American International University- Bangladesh
Faculty of Engineering (EEE)
EEE 4103: Microprocessor and Embedded Systems Laboratory

Title: Familiarization of assembly language programs.

Introduction:

In this experiment, the main objective is to learn how to write assembly programs using 8086 instructions and arrays.

Theory and Methodology:

The 8086 Microprocessor

The 8086 is a 16-bit microprocessor chip designed by Intel between early 1976 and mid-1978, when it was released. The 8086 became the basic x86- architecture of Intel's future processors.

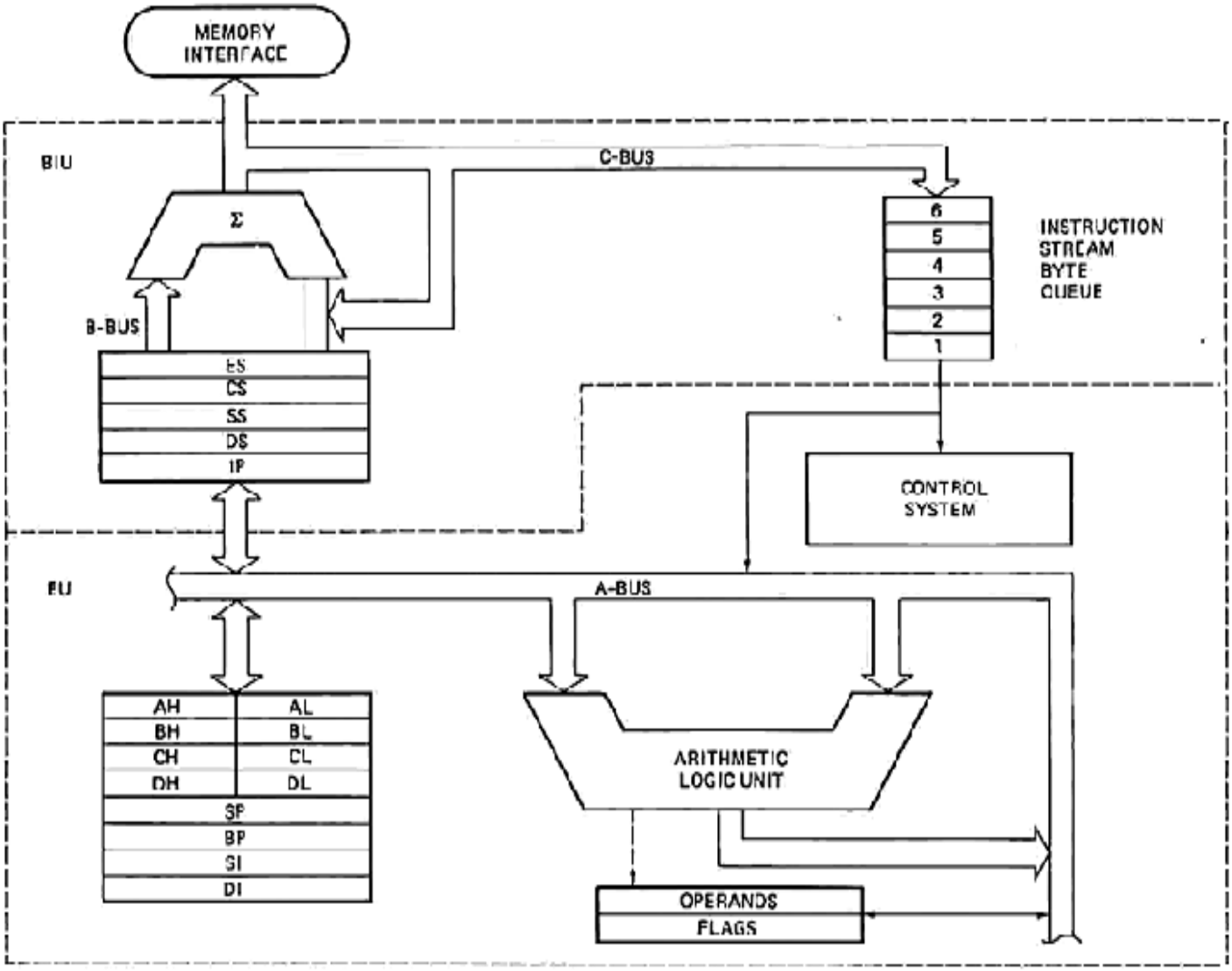

Fig 1: Intel 8086 internal architecture

Assembly language consists of the following instructions:

- **Data Transfer**

¤ MOV AX, BX; register: move contents of BX to AX

¤ COUNT to AX

MOV AX, COUNT; direct: move contents of the address labeled

¤ MOV CX, 0F0H; immediate: load CX with the value 240

¤ MOV CX, [0F0H]; memory: load CX with the value at address 240

¤ MOV [BX], AL; register indirect: move contents of AL to memory location in BX

16-bit registers can be pushed (the SP is first decremented by two and then the is value stored at the address in SP) or popped (the value is restored from the memory at SP and then SP is incremented by 2). For example:

¤ PUSH AX ; push contents of AX
¤ POP BX ; restore into B

- **I/O Operations**

The 8086 has separate I/O and memory address spaces. Values in the I/O space are accessed with IN and OUT instructions. The port address is loaded into DX and the data is read/written to/from AL or AX:

¤ MOV DX,372H ; load DX with port address
¤ OUT DX,AL ; output byte in AL to port372 (hex)
¤ IN AX,DX ; input word to AX

- **Arithmetic/Logic**

Arithmetic and logic instructions can be performed on byte and 16-bit values. The first operand has to be a register and the result is stored in that register.

¤ ADD BX, 4; increment BX by 4

¤ ADD AX, CX; AX= AX + CX

¤ SUB AL, 1; subtract 1 from AL

¤ SUB DX, CX; DX= DX – CX

¤ INC BX; increment BX

¤ CMP AX, 54h; compare (subtract and set flags but without storing result)

¤ XOR AX, AX; clear AX

- **Control Transfer**

Conditional jumps transfer control to another address depending on the values of the flags in the flag register. Conditional jumps are restricted to a range of -128 to +127 bytes from the next instruction while unconditional jumps can be to any point.

¤ JZ skip; jump to label defined as ‘skip’ if last result was zero (two values equal)

¤ JGE notneg; jump to label defined as‘notneg’ if greater than or equal

¤ JB smaller; jump to label defined as ‘smaller’ if below

¤ JMP loop; unconditional jump to a label defined by ‘loop’

*all jump instructions jump to a level defined by a label. Label can be any name; used to define a specific location of code as needed by the programmer.

Most instructions can operate on the general-purpose register set. By specifying the name of the register as an operand to the instruction, you may access the contents of that register. Consider the mov (move) instruction:

```
mov     destination, source
```

This instruction copies the data from the source operand to the destination operand. The eight and 16-bit registers are certainly valid operands for this instruction. The only restriction is that both operands must be the same size. Now let's look at some actual mov instructions:

```
mov    ax, bx  ;Copies the value from BX into AX
mov    dl, al   ;Copies the value from AL into DL
mov    si, dx   ;Copies the value from DX into SI
mov    sp, bp  ;Copies the value from BP into SP
mov    dh, cl   ;Copies the value from CL into DH
       mov    ax, ax  ;Yes, this is legal!
```

The registers are the best place to keep often-used variables.
In addition to the general-purpose registers, many 8086 instructions (including the mov instruction) allow you to specify one of the segment registers as an operand. There are two restrictions on the use of the segment registers with the mov instruction. First of all, one may not specify cs as the destination operand; second, only one of the operands can be a segment register. Data cannot be moved from one **segment register** to another with a single mov instruction. To copy the value of cs to ds, you'd have to use some sequence like:

```
mov    ax, cs
mov    ds, ax
```

Some common instructions used :

inc **Increment by 1**
Syntax: inc op
op: register or memory
Action: op = op + 1

dec **Decrement by 1**
Syntax: dec op
op: register or memory
Action: op = op - 1

mul **Unsigned multiply**
Syntax: mul op8
mul op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AX = AL * op8
If operand is op16, unsigned DX::AX = AX * op16

div **Unsigned divide**
Syntax: div op8
div op16
op8: 8-bit register or memory

op16: 16-bit register or memory
Action: If operand is op8, unsigned AL = AX / op8 and AH = AX % op8
If operand is op16, unsigned AX = DX::AX / op16 and DX = DX::AX % op16

LOOP B1

The LOOP instruction is a combination of a decrement of CX and a conditional jump. In the 8086, LOOP decrements CX and if CX is not equal to zero, it jumps to the address indicated by the label B1. If CX becomes a 0, the next sequential instruction executes.

Important tips

There are some things to note about Intel assembly language syntax:

- The order of the operands is *destination, source*
- Semicolons begin a comment
- The suffix 'H' is used to indicate a hexadecimal constant, if the constant begins with a letter it must be prefixed with a zero to distinguish it from a label
- The suffix 'B' indicates a binary constant
- Square brackets indicate indirect addressing or direct addressing to memory (with a constant)
- The size of the transfer (byte or word) is determined by the size of the *register*

Equipment:

1) EMU8086 [ver.408 (32 bit WINOS compatible)]
2) PC having Intel Microprocessor

Lab Procedure:

Familiarize with emulator EMU8086

Lab Task:

Write a program to exchange the contents of two registers.

Source Code

```
code segment
assume cs:code, ds:code
mov bx, 1234h
mov cx, 5678h
xchg bx, cx
hlt
code ends
end
```

```
code segment
assume cs:code, ds:code
mov bx, 1234h
mov cx, 5678h
mov cx,ax
mov ax,bx
mov bx,cx
hlt
code ends
end
```

Write a program for adding two numbers.

```
org 100h
code segment
assume cs:code, ds:code
```

```
mov al, 13h
mov dl, 01h
add al, dl
hlt
code ends
end
```

Write a program for subtraction between two numbers

```
org 100h
code segment
 assume cs:code, ds:code


mov al, 13h
mov dh, 01h
sub al, dh
hlt
code ends
end
```

Summation of a series: **[1+2+3+4+……+N] = BX .** The value of N is stored in CX.

Source code

```
code segment
        assume cs:code, ds:code

xor bx, bx
mov cx, 9

  start:
        add bx, cx
        loop start


hlt
        code ends
                End
```

Write a program which display two charaters at column#12 and row#7 at emulator screen.

```
        org  100h    ;

        include "emu8086.inc"

        GOTOXY 12,7
        PUTC 65
        PUTC 'B'

        ret
```

Write the assembly code for the following sequence 1+3+5+7…..+N. Where N = 5 using a loop.

```
org 100h

mov ax,0h
mov bx,1h
mov cx, 5h

a1:
   add ax,bx
   inc bx
   inc bx
   loop a1

ret
```

Write a code for finding the value of 6!

```
org 100h
mov ax,1h
mov cx, 6h

l1:
mul cx
loop l1
ret
```

Questions for Report writing:

1. Include all codes' list file printouts following lab report writing template mentioned in appendix A.
2. Write the assembly language program for DX= AX + BX – CX. Show the result on the emulator screen of DX.

Conclusion:

While writing each program, it is important to understand the purpose behind using each function and the associated values. It is very important to understand each line of code so that an individual can solve a given task or problem using the common functions that were learned from this experiment.