

**DEPARTMENT
OF
ELECTRONICS
&
COMMUNICATION
ENGINEERING**

Name of Student: _____

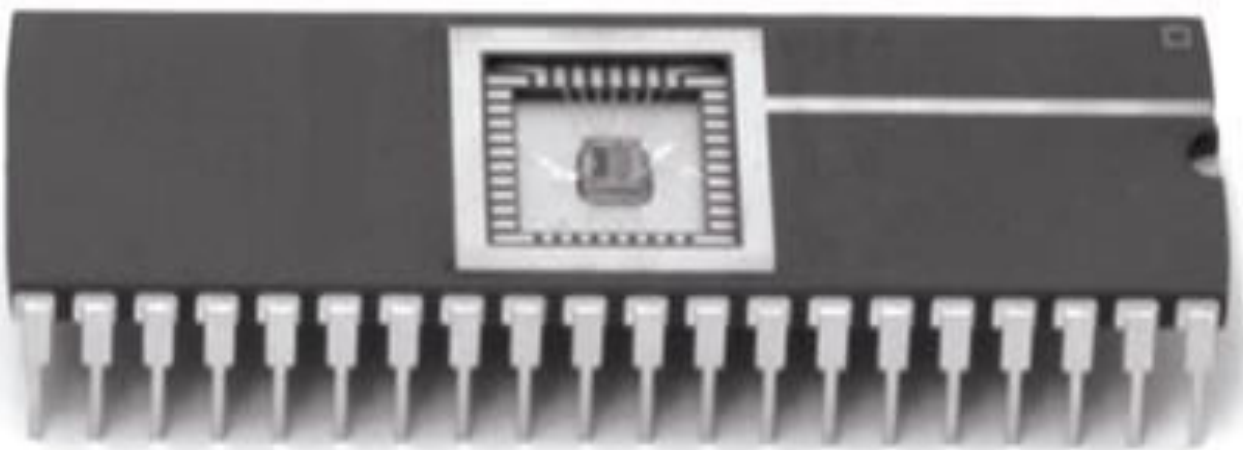
Enrollment No.: _____

Class: _____

Section: _____

Session: _____

**Microprocessor & Its Applications [EC-501]
Lab Manual**



Prepared By: Dr. Aparna Gupta

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE

Kalchuri Nagar, Raisen Road Bhopal (MP) 462023

Vision and Mission of the Department

Vision

To be world-wide recognized for adopting and keeping innovation and entrepreneurship mindset as abreast of learning to produce professionals as valuable, ethical and moral resource for industry and society.

Mission

- To establish an ecosystem where students could grow with innovative practices followed in communication engineering.
- To adopt the global approaches to transform the young aspirant into most acceptable engineering professional catering the modern challenges of the society in the most ethical with full of patriotic zeal.
- To facilitate and felicitate all the learners to have close interactions with the industry experts and researchers for keeping them updated of the current and future needs of the society.
- To develop the mindset of being innovative and entrepreneurial in the future engineering professionals.



ELETRONICS AND COMMUNICATION ENGINEERING

PSO

1. To analyze, design and develop solutions of real time problems and industry needs.
2. Ability of effectively communicating with the professionals and preparation of reports, documents and presentation while working in teams.
3. Knowledge and understanding of latest developments in the field of VLSI, Embedded system, Networking, Matlab and other major tools necessary for keeping pace with the industry.
4. Ability of solving complex engineering problems with ethical and law full approach to prevent the society and environment from adverse impacts.

DEPARTMENT OF ELECTRONIC AND COMMUNICATION ENGINEERING

PEOs

1. The graduate will have the knowledge and skills of analog and digital communication in providing necessary solutions to the real world problems.
2. The graduate will be able to design, develop, analyze and implement the modern tools and systems involving principles of electronics and telecommunication engineering.
3. The graduate will be following the ethical practices of the core industry and supporting software industry in providing most acceptable solution to the society.
4. The graduate will have the innovative mindset of learning and implementing the latest technological advancements and research outcomes in the electronic hardware and software to keep pace with the rapid developments in socio economic world.

INDEX

Name of Student: _____ Enrollment No.: _____

S.No.	Title of the Experiment	Date of Experiment	Date of Experiment	Remark
1	Installation and Study of 8086 assemblers/ compilers			
2	Perform addition and subtraction of 8-bit numbers.			
3	Perform addition, subtraction, multiplication and division of 16-bit numbers			
4	Perform Logical Operation using 8086			
5	Perform Universal Gate Operation using 8086			
6	Find 1's and 2's complements			
7	Find the average of numbers			
8	Exchanging the data of registers			
9	Finding factorial of a number			
10	Stepper Motor Rotation			

Code of Conducts for the Laboratory

- All bags must be left at the indicated place.
- The lab timetable must be strictly followed.
- Be **PUNCTUAL** for your laboratory session.
- Noise must be kept to a minimum.
- Workspace must be kept clean and tidy at all time.
- Handle the experiment kit and interfacing kits with care.
- All students are liable for any damage to the accessories due to their own negligence.
- Students are strictly **PROHIBITED** from taking out any items from the laboratory.
- Students are **NOT** allowed to work alone in the laboratory without the Lab Supervisor
- Report immediately to the Lab Supervisor if any malfunction of the accessories, is there.
- Before leaving the lab Place the stools properly.
- Please check the laboratory notice board regularly for updates.

Experiment -1 Installation and Study of 8086 assemblers/compiler

Aim: Installation and study of 8086 assemblers/compiler.

- (a) Debug
- (b) Emu86
- (c) NASM

Theory:

To understand the architecture of 8086 microprocessor.

Enable to write small assembly language programs using DEBUG utility program.

To understand the design flows of NASM and Emu8086.

Introduction:

Assembly language programs resemble a compiled language programs, like PASCAL. In compiled languages the user first creates a source file, which is a text file of entire program. The compiler changes it into machine language instructions. In the compiled language program the entire program is transformed into machine language at once. Assembly Language is converted into executable machine code by a utility program referred to as an assembler, the conversion process is referred to as assembly, or assembling the code.

Microprocessor:

A silicon chip that contains a CPU. In the world of personal computers, the terms microprocessor and CPU are used interchangeably. A Microprocessor (sometimes abbreviated μP) is a digital electronic component with miniaturized transistors on a single semiconductor integrated circuit (IC). Microprocessors also control the logic of almost all digital devices, from clock radios to fuel-injection systems for automobiles. A **microprocessor** incorporates the functions of a computer's CPU on a single integrated circuit, or at its most a few integrated circuits. All modern CPUs are microprocessors making the micro-prefix redundant.

Central Processing Unit(CPU) on a single silicon chip (called microchip) that can be 'Soft Wired' by using different programming instructions. When coupled with memory and input-output devices, a microprocessor becomes a microcomputer. INTEL 4004, one of the most earliest microprocessors, had a 4-bit data bus and could address 4.5 kilobytes of memory. The first mass produced Microprocessor was Intel 8080 than in 1973 cost about \$400.

Three basic characters differentiate microprocessors: -

- INSTRUCTION SET: - The set of the instructions that microprocessor can execute.
- BANDWIDTH: - The number of bits processed in a single instruction.
- CLOCK SPEED: - Given in Megahertz (MHz), the clock speed determines how many instructions per second the processor can execute.

In addition to bandwidth and clock speed, microprocessors are classified as being either RISC (reduced instruction set computer) or CISC (complex instruction set computer).



Two basic characteristics differentiate microprocessors:

Instruction set: The set of instructions that the microprocessor can execute.

Bus width: The number of bits processed in a single instruction.

8086 Architecture:

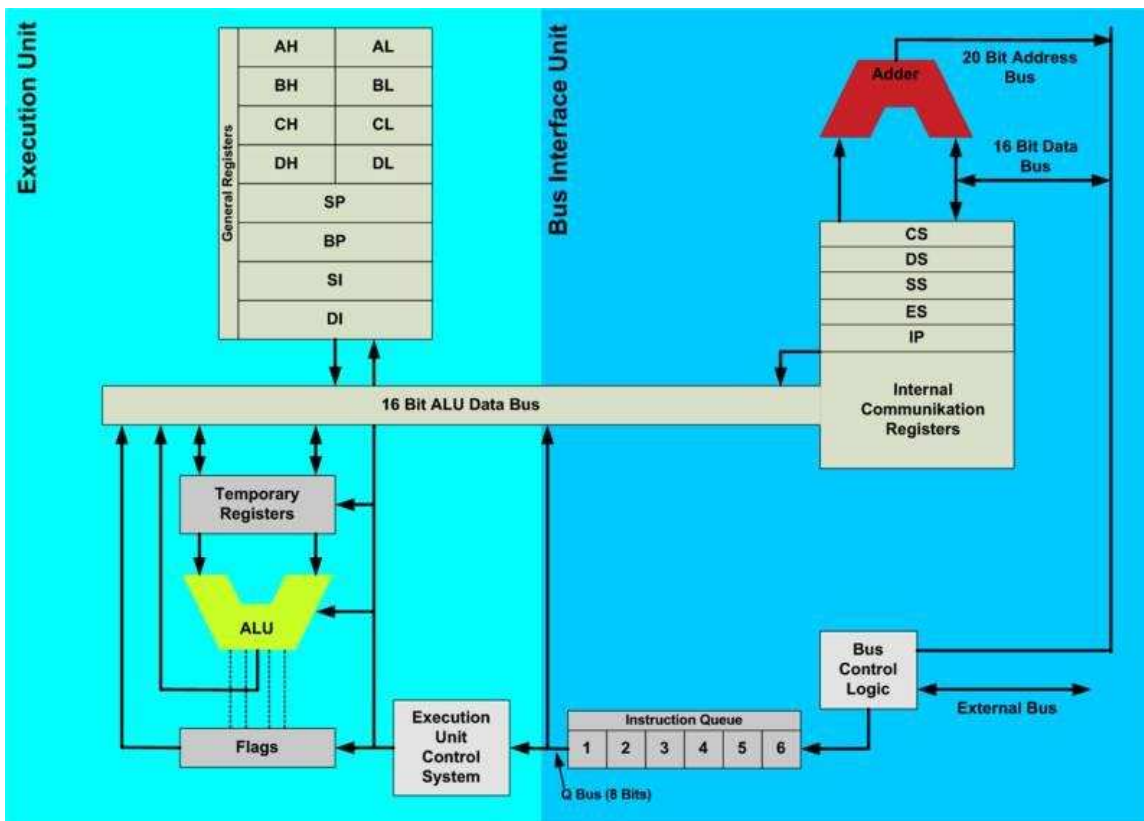


Figure 1. 8086 Architecture

- 8086 is a 16 bit μ p.

- 8086 has a 20 bit address bus can access upto 2^{20} memory locations (1 MB) .
- It can support up to 64K I/O ports.
- It provides 14, 16-bit registers.
- It has multiplexed address and data bus AD0-AD15 and A16–A19.
- It requires single phase clock with 33% duty cycle to provide internal timing.
- 8086 is designed to operate in two modes, Minimum and Maximum.
- It can prefetches up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40-pin dual in line package.

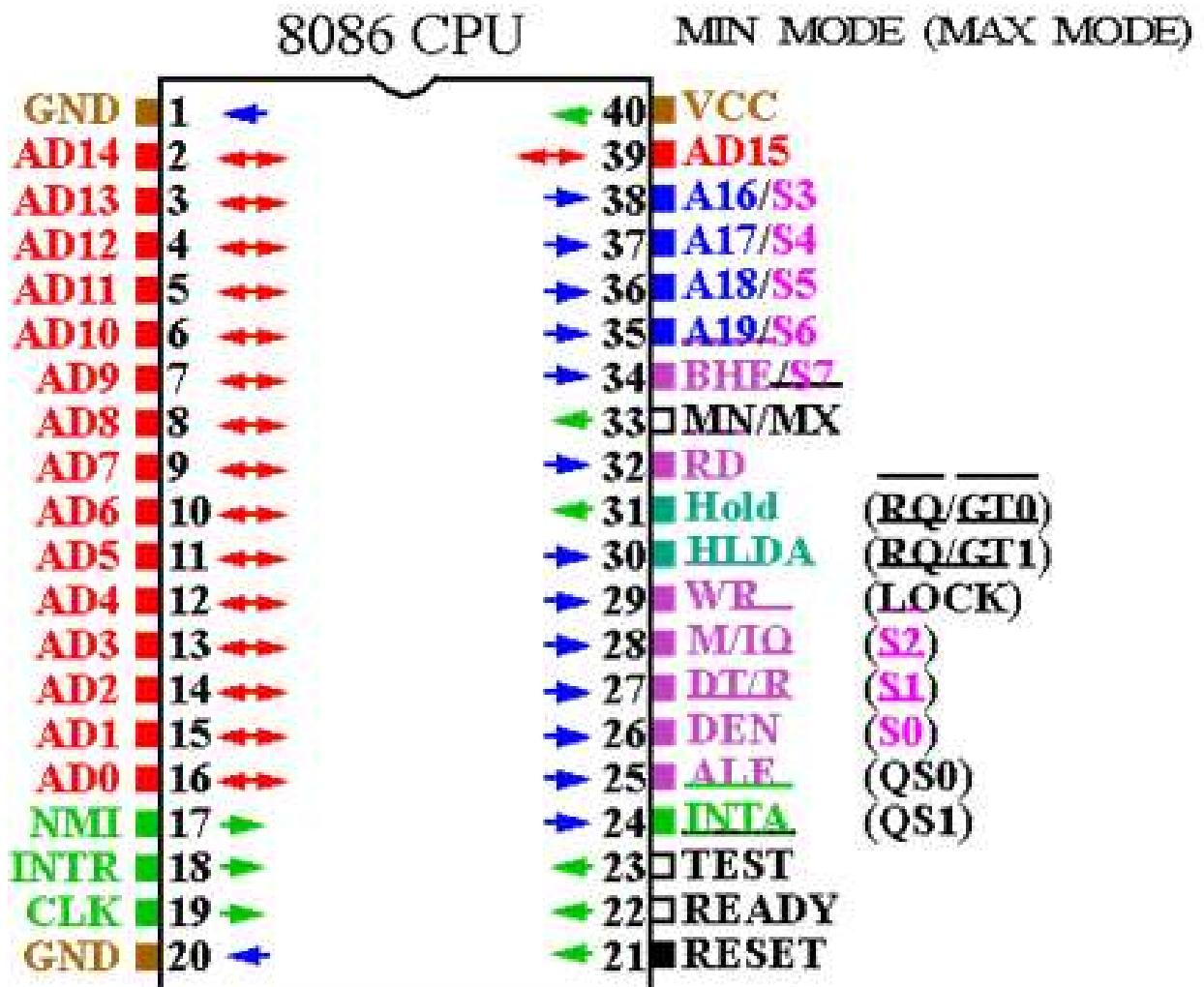


Figure 2. 8086 Pin Diagram

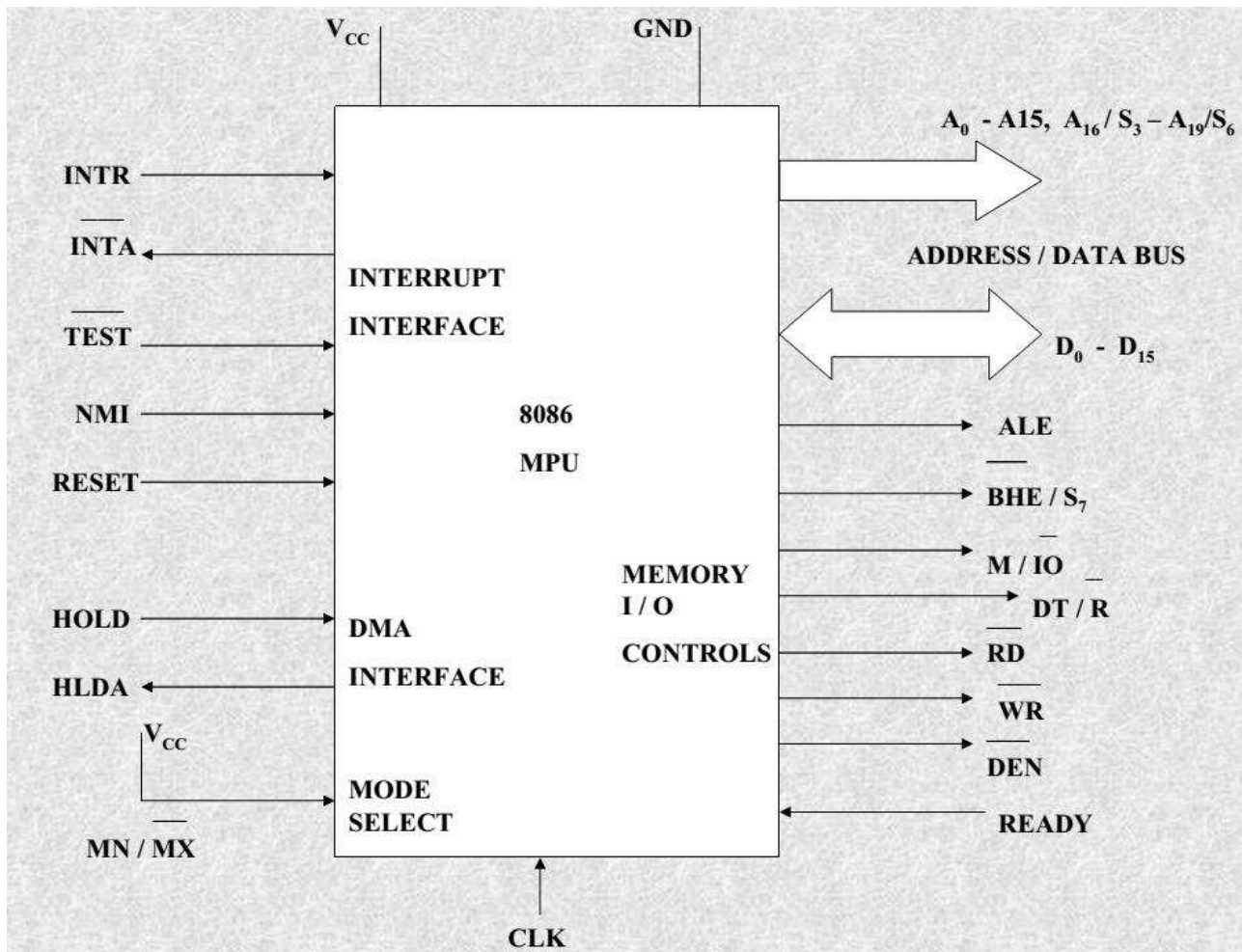


Figure 3. 8086 Signal Diagram

Debug for 8086:

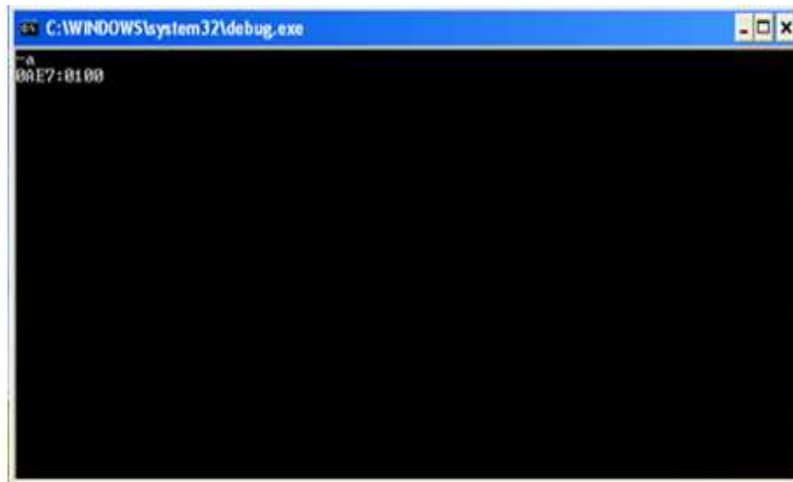


Figure4. Debug- DOS based emulator for Windows

A debugger displays the contents of memory and lets us view the registers and variables as they change. We can step through a program one line at a time called tracing, making it easier to find logic errors.

Debugger Functions

Some of the most rudimentary functions that a debugger can perform are the following:

- Assemble short programs
- View a program's source code, along with its machine code
- View the CPU registers and flags
- Trace or execute a program, watching variables for changes
- Enter new values into memory
- Search for binary or ASCII values in memory
- Move a block of memory from one location to another
- Fill a block of memory
- Load and write disk files

Debugger Commands

- A Assemble a program using instruction mnemonics
- R Display the contents of registers and flags
- T Trace a single instruction
- U Disassemble memory into assembler mnemonics
- D Dump (display) the contents of memory
- E Enter bytes into memory
- Q Quit DEBUG and return to DOS
- L Load data from disk
- W Write data from memory to disk
- N Create a filename for use by the L and W commands

EMU8086:

The integrated 8086 assembler can generate console programs that can be executed on any computer that runs x86 machine code (Intel/AMD architecture)

The architecture of the 8086 Intel microprocessor is called "**Von Neumann architecture**" after the mathematician who conceived of the design.

A CPU can interpret the contents of memory as either instructions or data; there's no difference in the individual bytes of memory, only the way in which they're arranged. Because of this, it's even possible for programs to re-write their own instructions, then execute the instructions they've changed.

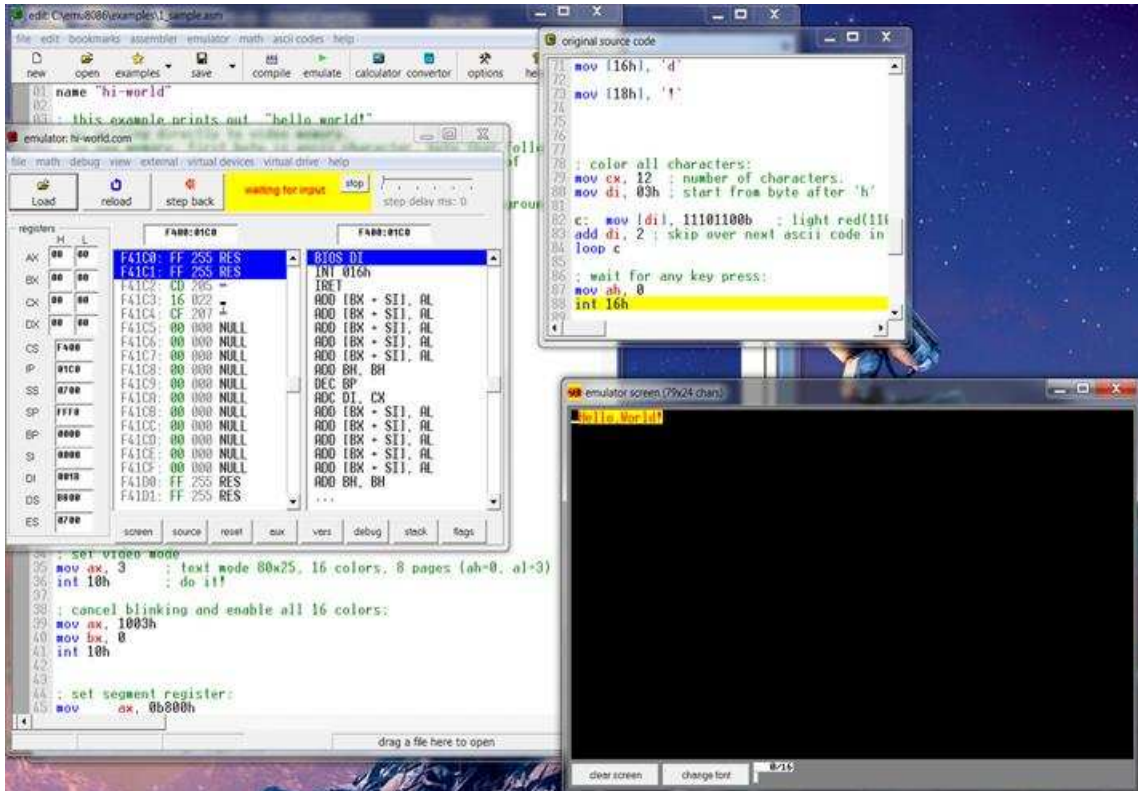


Figure 5.emu 8086 compiled output

NASM: The Netwide Assembler

The Netwide Assembler, NASM, is an 80x86 and x86-64 assembler designed for portability and modularity. It supports a range of object file formats, including Linux and *BSD a.out, ELF, COFF, Mach-O, Microsoft 16-bit OBJ, Win32 and Win64. It will also output plain binary files. Its syntax is designed to be simple and easy to understand, similar to Intel's but less complex. It supports all currently known x86 architectural extensions, and has strong support for macros.

To assemble a file, you issue a command of the form

nasm -f <format><filename> [-o <output>]

For example,

nasm -f elf myfile.asm

will assemble myfile.asm into an ELF object file myfile.o. And

nasm -f bin myfile.asm -o myfile.com

will assemble myfile.asm into a raw binary file myfile.com.

To produce a listing file, with the hex codes output from NASM displayed on the left of the original sources, use the -l option to give a listing file name, for example:

nasm -f coff myfile.asm -l myfile.lst

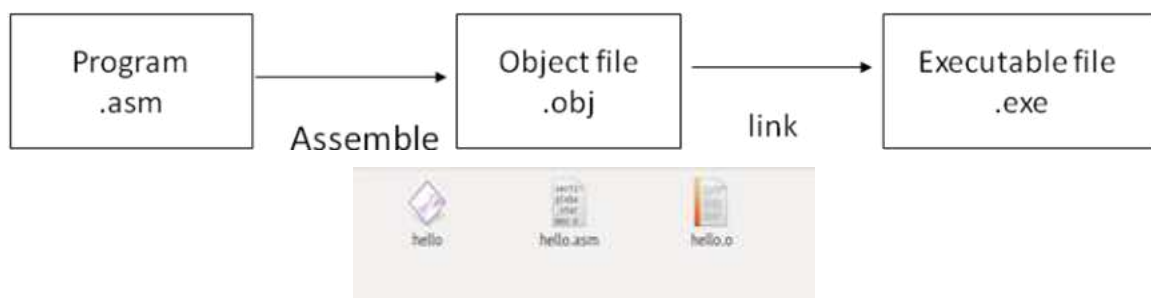


Figure 6. Flow of program development

```

ayoush@ubuntu: ~/coding/asm
_start:
  mov eax, 4
  mov ebx, 1
  mov ecx, string
  mov edx, length
  int 0ah

  mov eax, 1
  mov ebx, 0
  int 0ah

section .data
string: db 'Hello Ayoush', 0ah
length: equ 13

ayoush@ubuntu:~/coding/asm$ nasm -f elf -o hello.o hello.asm
ayoush@ubuntu:~/coding/asm$ ls -l hello.o
-rw-rw-r-- 1 ayoush ayoush 672 Jun 22 13:18 hello.o
ayoush@ubuntu:~/coding/asm$ ld hello.o -o hello
ld: warning: cannot find entry symbol _start; defaulting to 0000000000400000
ayoush@ubuntu:~/coding/asm$ ./hello
Hello Ayoush
ayoush@ubuntu:~/coding/asm$
  
```

Figure 7. Snapshot of NASM Assembler



Summary:

The native language is machine language (using 0,1 to represent the operation). A single machine instruction can take up one or more bytes of code. Assembly language is used to write the program using alphanumeric symbols (or mnemonic), eg. ADD, MOV, PUSH etc. The program will then be assembled (similar to compiled) and linked into an executable program. The executable program could be .com, .exe, or .bin files. An **assembly language** is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions.

Result: We have studied the installation & assembling of 8086 compilers.

Signature of Faculty

Experiment -2

Perform addition and subtraction of 8-bit numbers.

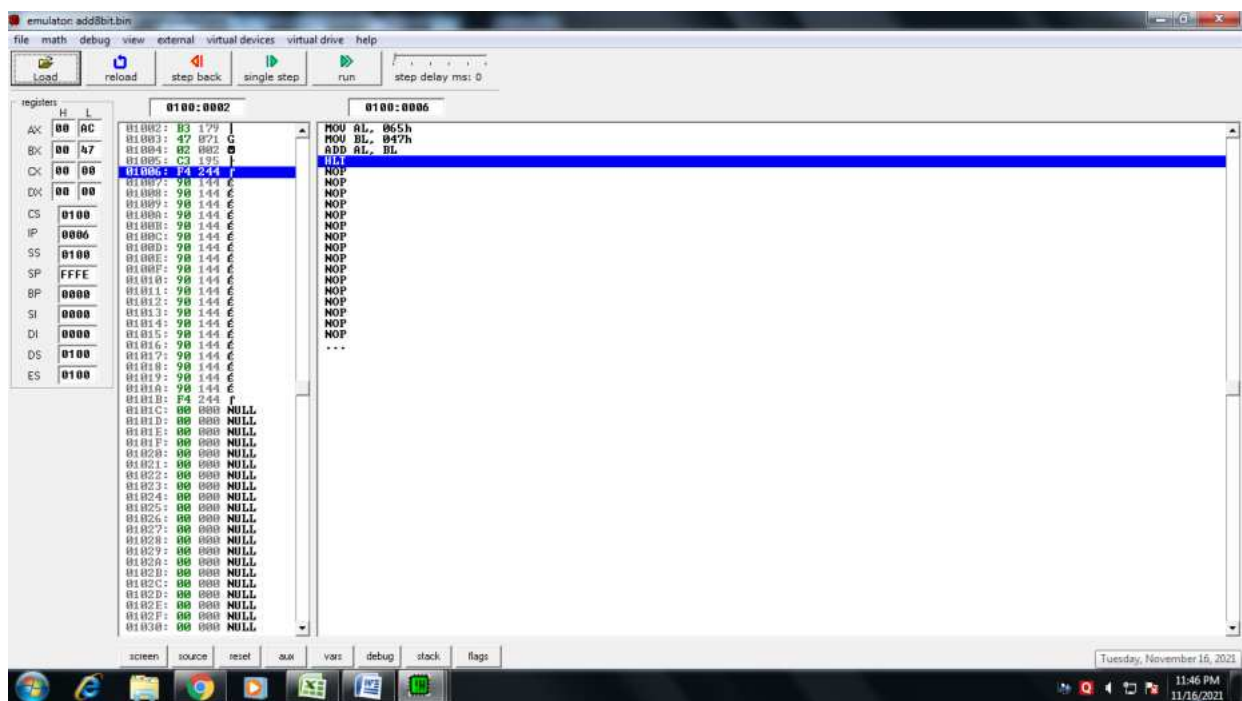
Aim: Write a program in 8086 to perform addition and subtraction of 8-bit numbers.

Software Used: EMU8086

Programs:

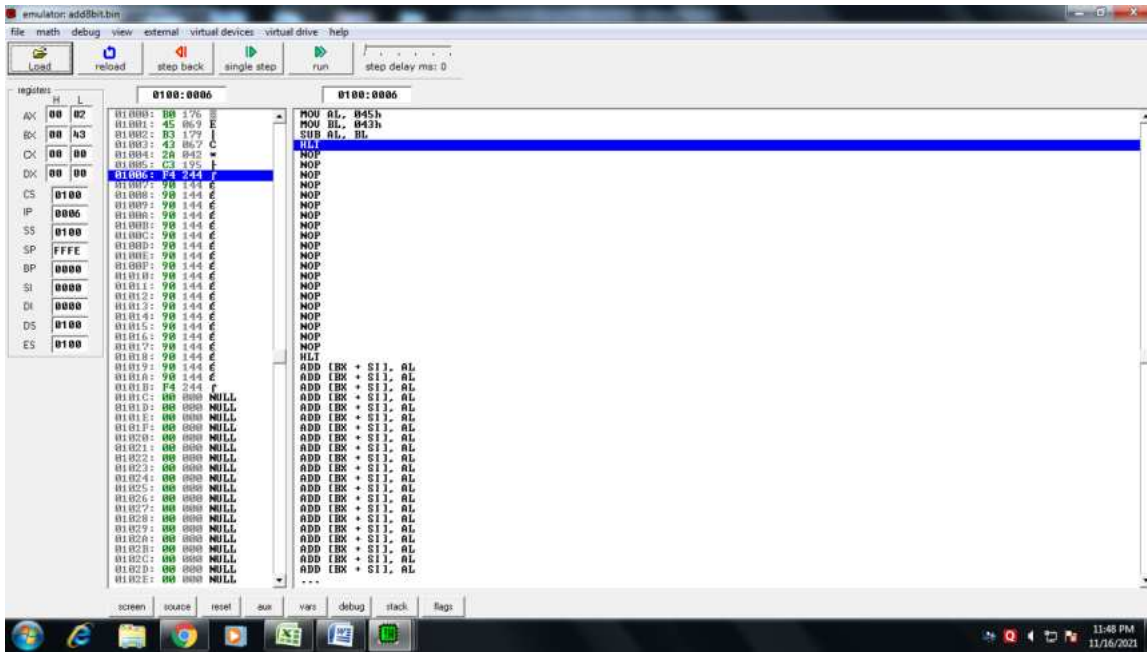
a) **Addition:**

```
MOV AL, 65H;
MOV BL, 47H;
ADD AL, BL;
HLT;
```



b) **Subtraction:**

```
MOV AL, 45H;
MOV BL, 43H;
SUB AL, BL;
HLT;
```

Result: We have performed addition and subtraction of 8-bit numbers.

Signature of Faculty

Experiment - 3

Perform addition, subtraction, multiplication and division of 16-bit numbers.

Aim: Write a program in 8086 to perform:

- a) addition of two 16-bit numbers
- b) subtraction of two 16-bit numbers
- c) multiplication of two 16-bit numbers
- d) division of two 16-bit numbers

Software Used: Emu8086

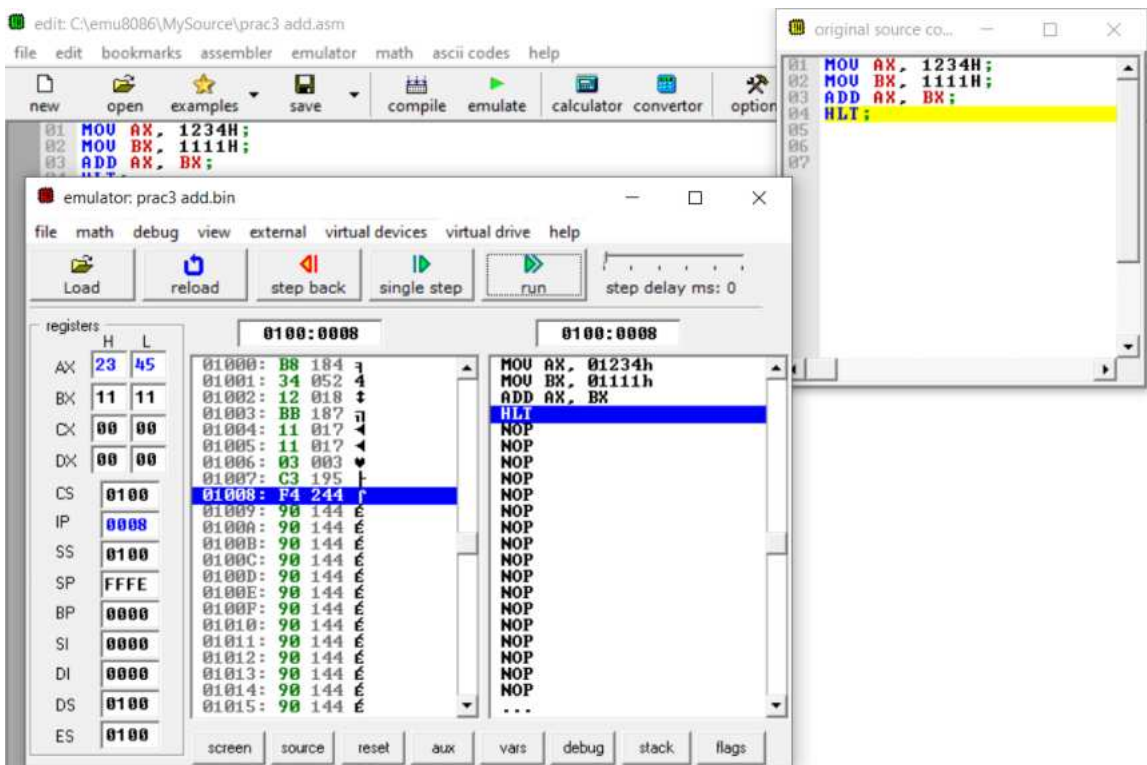
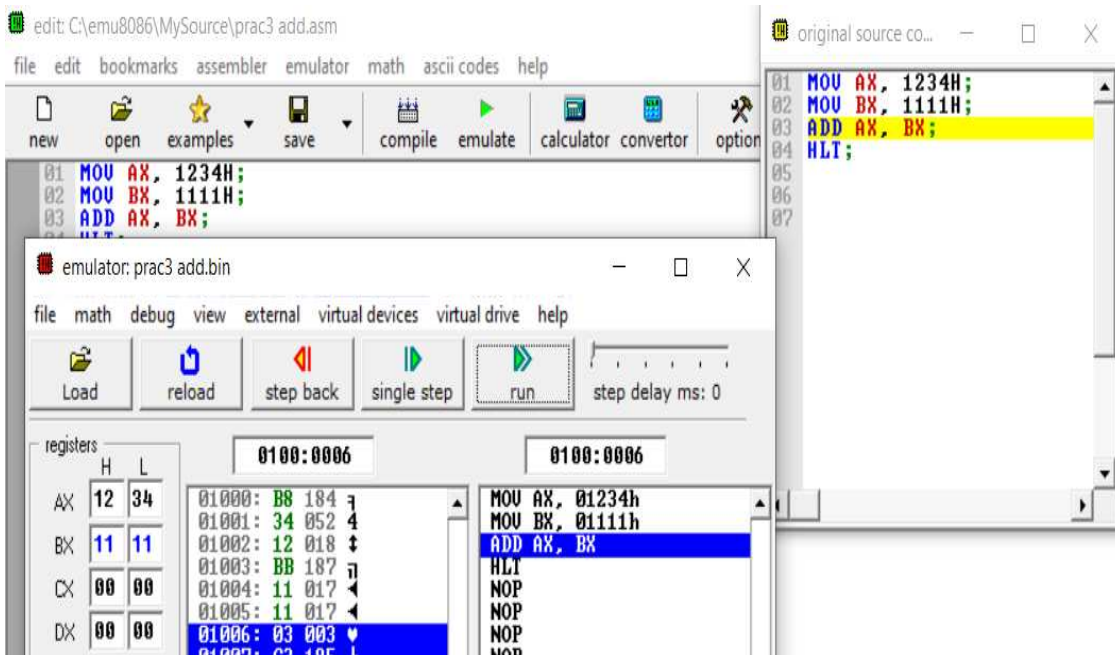
Programs:

a) **Addition:**

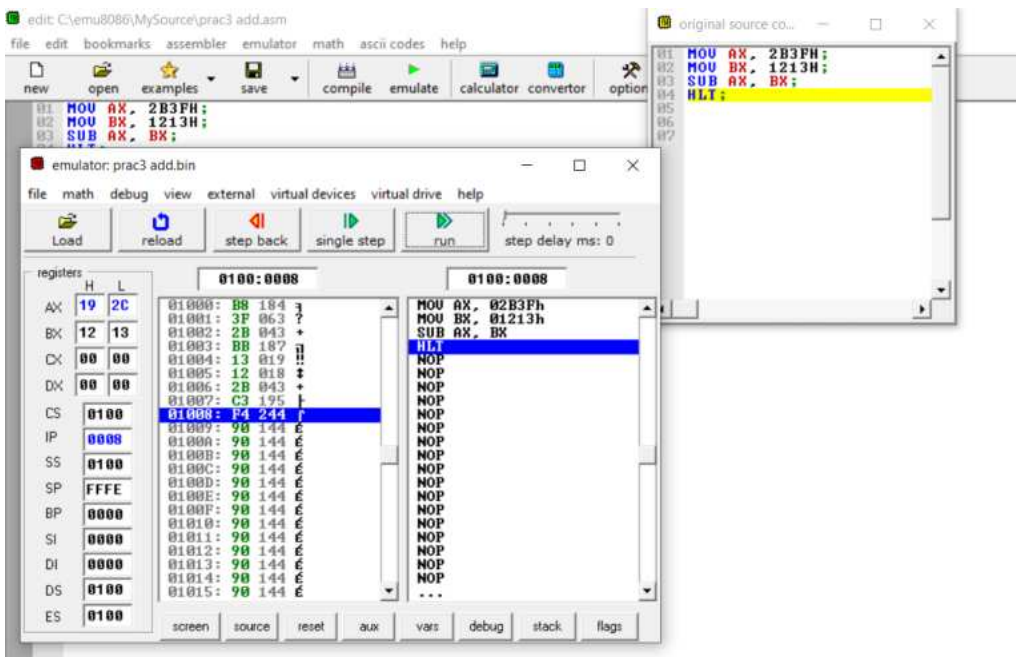
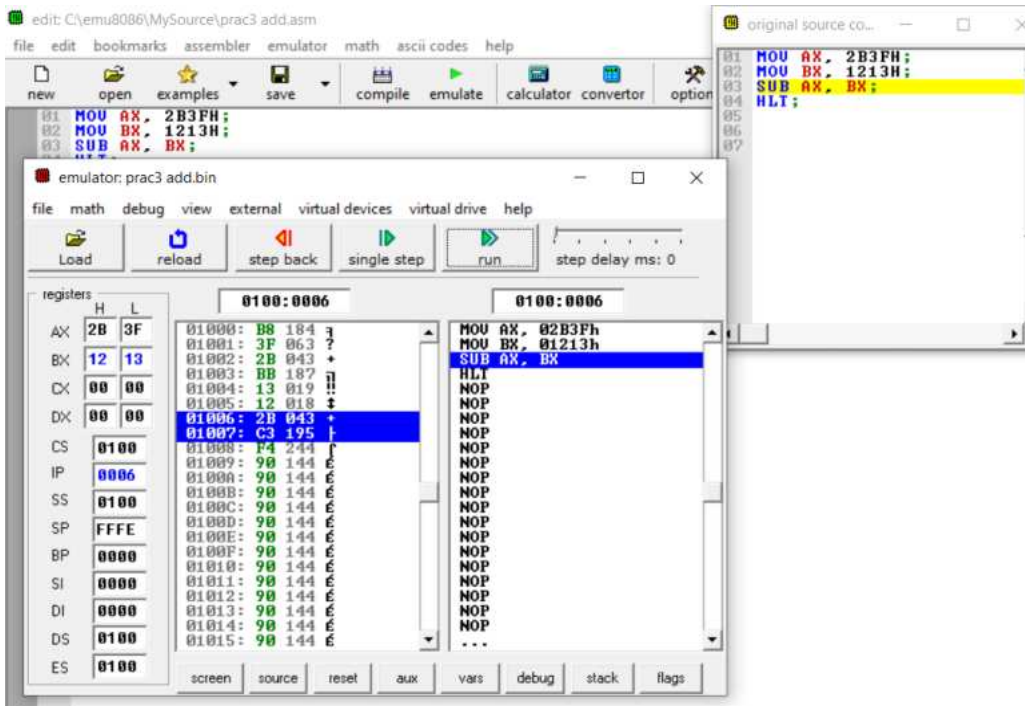
```
MOV AX, 1234H;  
MOV BX, 1111H;  
ADD AX, BX;  
HLT;
```



```
01 MOV AX, 1234H;  
02 MOV BX, 1111H;  
03 ADD AX, BX;  
04 HLT;  
05
```



- b) **Subtraction:**
 MOV AX, 2B3FH;
 MOV BX, 1213H;
 SUB AX, BX;
 HLT;



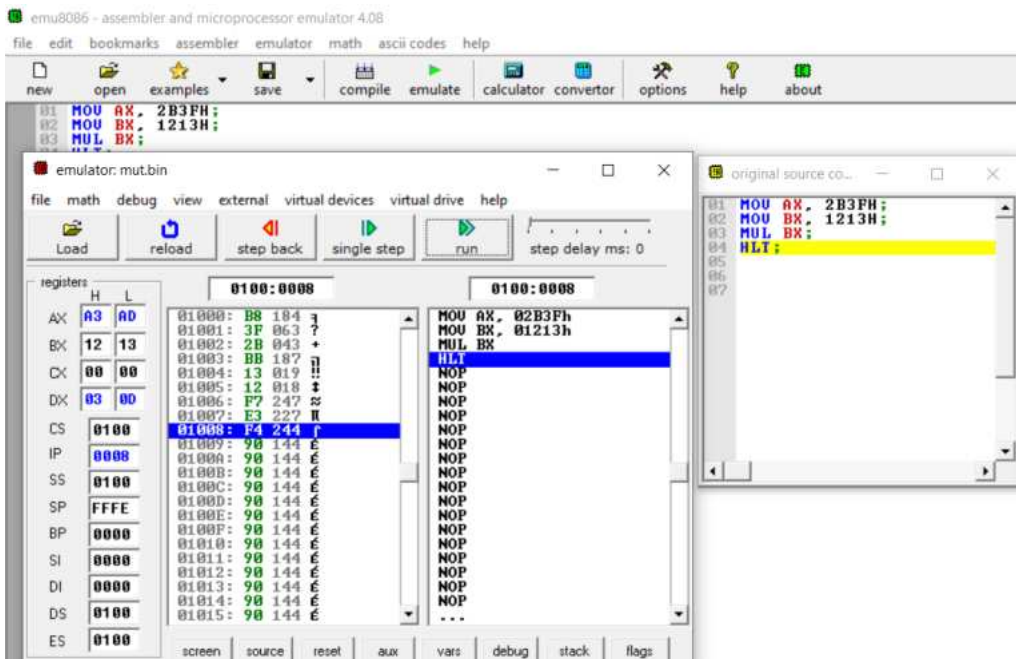
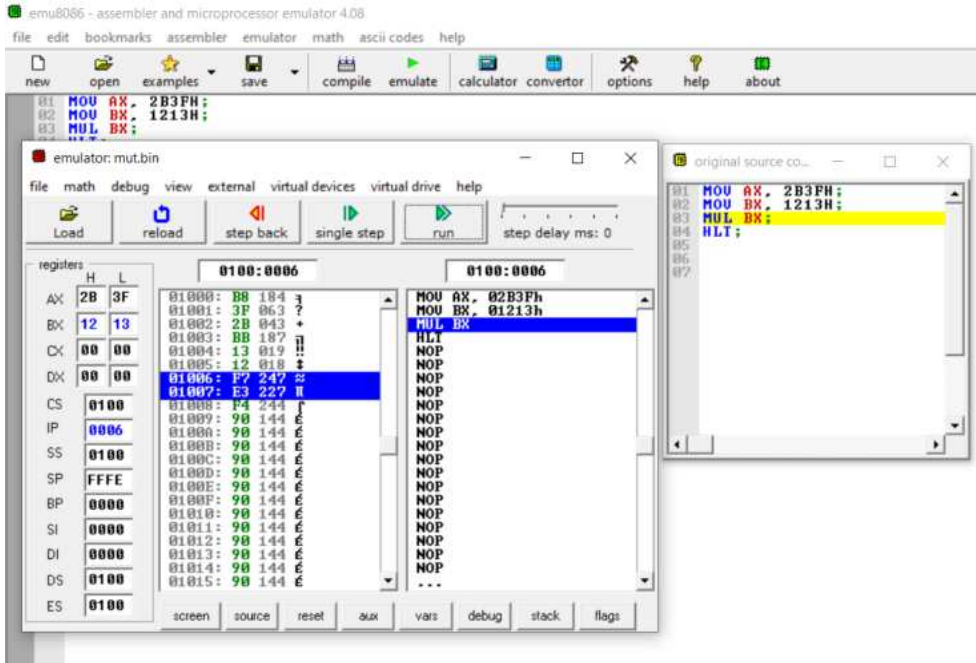
c) **Multiplication:**

MOV AX, 2B3FH;

MOV BX, 1213H;

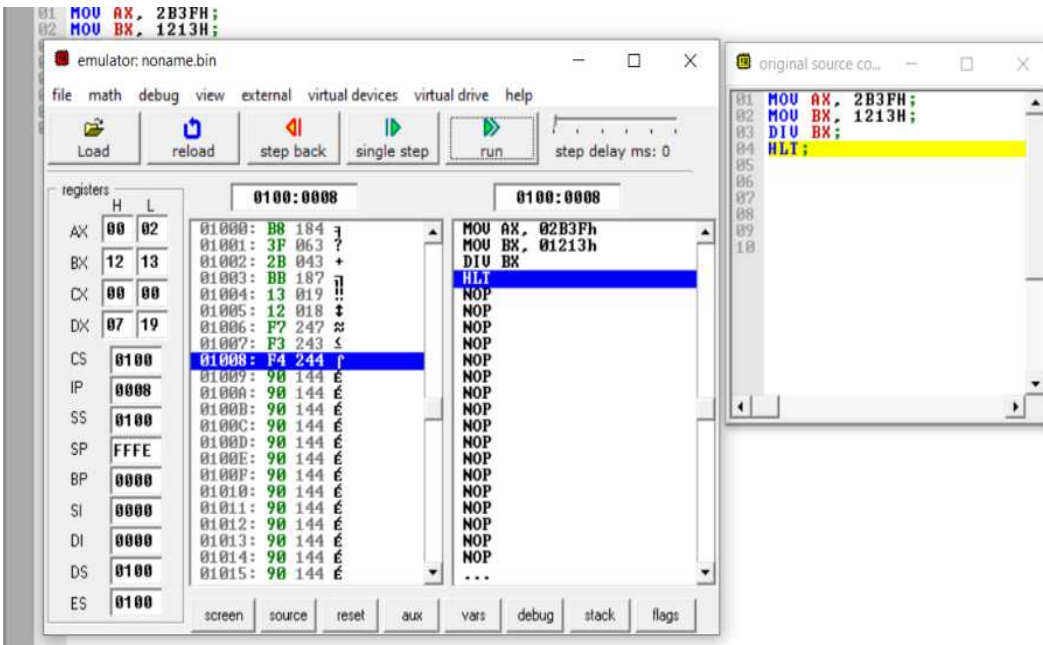
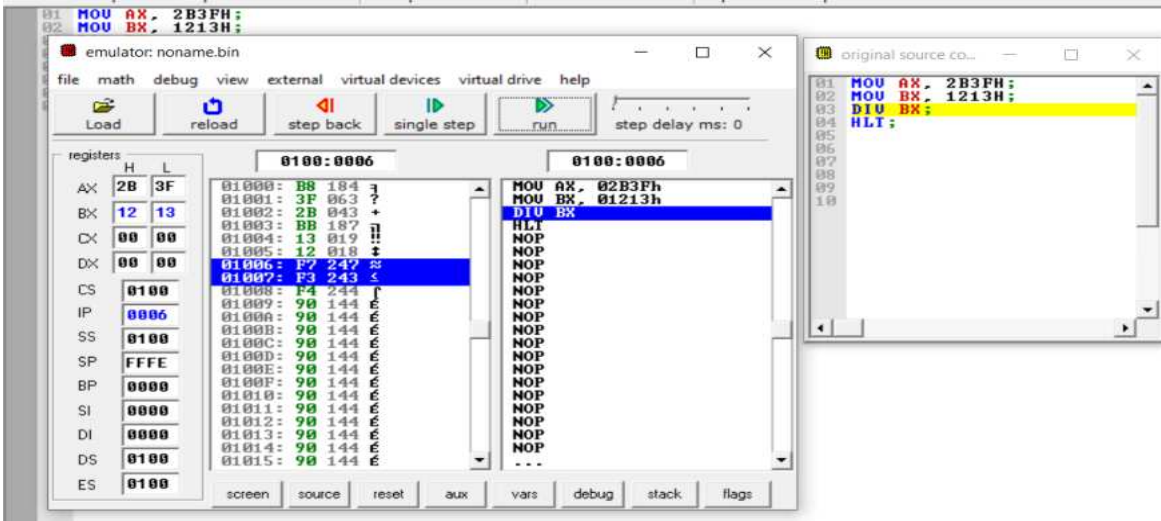
MUL BX;

HLT;



d) Division:

MOV AX, 2B3FH;
 MOV BX, 1213H;
 DIV BX;
 HLT;



Result: We have performed addition, subtraction, multiplication and division of 16-bit numbers

Signature of Faculty

Experiment - 4 Perform Logical Operation using 8086

Aim: Write a program in 8086 to perform various logical operation.

Software Used: EMU8086

Theory:

a) Not Operation

1. Start debug utility program on windows xp or Linux.
2. Take operand-1 into register AX.
3. Use logical operation instruction NOT for Not operation.
4. Store the result into register AX.
5. Program halted

b) AND Operation

1. Start debug utility program on windows xp or Linux.
2. Take operand-1 into register AX.
3. Take operand-2 into register BX.
4. Use logical operation instruction AND for AND operation.
5. Store the result into register AX.
6. Program halted

c) OR Operation

1. Start debug utility program on windows xp or Linux.
2. Take operand-1 into register AX.
3. Take operand-2 into register BX.
4. Use logical operation instruction OR for OR operation.
5. Store the result into register AX.
6. Program halted

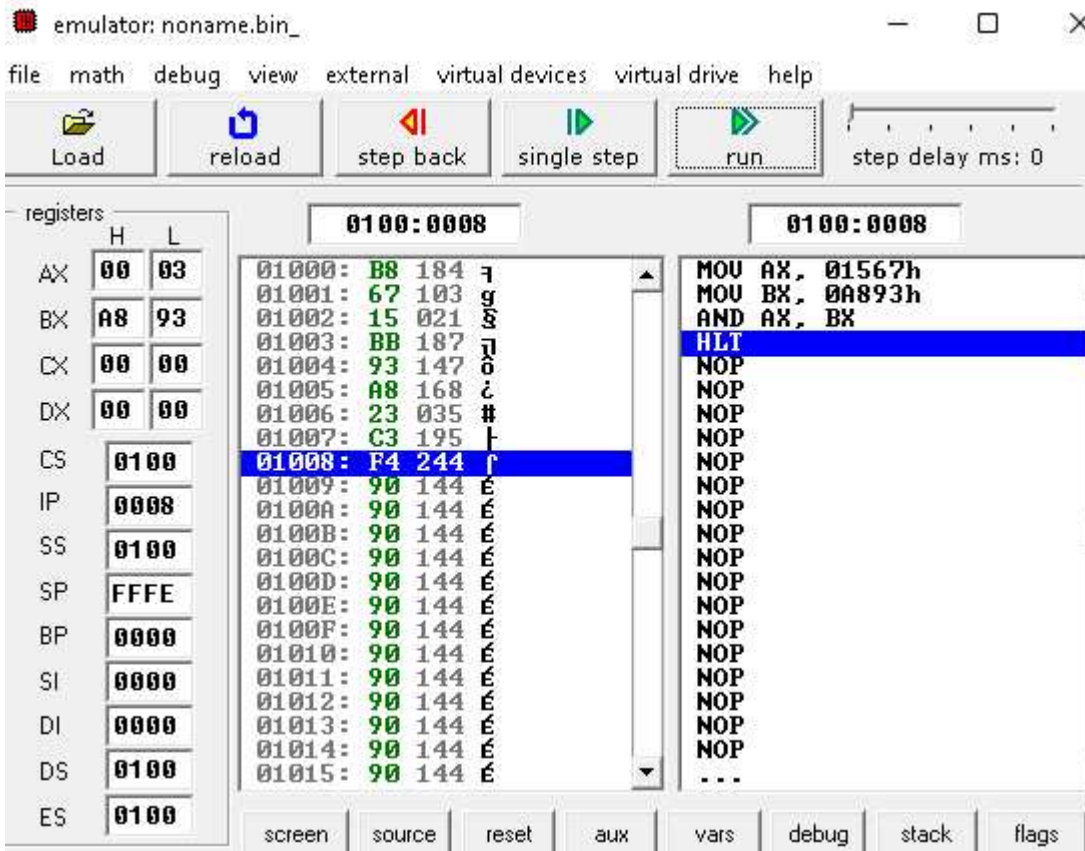
d) XOR Operation

1. Start debug utility program on windows xp or Linux.
2. Take operand-1 into register AX.
3. Take operand-2 into register BX.
4. Use logical operation instruction XOR for XOR operation.
5. Store the result into register AX.
6. Program halted

Programs:

i) NOT operation:


```
MOV BX, 0A893H;
AND AX, BX;
HLT;
```



The screenshot shows an emulator window titled "emulator: noname.bin_". The interface includes a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help) and a toolbar with buttons for Load, reload, step back, single step, and run. A "step delay ms: 0" slider is also present.

On the left, a "registers" panel displays the following values:

Register	H	L
AX	00	03
BX	A8	93
CX	00	00
DX	00	00
CS	0100	
IP	0008	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

The main window displays assembly code at memory address 0100:0008. The code is as follows:

```

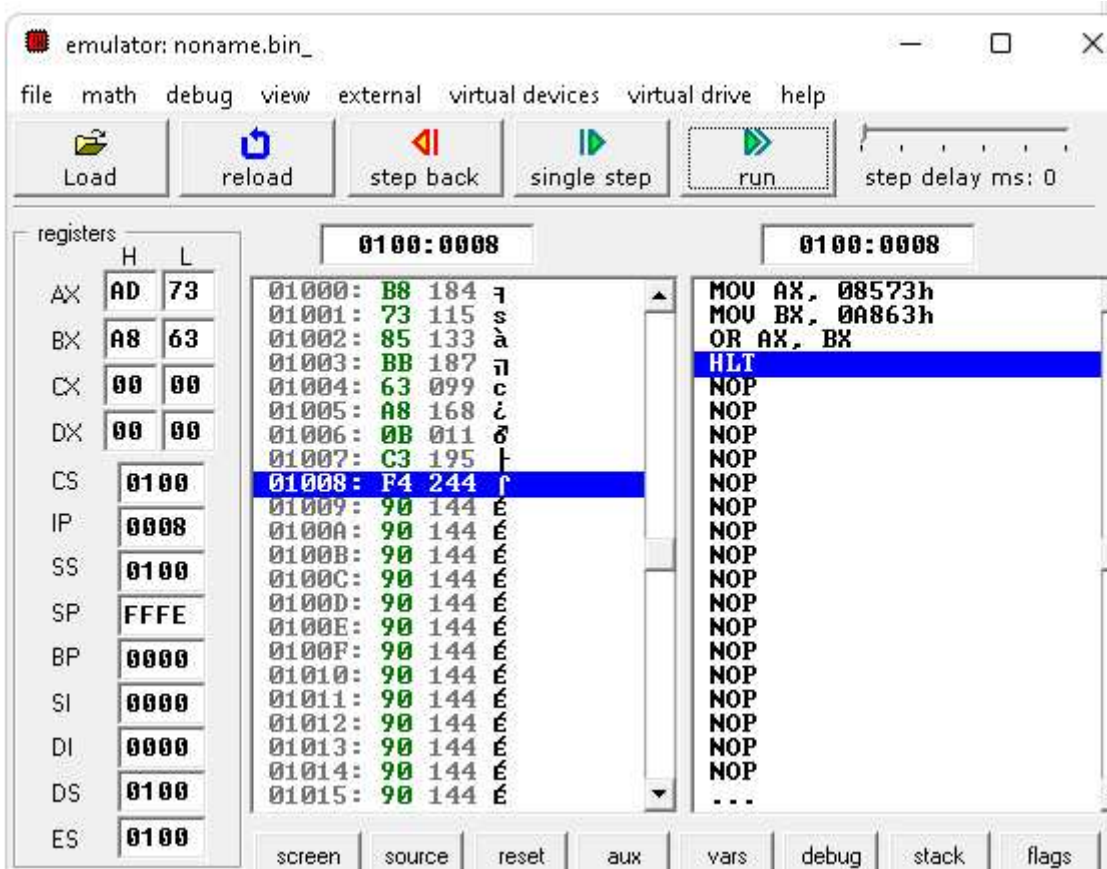
01000: B8 184 7 MOV AX, 01567h
01001: 67 103 g MOV BX, 0A893h
01002: 15 021 S AND AX, BX
01003: BB 187 u HLT
01004: 93 147 o NOP
01005: A8 168 z NOP
01006: 23 035 # NOP
01007: C3 195 t NOP
01008: F4 244 j NOP
01009: 90 144 E NOP
0100A: 90 144 E NOP
0100B: 90 144 E NOP
0100C: 90 144 E NOP
0100D: 90 144 E NOP
0100E: 90 144 E NOP
0100F: 90 144 E NOP
01010: 90 144 E NOP
01011: 90 144 E NOP
01012: 90 144 E NOP
01013: 90 144 E NOP
01014: 90 144 E NOP
01015: 90 144 E ...
    
```

At the bottom of the window, there are tabs for "screen", "source", "reset", "aux", "vars", "debug", "stack", and "flags".

iii) OR operation:

```
MOV AX, 1567H;
```

```
MOV BX, 0A893H;
OR AX, BX;
HLT;
```



emulator: noname.bin_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L
AX	AD	73
BX	A8	63
CX	00	00
DX	00	00
CS	0100	
IP	0008	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

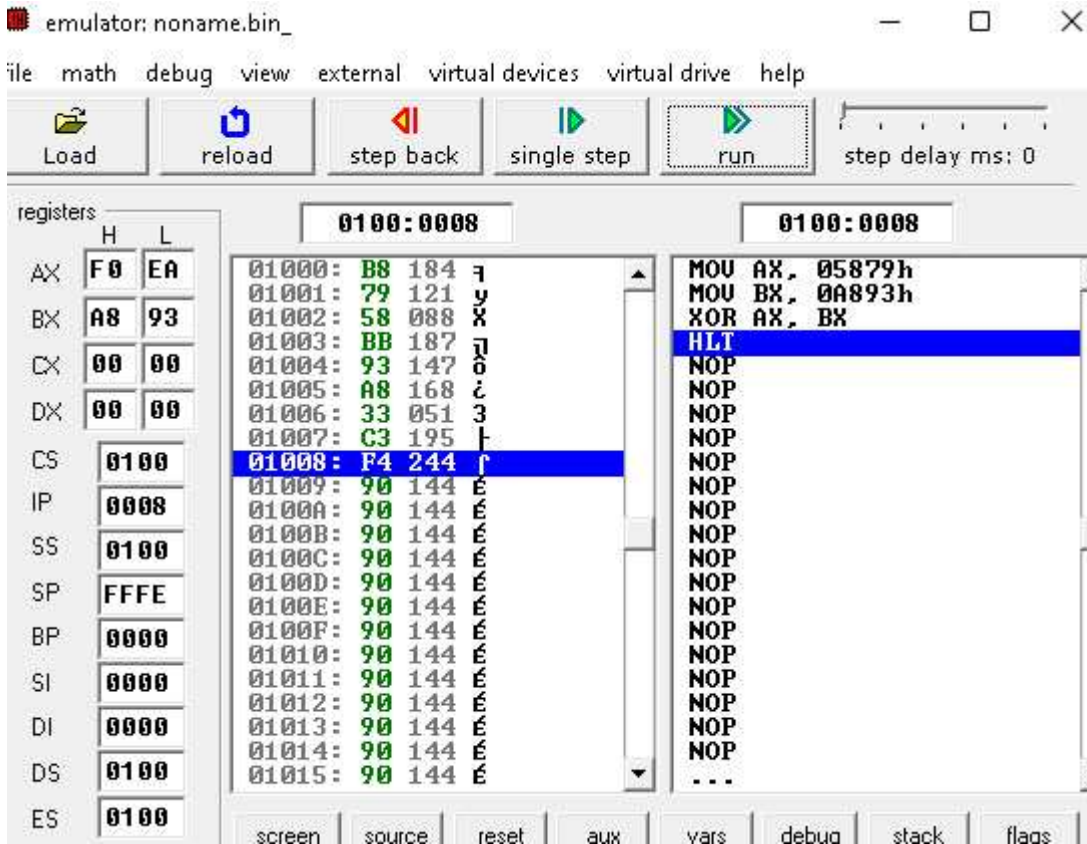
Address	Hex	Dec	Char
01000:	B8	184	ř
01001:	73	115	s
01002:	85	133	à
01003:	BB	187	ï
01004:	63	099	c
01005:	A8	168	è
01006:	0B	011	ó
01007:	C3	195	ı
01008:	F4	244	ŕ
01009:	90	144	É
0100A:	90	144	É
0100B:	90	144	É
0100C:	90	144	É
0100D:	90	144	É
0100E:	90	144	É
0100F:	90	144	É
01010:	90	144	É
01011:	90	144	É
01012:	90	144	É
01013:	90	144	É
01014:	90	144	É
01015:	90	144	É

```
MOV AX, 08573h
MOV BX, 0A863h
OR AX, BX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen source reset aux vars debug stack flags

iv) EX-OR operation:
 MOV AX, 1567H;

```
MOV BX, 0A893H;
EXOR AX, BX;
HLT;
```



Result: We have performed various logical operations in 8086.

Signature of Faculty

Experiment - 5

Perform Universal Gate Operation using 8086

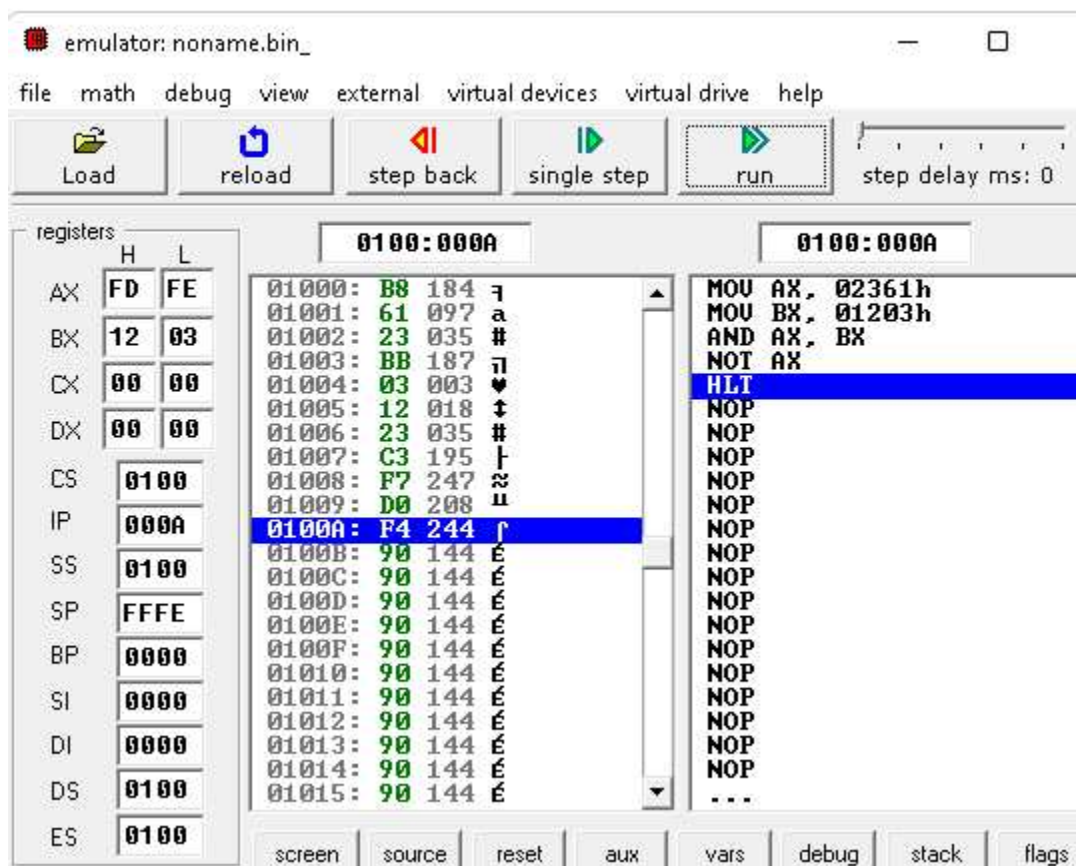
Aim: Write a program in 8086 to perform universal gate operation.

Software Used: EMU8086

Programs:

i) NAND operation:

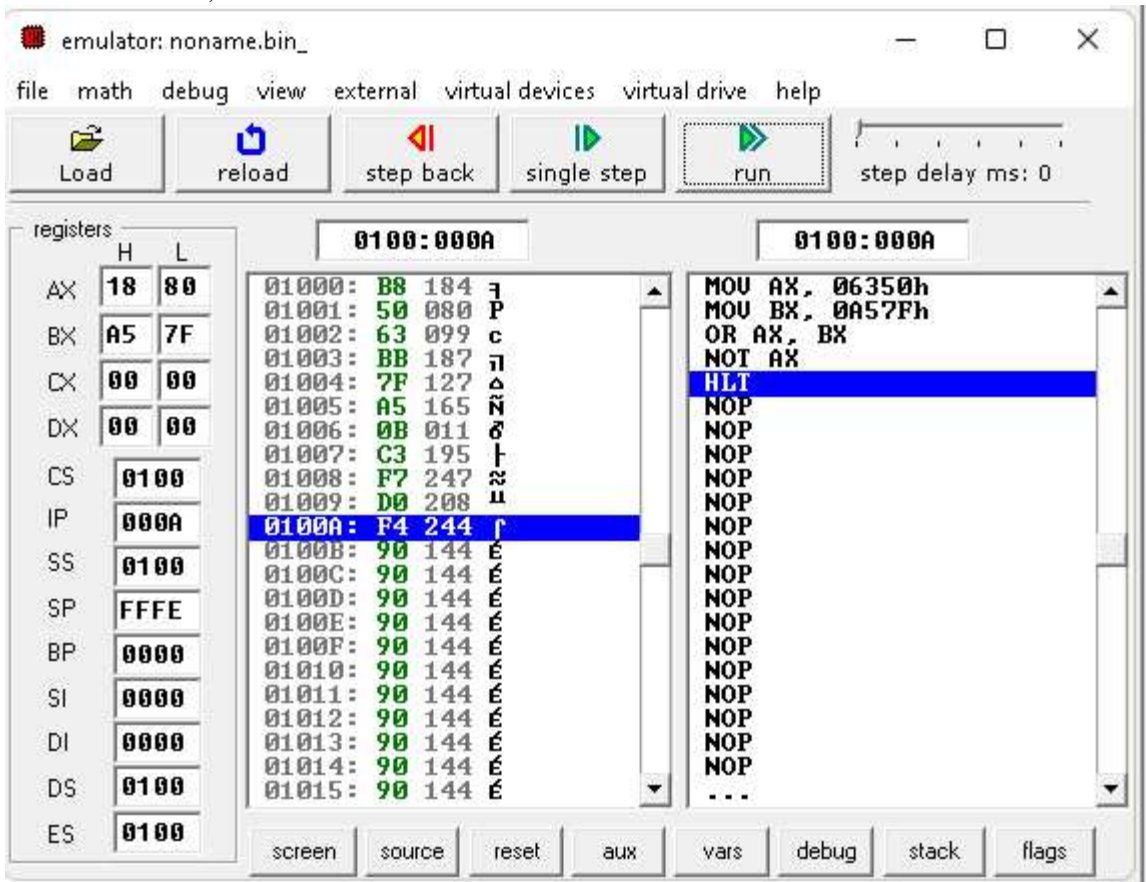
```
MOV AX, 1567H;
MOV BX, 0A893H;
AND AX, BX;
NOT AX;
HLT;
```



ii) NOR operation:

```
MOV AX, 1567H;
MOV BX, 0A893H;
```

OR AX, BX;
 NOT AX;
 HLT;



Result: Universal Gates has been performed in 8086.

Signature of Faculty

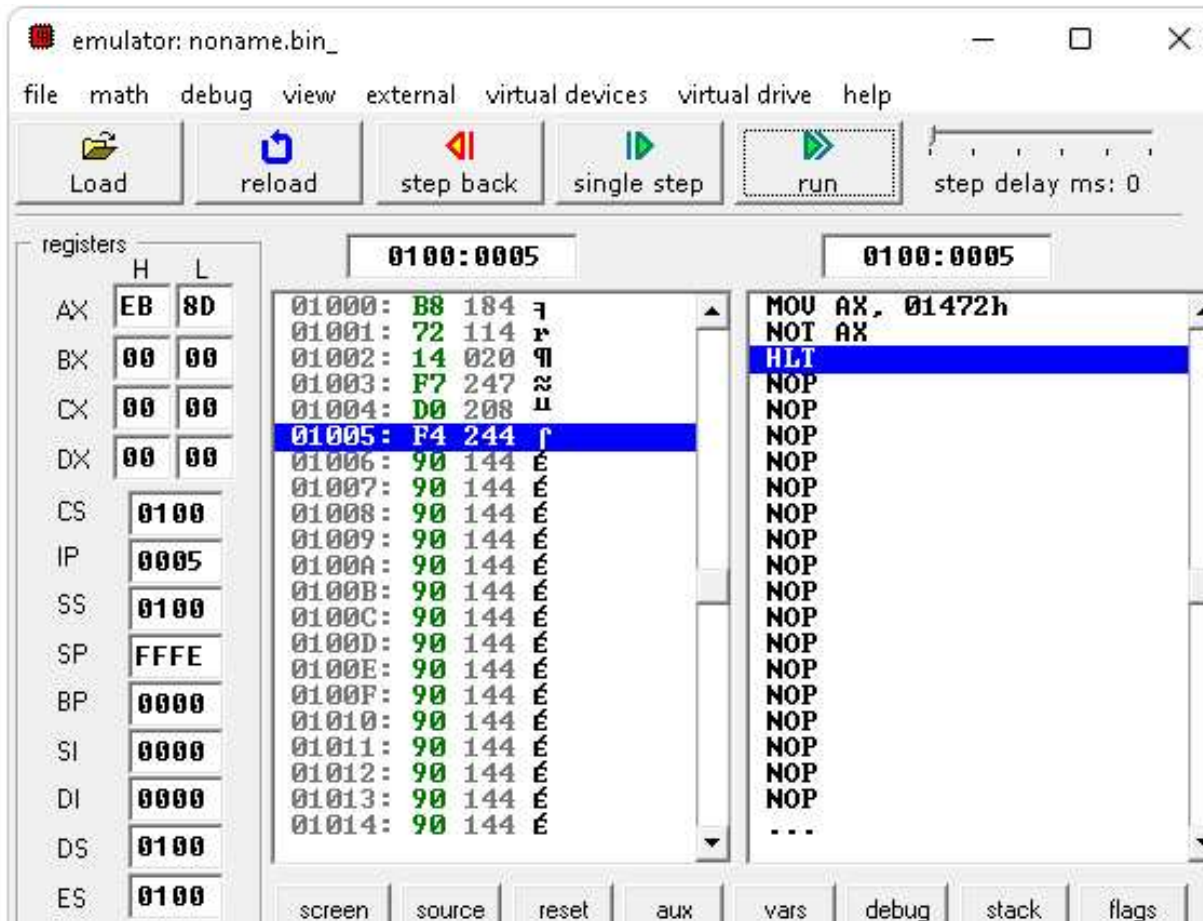
Experiment - 6
Find 1's and 2's complements

Aim: Write a program to find one's and two's complement of a number in 8086.

Program: -

1. One's complement:

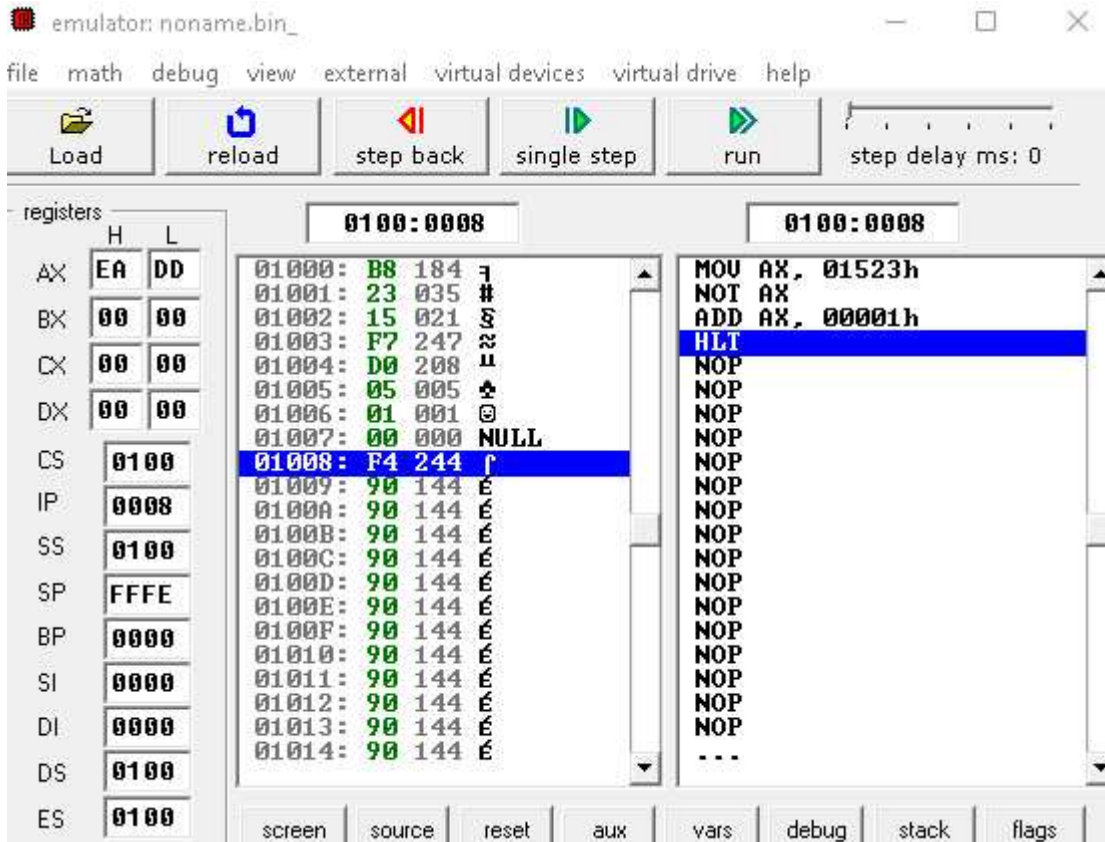
```
MOV AX, 1472H;
NOT AX;
HLT;
```



2. Two's complement:

```
a) MOV AX,1523H;
NOT AX;
```

ADD AX,0001H;
 HLT;



The screenshot shows an emulator window titled "emulator: noname.bin_". The menu bar includes "file", "math", "debug", "view", "external", "virtual devices", "virtual drive", and "help". The toolbar contains buttons for "Load", "reload", "step back", "single step", "run", and a "step delay ms: 0" slider.

The "registers" panel on the left shows the following values:

Register	H	L
AX	EA	DD
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	0008	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

The main assembly window shows memory addresses from 01000 to 01014. The instruction at 01008 is highlighted in blue:

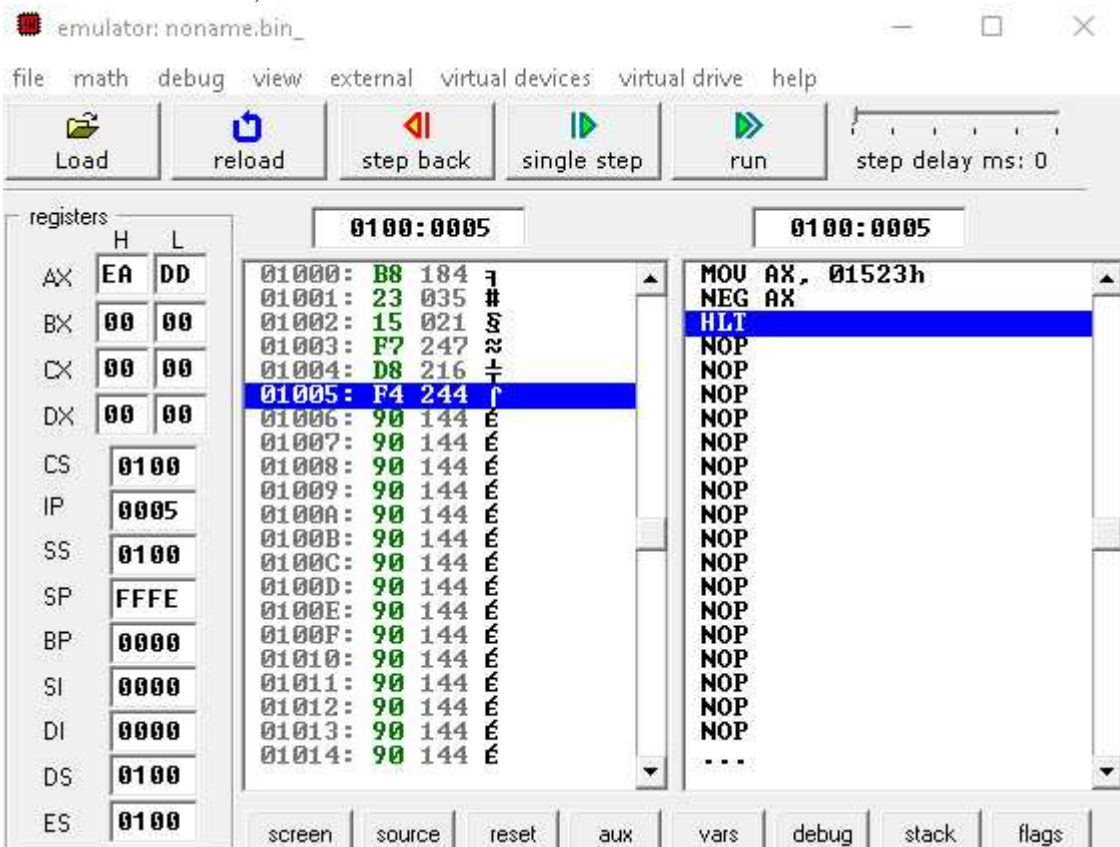
```

01000: B8 184  MOV AX, 01523h
01001: 23 035  NOT AX
01002: 15 021  ADD AX, 00001h
01003: F7 247  HLT
01004: D0 208  NOP
01005: 05 005  NOP
01006: 01 001  NOP
01007: 00 000  NOP
01008: F4 244  HLT
01009: 90 144  NOP
0100A: 90 144  NOP
0100B: 90 144  NOP
0100C: 90 144  NOP
0100D: 90 144  NOP
0100E: 90 144  NOP
0100F: 90 144  NOP
01010: 90 144  NOP
01011: 90 144  NOP
01012: 90 144  NOP
01013: 90 144  NOP
01014: 90 144  NOP
    
```

At the bottom, there are tabs for "screen", "source", "reset", "aux", "vars", "debug", "stack", and "flags".

- b) MOV AX,1523H;
 NEG AX;

HLT;



emulator: noname.bin_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers		0100:0005		0100:0005	
	H	L			
AX	EA	DD	01000: B8 184 3	MOU AX, 01523h	
BX	00	00	01001: 23 035 #	NEG AX	
CX	00	00	01002: 15 021 S	HLT	
DX	00	00	01003: F7 247 S	NOP	
CS	0100		01004: D8 216 T	NOP	
IP	0005		01005: F4 244 J	NOP	
SS	0100		01006: 90 144 E	NOP	
SP	FFFE		01007: 90 144 E	NOP	
BP	0000		01008: 90 144 E	NOP	
SI	0000		01009: 90 144 E	NOP	
DI	0000		0100A: 90 144 E	NOP	
DS	0100		0100B: 90 144 E	NOP	
ES	0100		0100C: 90 144 E	NOP	
			0100D: 90 144 E	NOP	
			0100E: 90 144 E	NOP	
			0100F: 90 144 E	NOP	
			01010: 90 144 E	NOP	
			01011: 90 144 E	NOP	
			01012: 90 144 E	NOP	
			01013: 90 144 E	NOP	
			01014: 90 144 E	NOP	
				...	

screen source reset aux vars debug stack flags

Result: one's and two's complement of a number in 8086 has been evaluated.

Signature of Faculty

Experiment – 7

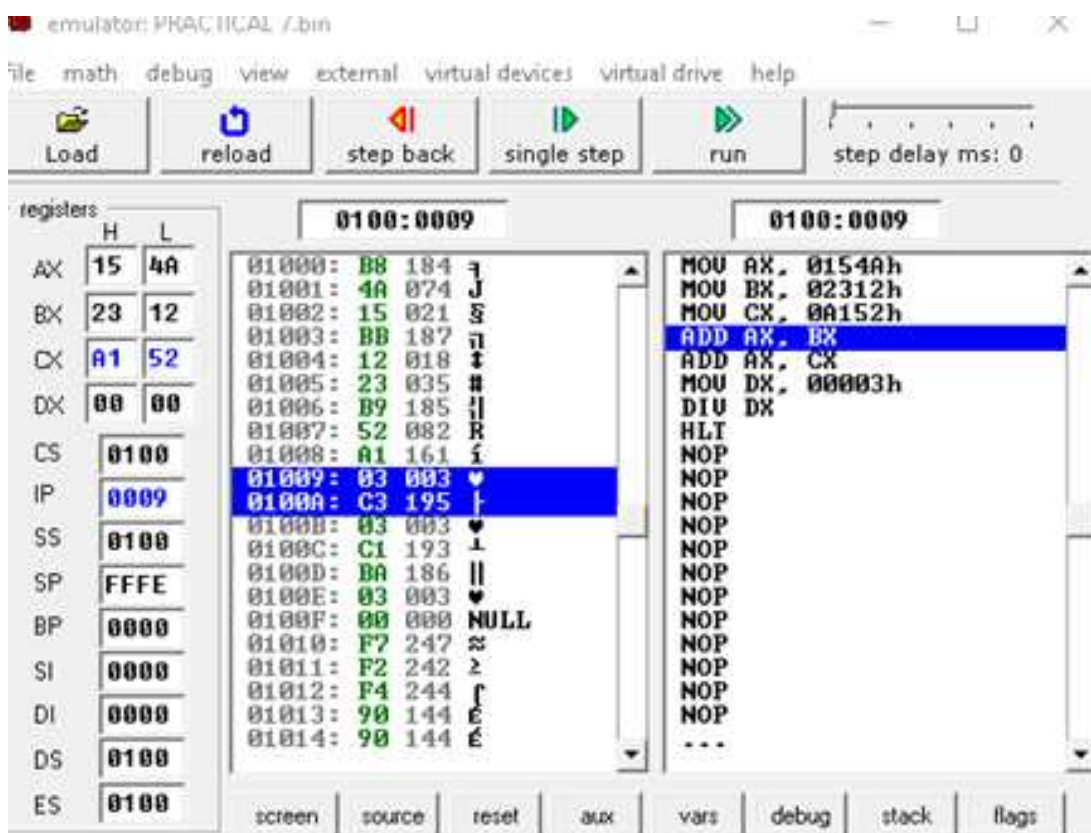
Department of Electronics & Communication Engineering

Find the average of numbers

Aim: Write a program to find average of 3 numbers in 8086.

Program: -

```
MOV AX, 154AH;
MOV BX, 2312H;
MOV CX, 0A152H;
ADD AX, BX;
ADD AX, CX;
MOV DX, 0003H;
DIV DX;
HLT;
```



emulator: PRACTICAL /bin

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers		0100:0009		0100:0009	
	H	L			
AX	15	4A	01000: B8 184	MOV AX, 0154Ah	
BX	23	12	01001: 4A 074	MOV BX, 02312h	
CX	A1	52	01002: 15 021	MOV CX, 0A152h	
DX	00	00	01003: BB 187	ADD AX, BX	
CS	0100		01004: 12 018	ADD AX, CX	
IP	0009		01005: 23 035	MOV DX, 00003h	
SS	0100		01006: B9 185	DIV DX	
SP	FFFE		01007: 52 082	HLT	
BP	0000		01008: A1 161	NOP	
SI	0000		01009: 03 003	NOP	
DI	0000		0100A: C3 195	NOP	
DS	0100		0100B: 03 003	NOP	
ES	0100		0100C: C1 193	NOP	
			0100D: BA 186	NOP	
			0100E: 03 003	NOP	
			0100F: 00 000	NOP	
			01010: F7 247	NOP	
			01011: F2 242	NOP	
			01012: F4 244	NOP	
			01013: 90 144	NOP	
			01014: 90 144	NOP	

screen source reset aux vars debug stack flags



emulator: PRACTICAL 7.bin

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers		0100:0012	0100:0012
	H L		
AX	D9 AE	01012: F4 244 ↑	HLT
BX	23 12	01013: 90 144 é	NOP
CX	A1 52	01014: 90 144 é	NOP
DX	00 03	01015: 90 144 é	NOP
CS	0100	01016: 90 144 é	NOP
IP	0012	01017: 90 144 é	NOP
SS	0100	01018: 90 144 é	NOP
SP	FFFE	01019: 90 144 é	NOP
BP	0000	0101A: 90 144 é	NOP
SI	0000	0101B: 90 144 é	NOP
DI	0000	0101C: 90 144 é	NOP
DS	0100	0101D: 90 144 é	NOP
ES	0100	0101E: 90 144 é	NOP
		01020: 90 144 é	NOP
		01021: 90 144 é	NOP
		01022: 90 144 é	NOP
		01023: 90 144 é	NOP
		01024: 90 144 é	NOP
		01025: 90 144 é	NOP
		01026: 90 144 é	NOP

screen source reset aux vars debug stack flags

Result: Average of three numbers has been evaluated.

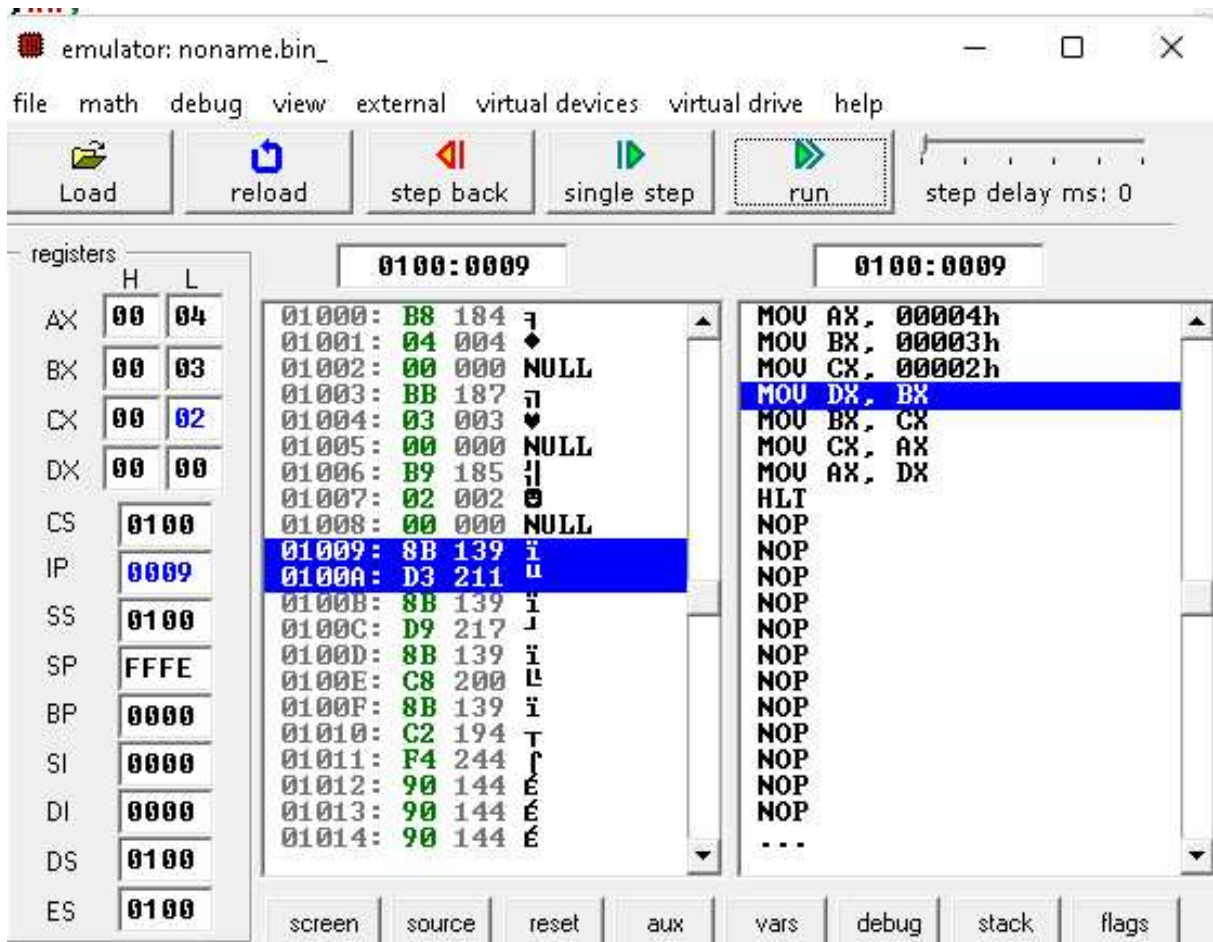
Signature of Faculty

Experiment – 8 Exchanging the data of registers

Aim: Write a program in 8086 to exchange the data of the registers.

Program: -

```
MOV AX, 0004H;
MOV BX, 0003H;
MOV DX, BX;
MOV BX, CX;
MOV CX, AX;
MOV AX, DX;
HLT;
```



emulator: noname.bin_

file math debug view external virtual devices virtual drive help

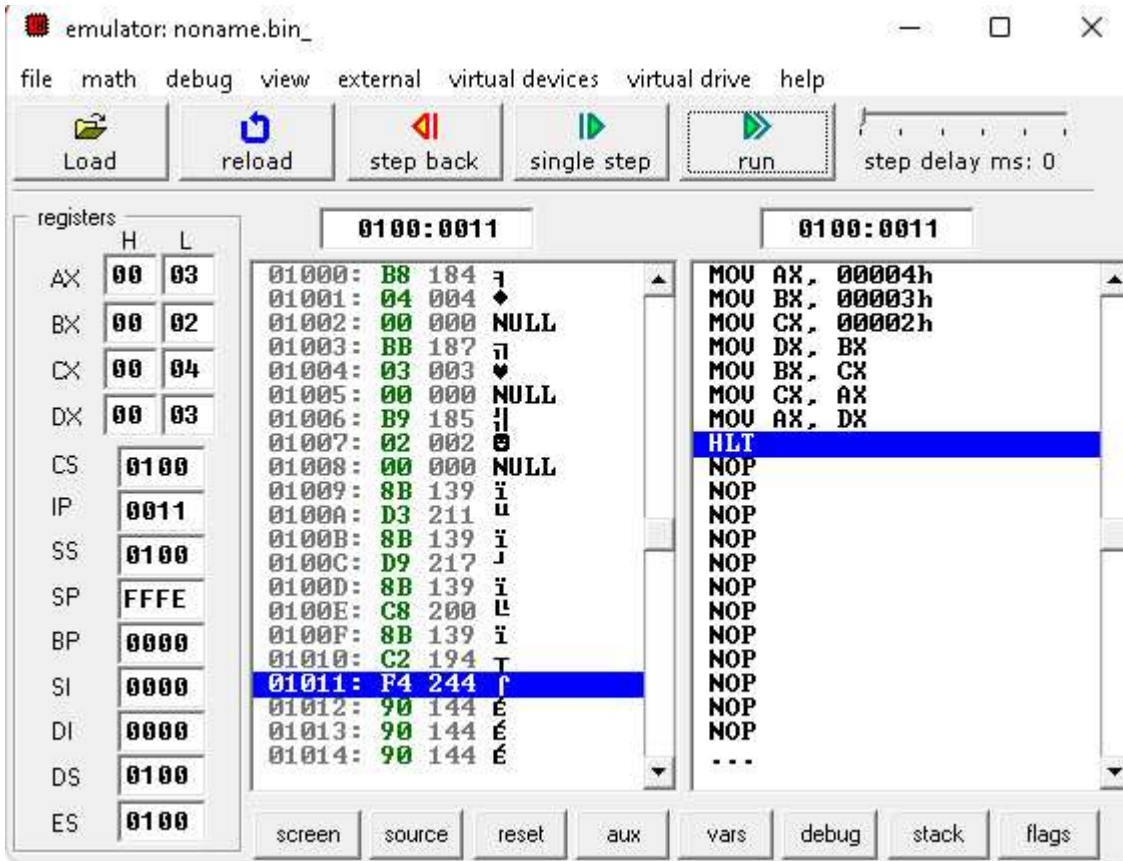
Load reload step back single step run step delay ms: 0

registers	H	L
AX	00	04
BX	00	03
CX	00	02
DX	00	00
CS	0100	
IP	0009	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

Address	Hex	Dec	Symbol
01000:	B8	184	↵
01001:	04	004	↴
01002:	00	000	NULL
01003:	BB	187	↵
01004:	03	003	↴
01005:	00	000	NULL
01006:	B9	185	↵
01007:	02	002	↴
01008:	00	000	NULL
01009:	8B	139	↵
0100A:	D3	211	↵
0100B:	8B	139	↵
0100C:	D9	217	↵
0100D:	8B	139	↵
0100E:	C8	200	↵
0100F:	8B	139	↵
01010:	C2	194	↵
01011:	F4	244	↵
01012:	90	144	↵
01013:	90	144	↵
01014:	90	144	↵

```
MOV AX, 00004h
MOV BX, 00003h
MOV CX, 00002h
MOV DX, BX
MOV BX, CX
MOV CX, AX
MOV AX, DX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen source reset aux vars debug stack flags



Result: Ex-changing the data of the registers has been performed.

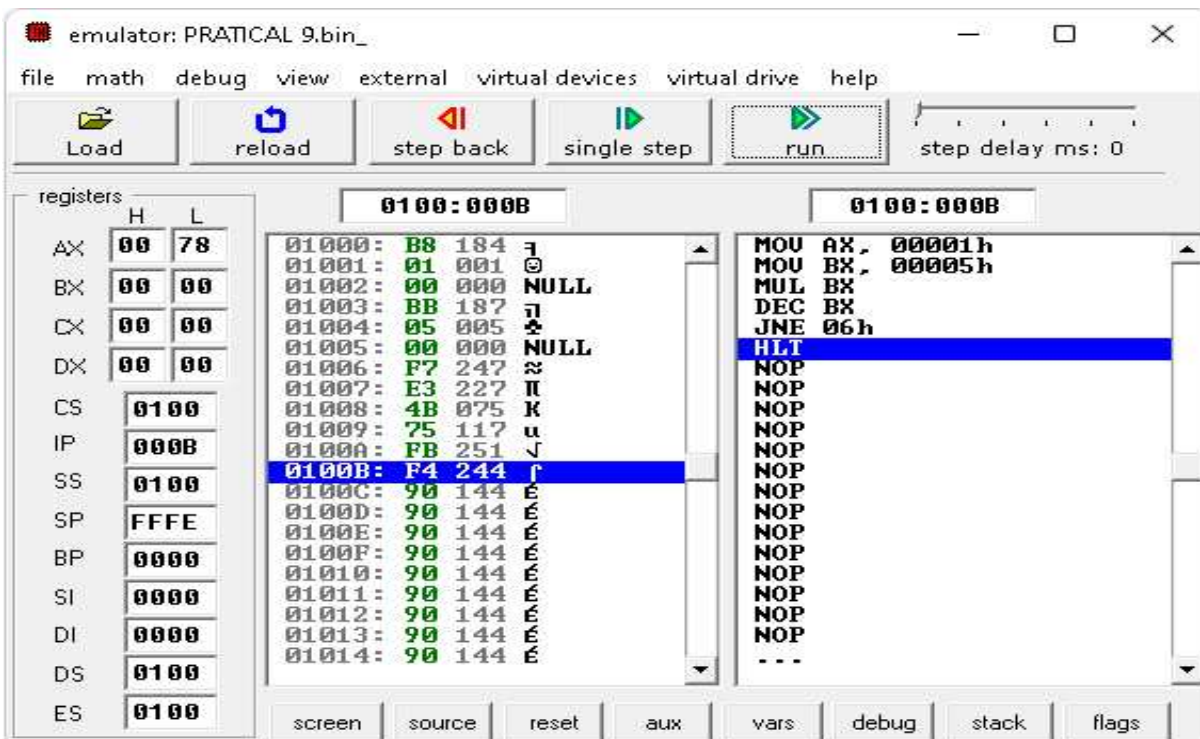
Signature of Faculty

Experiment – 9 Finding factorial of a number

Aim: Write a program to find factorial of a number in 8086.

Program: -

```
MOV AX,0001H;
MOV BX,0005H;
MUL BX;
DEC BX;
JNZ L1;
HLT;
```



Result: Factorial of a number in 8086 has been evaluated.

Signature of Faculty

Experiment – 10 Stepper Motor Rotation

Aim: Write a program to rotate stepper motor using 8051

Program: -

```

0000| MOV TMOD, #50H           ; put timer 1 in event
counting mode
0003|     SETB TR1             ; start timer 1

0005|     MOV DPL, #LOW(LEDcodes) ; | put the low byte of the
start address of the                               ; | 7-segment code table
into DPL

0008|     MOV DPH, #HIGH(LEDcodes) ; put the high byte into DPH

000B|     CLR P3.4             ; |
000D|     CLR P3.3             ; | enable Display 0
again:
000F|     CALL setDirection     ; set the motor's direction
0011|     MOV A, TL1            ; move timer 1 low byte
to A
0013|     CJNE A, #10, skip     ; if the number of
revolutions is not 10 skip next instruction
0016|     CALL clearTimer       ; if the number of
revolutions is 10, reset timer 1
skip:
0018|     MOVC A, @A+DPTR       ; | get 7-segment code
from code table - the index into the table is    ; |
                                                    ; | decided by the value
in A                                              ; |
                                                    ; | (example: the data
pointer points to the start of the                ; |
                                                    ; | table - if there are
two revolutions, then A will contain two,         ; |
                                                    ; | therefore the second
code in the table will be copied to A)

0019|     MOV C, F0             ; move motor direction value
to the carry

```



```

001B|      MOV ACC.7, C          ; | and from there to ACC.7
      (this will ensure Display 0's decimal point
                                   ; | will indicate the
motor's direction)

001D|      MOV P1, A           ; | move (7-seg code for)
      number of revolutions and motor direction
                                   ; | indicator to Display
0

001F|      JMP again          ; do it all again

      setDirection:
0021|      PUSH ACC           ; save value of A on stack

0023|      PUSH 20H          ; | save value of location
20H (first bit-addressable
                                   ; | location in RAM) on
stack

0025|      CLR A              ; clear A
0026|      MOV 20H, #0        ; clear location 20H
0029|      MOV C, P2.0        ; put SW0 value in carry
002B|      MOV ACC.0, C      ; then move to ACC.0
002D|      MOV C, F0         ; move current motor
      direction in carry
002F|      MOV 0, C          ; and move to LSB of location
20H (which has bit address 0)

0031|      CJNE A, 20H, changeDir ; | compare SW0 (LSB of A)
with F0 (LSB of 20H)
                                   ; | - if they are not the
same, the motor's direction needs to be reversed

0034|      JMP finish        ; if they are the same,
motor's direction does not need to be changed
      changeDir:
0036|      CLR P3.0          ; |
0038|      CLR P3.1          ; | stop motor

003A|      CALL clearTimer   ; reset timer 1
(revolution count restarts when motor direction changes)
003C|      MOV C, P2.0        ; move SW0 value to carry

```

```

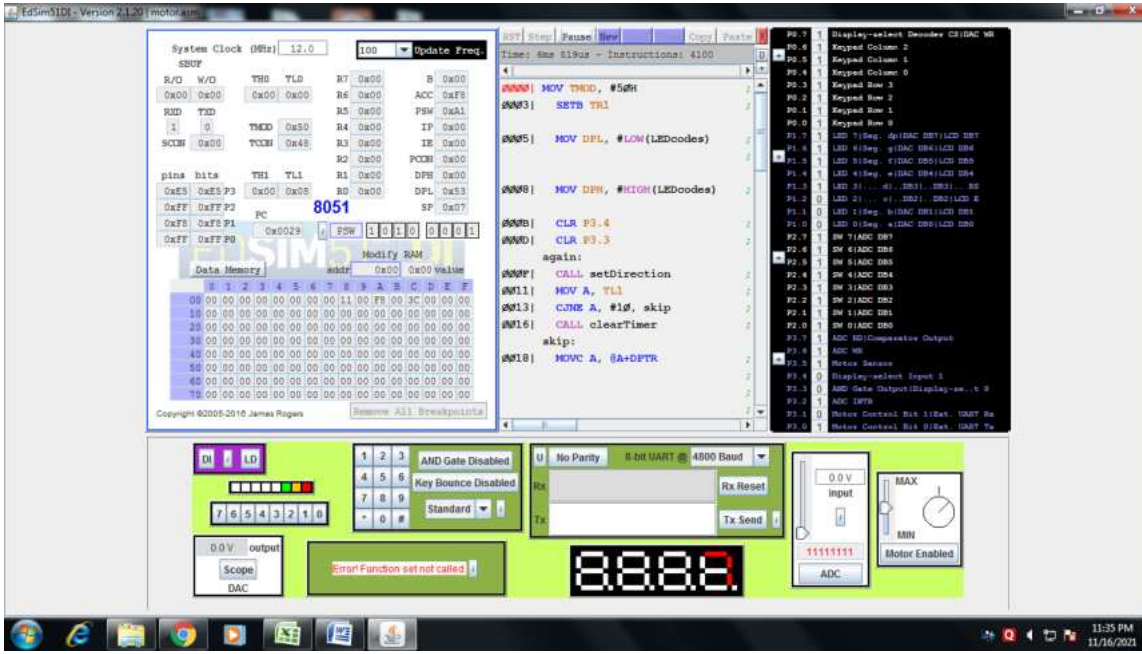
003E|    MOV F0, C                ; and then to F0 - this is
the new motor direction
0040|    MOV P3.0, C            ; move SW0 value (in
carry) to motor control bit 1
0042|    CPL C                  ; invert the carry

0043|    MOV P3.1, C            ; | and move it to motor
control bit 0 (it will therefore have the opposite
; | value to control bit
1 and the motor will start
; | again in the new
direction)
    finish:
0045|    POP 20H                ; get original value for
location 20H from the stack
0047|    POP ACC               ; get original value for
A from the stack
0049|    RET                   ; return from subroutine

    clearTimer:
004A|    CLR A                  ; reset revolution count
in A to zero
004B|    CLR TR1               ; stop timer 1
004D|    MOV TL1, #0          ; reset timer 1 low byte
to zero
0050|    SETB TR1              ; start timer 1
0052|    RET                   ; return from subroutine

    LEDcodes:                  ; | this label points to the start address
of the 7-segment code table which is
; | stored in program memory using the DB
command below
                                DB 11000000B, 11111001B, 10100100B,
10110000B, 10011001B, 10010010B, 10000010B, 11111000B, 10000000B,
10010000B

```



Result: Stepper motor rotation has been performed using 8051 programming.

Signature of Faculty