required Digit
        MUL MULT_FACT           ;Multiply by Multiplication Factor
        ADD HEX_NUM,AX          ;Add to HEX_NUM
        MOV AX,MULT_FACT        ;Change Multiplication Factor
        MOV DX,00
        DIV BX
        MOV MULT_FACT,AX
        INC SI                  ;Increment Memory Pointer
        LOOP UP                 ;Is a Last Digit if no Jump to UP
        MOV AH,4CH
        INT 21H
        ENDS
        END
```

4. Output:


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD
File Edit View Run Breakpoints Data Options Window Help READY
CPU 80486
48AD:0000 B8AF48 mov ax,48AF
48AD:0003 8ED8 mov ds,ax
48AD:0005 BB0A00 mov bx,000A
48AD:0008 8B0E1200 mov cx,[0012]
48AD:000C BE0C00 mov si,000C
48AD:000F 8A04 mov al,[si]
48AD:0011 250F00 and ax,000F
48AD:0014 F7261000 mul word ptr [001
48AD:0018 01060E00 add [000E],ax
48AD:001C BA0000 mov dx,0000
48AD:001F F7F3 div bx
48AD:0021 A31000 mov [0010],ax
48AD:0024 46 inc si
489D:0000 CD 20 FF 9F 00 EA FF FF
489D:0008 AD DE E0 01 C5 15 AA 01
489D:0010 C5 15 89 02 20 10 92 01
489D:0018 FF FF FF FF FF FF FF FF
ax 0192 c=1
bx F68A z=0
cx 5785 s=1
dx 66A6 o=0
si BEE2 p=0
di 6687 a=0
bp 0100 i=1
sp 0106 d=1
ds 2110
es 7246
ss 0192
cs 0000
ip 0000
48AC:0002 6474
48AC:0000 0000
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Chapter 4
CO's, PO's, PRO's

1. Course Outcome (COs):

a. Analyse the functional block diagram of 8086.
b. Write assembly language program for the given problem.
c. Use instructions for different addressing modes.
d. Develop assembly language program using assembler.
e. Develop assembly language program procedures, macros and modular programming approach.

2. Practical Outcome (POs):

a. Basic knowledge: Apply knowledge of basic mathematics, sciences and basic engineering to solve the broad-based Electronics related problems.
b. Discipline knowledge: Apply Computer Programming knowledge to solve broad-based Electronics related problems.
c. Experiments and practice: Plan to perform experiments and practices to use the results to solve broad-based Electronics related problems.
d. Engineering tools: Apply relevant Computer programming/electrical technologies and tools with an understanding of the limitations.
e. Individual and teamwork: Function effectively as a leader and team member in
f. diverse/ multidisciplinary teams.
g. Communication: Communicate effectively in oral and written form.
h. Life-long learning: Engage in independent and life-long learning activities in the context of technological changes also in the Electronics engineering and allied industry.

3. Practical Related Outcomes (PROs):

- Write ALP to convert BCD to equivalent Hex Number.
- Use instructions for different addressing modes
- Identify the various pins of the given microprocessor
- Use assembly language programming tools.

Chapter 5

References

A. Books:

i. 8086 Architecture and Programming By B.Ram.
Edition: 1st Edition Published in the year 2008.
Publication: Dhanpat Rai Publication.

ii. 8086 Architecture By A.P.Mathur.
Edition: 1st Edition Published in the year 1995.
Publication: New Age International Publications.

iii. 8086/8088 Microprocessor and Programming by Douglas Hall.
Edition: 1st Edition Published in the year 1999.
Publication: Prentice-Hall of India Pvt.ltd.

B. Websites:

Intel 8086 - Wikipedia

https://en.wikipedia.org/wiki/Intel_8086

Microprocessor - 8086 Overview (tutorialspoint.com)
https://www.tutorialspoint.com/microprocessor/microprocessor_8086_overview.htm