

Experiment 05

Experiment 5: Write an ALP program

1. Program to ADD,SUB 8bit and 16 bit number
2. Program for 8 bit and 16 bit BCD addition

--

Roll No.	17
Name	Manav Jawrani
Class	D10-A
Subject	Microprocessor Lab
LO Mapped	LO3: Build a program on a microprocessor using arithmetic & logical instruction set of 8086.

Aim: Write an ALP program

1. Program to ADD,SUB 8bit and 16 bit number
2. Program for 8 bit and 16 bit BCD addition

Introduction:

An assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware. Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans. Low-level programming languages such as assembly language are a necessary bridge between the underlying hardware of a computer and the higher-level programming languages—such as Python or JavaScript—in which modern software programs are written.

Theory:

The 8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and 16 data lines that provides up to 1MB storage. It consists of a powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for a system having multiple processors and Minimum mode is suitable for a system having a single processor.

8 bits can represent positive numbers from 0 to 255. While 16 bit: Signed Integers ranging from -32768 to +32767. The 16-bit data type is used for numerical tags where variables have the potential for negative or positive values.

A.**1. Addition of two 8 bit numbers.**

Algorithm:

Step 1 - Initialize the data segment with input numbers and memory location for the answer.

Step 2 - Start the program by loading the first data into the Accumulator.

Step 3 - Move the first number to register 'al' and move the second number to register 'bl'.

Step 4 - Add the two register contents.

Step 5 - Store the content of al to ans.

Step 6 - Terminate the program.

Code:

```
.model small
```

```
.data
```

```
n1 db 10h
```

```
n2 db 22h
```

```
sum db ?
```

```
carry db 00h
```

```
.code
```

```
start: mov ax, @data
```

```
        mov ds,ax
```

```
mov al, n1
```

```
add al, n2
```

```
jnc skip
```

```
mov carry, 01h
```

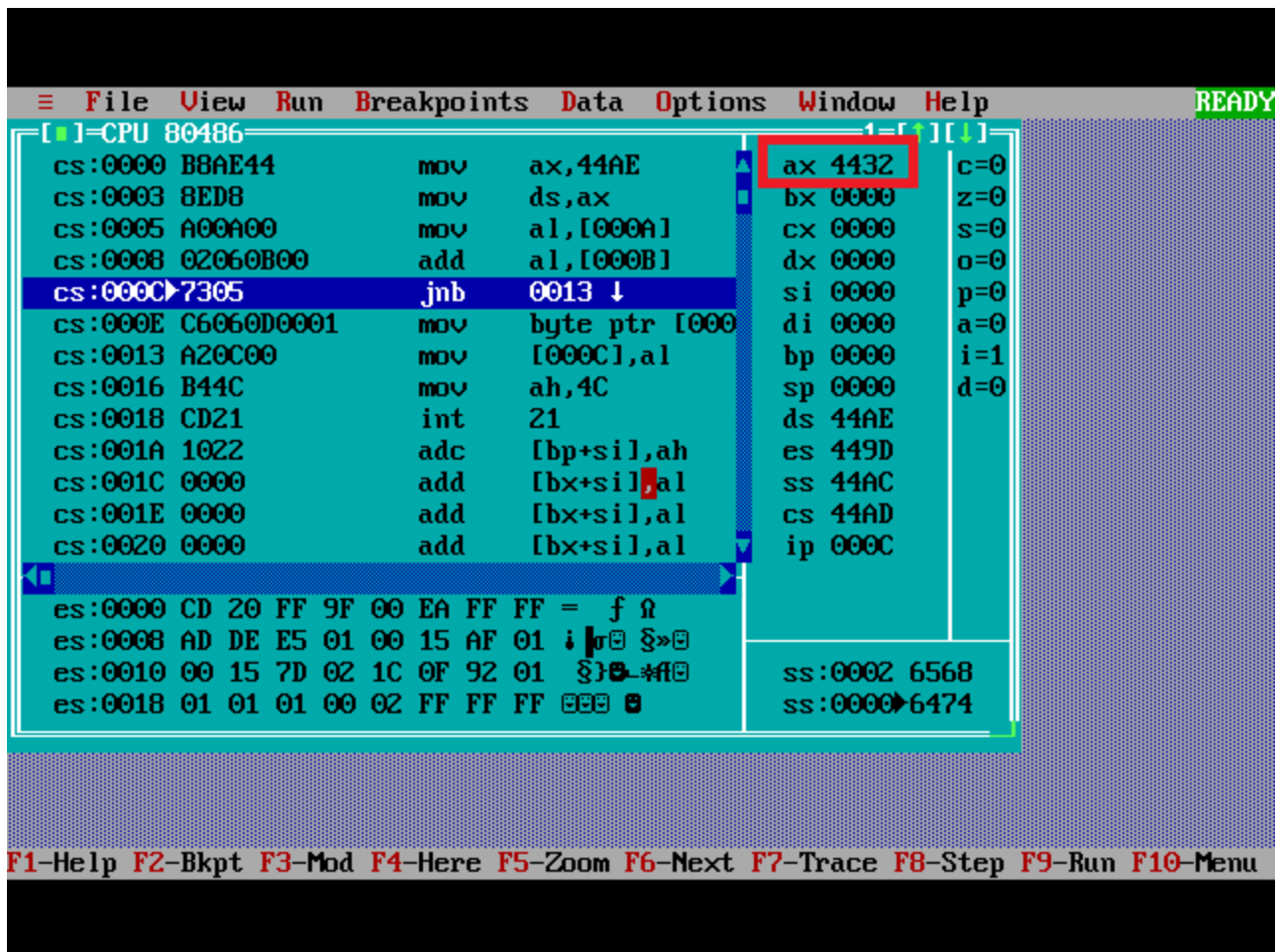
```
skip: mov sum, al
```

```

mov ah,4ch
int 21h
end start
end code
end

```

Output:



2. Subtraction of two 8 bit numbers.

Algorithm:

Step 1 - Initialize the data segment with input numbers and memory location for the answer.

Step 2 - Start the program by loading the first data into the Accumulator.

Step 3 - Move the first number to register 'al' and move the second number to register 'bl'.

Step 4 - Subtract the two register contents.

Step 5 - Store the content of al to ans

Step 6 - Terminate the program.

Code:

```
data segment
```

```
n1 db 05h
```

```
n2 db 03h
```

```
ans db ?
```

```
data ends
```

```
code segment
```

```
start : assume cs :code, ds :data
```

```
mov ax,data
```

```
mov ds,ax
```

```
mov al,n1
```

```
mov bl,n2
```

```
SUB al,bl
```

```
mov cl,al
```

```
mov ah, 4ch
```

```
int 21h
```

code ends
end start

Output:

The screenshot displays a DOS-based 80486 debugger interface. The main window shows assembly code for the CS segment, with the instruction at address 0010 (mov ah, 4C) highlighted. The register window on the right shows the value of the AX register as 4402, which is highlighted with a red box. The status bar at the bottom indicates the current instruction is at address 0010.

Address	Instruction	Register/Value
cs:0000	mov ax, 44AD	ax 4402
cs:0003	mov ds, ax	bx 0003
cs:0005	mov al, [0000]	cx 0002
cs:0008	mov bl, [0001]	dx 0000
cs:000C	sub al, bl	si 0000
cs:000E	mov cl, al	di 0000
cs:0010	mov ah, 4C	bp 0000
cs:0012	int 21	sp 0000
cs:0014	add [bx+si], al	ds 44AD
cs:0016	add [bx+si], al	es 449D
cs:0018	add [bx+si], al	ss 44AC
cs:001A	add [bx+si], al	cs 44AE
cs:001C	add [bx+si], al	ip 0010

es:0000 CD 20 FF 9F 00 EA FF FF = f Ω
 es:0008 AD DE E5 01 00 15 AF 01 : | 0 0 0 0 0 0 0 0
 es:0010 00 15 7D 02 1C 0F 92 01 0 0 0 0 0 0 0 0
 es:0018 01 01 01 00 02 FF FF FF 0 0 0 0 0 0 0 0

ss:0002 F5D0
 ss:0000 5ED8

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

3. Addition of two 16 bit numbers.

Algorithm:

Step 1 - Initialize the data segment with input numbers and memory location for the answer.

Step 2 - Start the program by loading the first data into the Accumulator.

Step 3 - Move the first number to register 'ax' and move the second number to register 'bx'.

Step 4 - Add the two register contents.

Step 5 - Store the content of 'ax' to 'c'.

Step 6 - Terminate the program.

Code:

```
data segment
a dw 1234h
b dw 5678h
c dw ?
data ends
code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov ax,a
mov bx,b
add ax,bx
mov cx,ax
int 3
code ends
```

end

Output:

The screenshot shows a DOS-based 80486 emulator interface. The menu bar at the top includes File, View, Run, Breakpoints, Data, Options, Window, and Help. The status bar at the bottom shows function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu. The main window is divided into several panes. The top-left pane displays assembly code with the following instructions: cs:001C 03C3 add ax,bx; cs:001E 8BC8 mov cx,ax; cs:0020 CC int 03; cs:0021 4C dec sp; cs:0022 CD21 int 21; cs:0024 0000 add [bx+si],al; cs:0026 0000 add [bx+si],al; cs:0028 0000 add [bx+si],al; cs:002A 0000 add [bx+si],al; cs:002C 0000 add [bx+si],al; cs:002E 0000 add [bx+si],al; cs:0030 0000 add [bx+si],al; cs:0032 0000 add [bx+si],al. The top-right pane shows register values: ax 68AC, bx 5678, cx 0002, dx 0000, si 0000, di 0000, bp 0000, sp 0000, ds 44AD, es 449D, ss 44AC, cs 44AD, and ip 001E. The bottom-left pane shows memory dump for es:0000 to es:0018. The bottom-right pane shows stack pointer values: ss:0002 F5D0 and ss:0000 5ED8. The status bar at the bottom right indicates 'READY'.

```
File View Run Breakpoints Data Options Window Help
[ ]=CPU 80486
cs:001C 03C3 add ax,bx
cs:001E 8BC8 mov cx,ax
cs:0020 CC int 03
cs:0021 4C dec sp
cs:0022 CD21 int 21
cs:0024 0000 add [bx+si],al
cs:0026 0000 add [bx+si],al
cs:0028 0000 add [bx+si],al
cs:002A 0000 add [bx+si],al
cs:002C 0000 add [bx+si],al
cs:002E 0000 add [bx+si],al
cs:0030 0000 add [bx+si],al
cs:0032 0000 add [bx+si],al
es:0000 CD 20 FF FF 00 EA FF FF = Ω
es:0008 AD DE E5 01 00 15 AF 01 ↓ 0 0 0 0 0 0 0 0
es:0010 00 15 7D 02 1C 0F 92 01 0 0 0 0 0 0 0 0
es:0018 01 01 01 00 02 FF FF FF 0 0 0 0 0 0 0 0
ax 68AC
bx 5678
cx 0002
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 44AD
es 449D
ss 44AC
cs 44AD
ip 001E
ss:0002 F5D0
ss:0000 5ED8
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```


4. Subtraction of two 16 bit numbers.

Algorithm:

Step 1 - Initialize the data segment with input numbers and memory location for the answer.

Step 2 - Start the program by loading the first data into the Accumulator.

Step 3 - Move the first number to register 'ax' and move the second number to register 'bx'.

Step 4 - Subtract the two register contents.

Step 5 - Store the content of 'ax' to 'c'.

Step 6 - Terminate the program.

Code:

```
data segment
```

```
a dw 5678h
```

```
b dw 1234h
```

```
c dw ?
```

```
data ends
```

```
code segment
```

```
assume cs:code,ds:data
```

```
start:
```

```
mov ax,data
```

```
mov ds,ax
```

```
mov ax,a
```

```
mov bx,b
```

```
sub ax,bx
```

```
mov cx,ax
```

```
int 3
```

```
code ends
```

end

Output:

The screenshot displays a DOS-based 80486 debugger interface. The main window shows assembly code with the instruction at address 0020 highlighted. The register window on the right shows the current values of the 80486 registers, with CX highlighted. The status bar at the bottom provides function key shortcuts.

Address	Disassembly	Register Values
cs:001C	sub ax,bx	ax 4444
cs:001E	mov cx,ax	bx 1234
cs:0020	int 03	cx 4444
cs:0021	dec sp	dx 0000
cs:0022	int 21	si 0000
cs:0024	add [bx+si],al	di 0000
cs:0026	add [bx+si],al	bp 0000
cs:0028	add [bx+si],al	sp 0000
cs:002A	add [bx+si],al	ds 44AD
cs:002C	add [bx+si],al	es 449D
cs:002E	add [bx+si],al	ss 44AC
cs:0030	add [bx+si],al	cs 44AD
cs:0032	add [bx+si],al	ip 0020

Registers: ax 4444, bx 1234, cx 4444, dx 0000, si 0000, di 0000, bp 0000, sp 0000, ds 44AD, es 449D, ss 44AC, cs 44AD, ip 0020

Stack (ss): 0002 F5D0, 0000 5ED8

Status Bar: F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

B.

BCD is a binary code of the ten decimal digits. It is not a binary equivalent.

To perform BCD addition

1. Add the BCD digits as regular binary numbers.
2. If the sum is 9 or less and no carry was generated, it is a valid BCD digit.
3. If the sum produces a carry, the sum is invalid and the number 6 (0110) must be added to the digit.
4. If the sum is greater than nine, the sum is invalid and the number 6 (0110) must be added to the digit.
5. Repeat for each of the BCD digits.

Ex.

GROW

1 = sum < 9 and carry = 0
 2 = sum < 9 and carry = 1
 3 = sum > 9 and carry = 0

(correct)

(incorrect and add 6 (0110))

(incorrect and add 6 (0110))

1 = (5)₁₀ + (2)₁₀ BCD ADDITION

0101 = 5 sum < 9
 + 0010 = 2 carry = 0
 0111

2 = (7)₁₀ + (5)₁₀ BCD ADDITION

111
 0111 = 7 sum > 9
 + 0101 = 5 carry = 0
 1100

0

1

2

2⁴ = 16

3

15 - 9 = 6

4

5

6

7

8

9

10 = A

11 = B

12 = C

13 = D

LEARN

1. BCD addition of two 8 bit numbers.

Algorithm:

Step 1 - Initialize the data section.

Step 2 - Load number1 in 'al' and load number2 in 'bl'.

Step 3 - Add numbers and store the result in 'al'.

Step 4 - Adjust result to valid BCD number. Also count the digits displayed.

Step 5 - Count to roll by 4 bits. And result in registered 'bh'.

Step 6 - Roll 'bl' so that msb comes to lsb. Now load 'dl' with data to be displayed.

Step 7 - Get only lsb.

Step 8 - Check if digit is 0-9 or letter A-F. And if a letter is there then add 37H else only add 30H.

Step 9 - Function 2 under INT 21H (Display character).

Step 10 - Decrement count and terminate the program.

Code:

```
.model small
.data
a db 08H
b db 08H
.code
    mov    ax, @data
    mov    ds, ax
    mov    al, a
    mov    bl, b
    add    al, bl
    daa
```

```
    mov    ch, 02h
    mov    cl, 04h
    mov    bh, al
12:  rol    bh, cl
    mov    dl, bh
    and    dl, 0fH
    cmp    dl, 09
    jbe    l4
    add    dl, 07
14:  add    dl, 30H
    mov    ah, 02
    int     21H
    dec    ch
    jnz    l2
    mov    ah, 4cH
    int     21H
    end
```

Output:

The screenshot displays a DOS-based 80486 emulator interface. The main window is titled "[]-CPU 80486" and contains a list of assembly instructions with their addresses and hex values. The instruction at address 0015 is highlighted in blue. To the right of the instruction list, a register window shows the current values of various registers. The 'ax' register is highlighted with a red box and shows the value 4416. Below the register window, the stack pointer (sp) is shown at 44B0. At the bottom of the window, a status bar displays various function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu. The status bar also indicates the current state as 'READY'.

```
File View Run Breakpoints Data Options Window Help READY
[ ]-CPU 80486
cs:0003 8ED8      mov     ds,ax
cs:0005 A00400     mov     al,[0004]
cs:0008 8A1E0500   mov     bl,[0005]
cs:000C 02C3      add     al,bl
cs:000E 27        daa
cs:000F B502      mov     ch,02
cs:0011 B104      mov     cl,04
cs:0013 8AF8      mov     bh,al
cs:0015 D2C7      rol     bh,cl
cs:0017 8AD7      mov     dl,bh
cs:0019 80E20F     and     dl,0F
cs:001C 80FA09     cmp     dl,09
cs:001F 7603      jbe     0024

es:0000 CD 20 FF 9F 00 EA FF FF = f Ω
es:0008 AD DE E5 01 00 15 AF 01 ; 10 5»
es:0010 00 15 7D 02 1C 0F 92 01 5»-#ff
es:0018 01 01 01 00 02 FF FF FF 000

ax 4416
bx 1608
cx 0204
dx F5C0
si F4BE
di F5C0
bp 0100
sp 0000
ds 44B0
es 449D
ss 44AC
cs 44AD
ip 0015

ss:0002 0000
ss:0000 AB90

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

2. BCD addition of two 16 bit numbers.

Algorithm:

Step 1 - Initialize data section. And load number1 in 'ax' and load number2 in 'bx'.

Step 2 - Add lower two digits. And store the result in 'al'.

Step 3 - Adjust result to valid bcd and store result in 'bl'.

Step 4 - Add upper two digits. And store the result in 'ah'.

Step 5 - 'al'='ah' as daa works on 'al' only.

Step 6 - Adjust result to valid BCD. And store the result in 'bh'.

Step 7 - Count the digits displayed.

Step 8 - Count to roll by 4 bits and roll 'bl' so that msb comes to lsb.

Step 9 - Now load 'dl' with data to be displayed and get only lsb.

Step 10 - Check if digit is 0-9 or letter A-F. If a letter is there then add 37H or else only add 30H.

Step 11 - Function 2 under INT 21H (Display character).

Step 12 - Decrement count and terminate the program.

Code:

```
.model small
.data
a dw 3629H
b dw 4738H
.code
    mov ax, @data
```

```
    mov ds, ax
    mov ax, a
    mov bx, b
    add al, bl
    daa
    mov bl, al
    adc ah, bh
    mov al, ah
    daa
    mov bh, al
    mov ch, 04h
    mov cl, 04h
12 : rol  bx, cl
    mov dl, bl
    and dl, 0fH
    cmp dl, 09
    jbe 14
    add dl, 07
14 : add  dl, 30H
    mov ah, 02
    int 21H
    dec ch
    jnz 12
    mov ah, 4cH
    int 21H
    end
```

Output:

The screenshot shows a debugger window for an 8086 microprocessor. The menu bar includes File, View, Run, Breakpoints, Data, Options, Window, and Help. The status bar at the bottom shows function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu.

The main window is divided into several sections:

- Instruction List:** A list of assembly instructions with their addresses and hex values. The instruction at address 0018 is highlighted in blue.

Address	Hex Value	Instruction
cs:0000	B8B044	mov ax,44B0
cs:0003	8ED8	mov ds,ax
cs:0005	A10A00	mov ax,[000A]
cs:0008	8B1E0C00	mov bx,[000C]
cs:000C	02C3	add al,bl
cs:000E	27	daa
cs:000F	8AD8	mov bl,al
cs:0011	12E7	adc ah,bh
cs:0013	8AC4	mov al,ah
cs:0015	27	daa
cs:0016	8AF8	mov bh,al
cs:0018	B504	mov ch,04
cs:001A	B104	mov cl,04
- Register Window:** A window on the right showing the current values of the 8086 registers. The 'bx' register value of 8367 is highlighted with a red box.

Register	Value
ax	7D83
bx	8367
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0000
ds	44B0
es	449D
ss	44AC
cs	44AD
ip	0018
- Disassembly Window:** A window at the bottom showing the disassembly of the instruction at the current address (0018).

Address	Hex Value	Disassembly
es:0000	CD 20 FF 9F 00 EA FF FF	= f 0
es:0008	AD DE E5 01 00 15 AF 01	i 0 0 0 0 0 0 0 0
es:0010	00 15 7D 02 1C 0F 92 01	0 0 0 0 0 0 0 0
es:0018	01 01 01 00 02 FF FF FF	0 0 0 0 0 0 0 0

Conclusion:

We have studied and understood 8086 Assembly Language Programming.