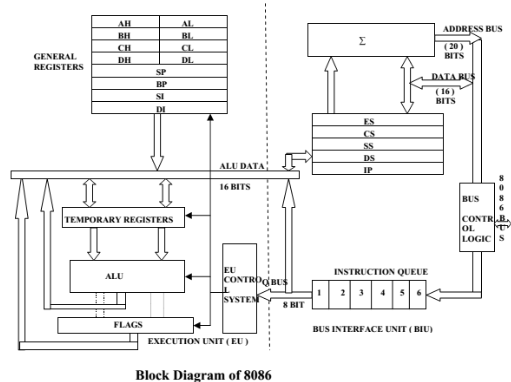
2. Use based indexed addressing mode to do a 3\*3 matrix computation. The user is requested to key in the type of computation and there after the elements of the matrices

**Objective**

To write a program that performs basic matrix calculations in assembly language

**Theory**



**Introduction**

The 8086 microprocessor is one of the most popular Intel microprocessors from its early line of products. It is a 16-bit device, with a 20-bit data bus. The basic architecture of the processor is as illustrated below:

The Arithmetic and Logic Unit performs basic arithmetic operations like addition and subtraction, and bit shifting. The control unit co-ordinates all other operations in the various sections, and handles all timing functions. Registers are small memories that are directly accessible to the processor, and are very fast. They are used as temporary memories in processor operations. The system bus is used to transfer data and for addressing in the 8086 processor.

**Registers**

For the processor to perform an arithmetic operation, the operands are first fetched from the memories and loaded onto registers. The Arithmetic and Logic Unit accesses the operand from the registers, performs the operations, and then saves the result in a register. The result may then be saved to the main memory. Registers in the Intel 8086 are:  Accumulator – mainly used for arithmetic operations

* Base register – used in base-plus-index addressing
* Count register – used to count how many times the program has looped, or is to loop
* Data register – holds part of a result from arithmetic operations such as multiplication or addition  Base pointer – used to point to a memory location
* Destination index – often used to address string data at the destination
* Source index – used to address string data at the source

These four are general purpose registers. They are also known as multi-purpose registers, since the programmer can access them freely and use them as they please. Some special purpose registers include:

* Instruction pointer – points to the next instruction to be executed in a program.
* Segment registers – point to the first memory addresses of the code, data, extra, and stack segments
* Flags register – used to indicate the state of various events in the microprocessor
* Stack pointer – points to addresses in the stack memory.

**Addressing Modes**

For the processor to be able to save or retrieve data from the main memory, it needs to specify the specific location. This is achieved using addressing. The processor specifies a logical address which is then converted to a physical address by the Memory Management Unit.

8086 addressing is done in a segment: offset fashion. Segment registers are used to point to specific segments in the memories. These are the code segment, data segment, stack segment, and extra segment. The offset is the displacement of the desired memory location from the first address of the segment.

Depending on the method used to specify the data or location of data, there are various types of addressing modes:

1. Register addressing Copies the contents of the source register to the destination register:

MOV AX, BX copies the contents of BX into AX

1. Immediate addressing Transfers the immediate data specified as the source into the destination register:

MOV AL, 020h changes the contents of AL to 020h

1. Direct memory addressing The offset value is directly stated. The segment is declared implicitly as the data segment:

MOV AX, [2000h] transfers a copy of the contents of ds:2000h to the accumulator.

1. Register indirect addressing The offset value is first stored in a register: MOV [AX], BX transfers a copy of BX into the location whose offset is the value of AX.
2. Based indexed addressing The value of the offset is given as the sum of the value of a base register and an index register:

MOV [BX+DI], AX transfers a copy of the contents of AX into the location whose offset is the value BX summed to the value of DI.This method is useful in addressing arrays and strings. vi) Register relative addressing The offset value is given as the sum of the contents of a register and a constant:

MOV CX, [AX+4]

**Procedure**

1. A high-level flow chart was designed to show the proposed functioning of the program:

2. The program was written in assembly.

3. The program was tested using an 8086 emulator.

**FLOWCHART**

SAVE THE MATRICES

ENTER MATRIX 2

ENTER MATRIX 1

SETUP

ENTER OPERATION

OP

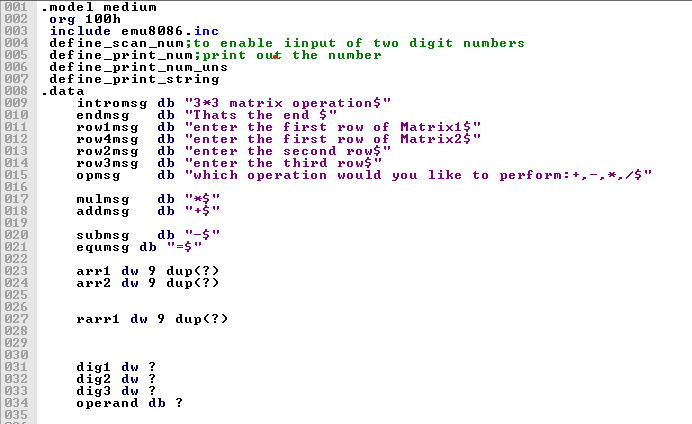
MULTIPLY

SUBTRACT

ADD

DISPLAY RESULT

**1. SETUP**



Arrays used to fetch data were defined in the data segment. These are arr1 for matrix 1 and arr2 for matrix 2

Various messages stored as strings were also defined

**2. Entering Matrix 1 and 2**

gotoxy 0,2

lea dx, row1msg

call printx

jmp input1

input1:

lea bx , arr1

gotoxy 3,4

call scan\_num

mov dig1 ,cx

gotoxy 6,4

call scan\_num

mov dig2 ,cx

gotoxy 9,4

call scan\_num

mov dig3 ,cx

mov dx,dig1

mov [bx],dx

add bx,2d

mov dx, dig2

mov [bx], dx

add bx ,2d

mov dx, dig3

mov [bx],dx

gotoxy 0,6

lea dx,row2msg

call printx

jmp input2

input2:

add bx, 2d

gotoxy 3,8

call scan\_num

mov dig1 ,cx

gotoxy 6,8

call scan\_num

mov dig2 ,cx

gotoxy 9,8

call scan\_num

mov dig3 ,cx

mov dx,dig1

mov [bx],dx

add bx,2d

mov dx, dig2

mov [bx],dx

add bx ,2d

mov dx, dig3

mov [bx],dx

gotoxy 0,10

lea dx,row3msg

call printx

jmp input3

input3:

add bx , 2d

gotoxy 3,12

call scan\_num

mov dig1 ,cx

gotoxy 6,12

call scan\_num

mov dig2 ,cx

gotoxy 9,12

call scan\_num

mov dig3 ,cx

mov dx,dig1

mov [bx],dx

add bx,2d

mov dx, dig2

mov [bx],dx

add bx ,2d

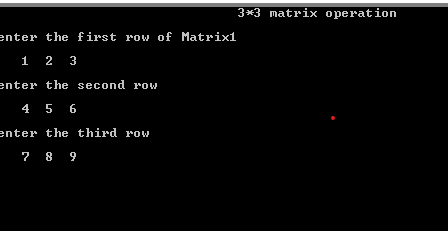
mov dx, dig3

mov [bx],dx

Dig 1, 2 and 3 are used to temporarily hold the rows of the matrices .

A defined function scan\_num is used to read the users input and stores it in the control register cx.

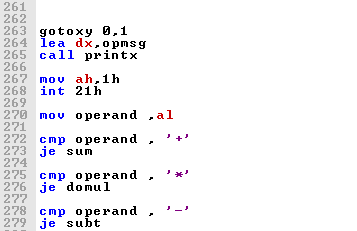
The matrices are saved in arr1 and arr2 respectively



**3. Choosing the operator**

The user has a choice of three matrix operations i.e. multiplication, addition and subtraction.

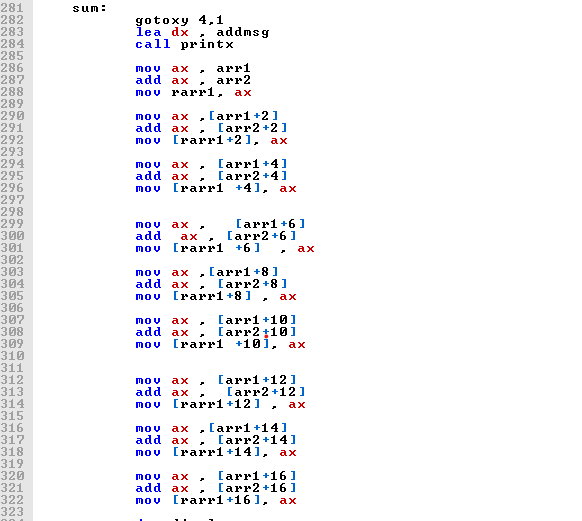
The user inputs the symbol of the operation he / she would like.its then compared and the program jumps to the relevant function



**4. Operations**

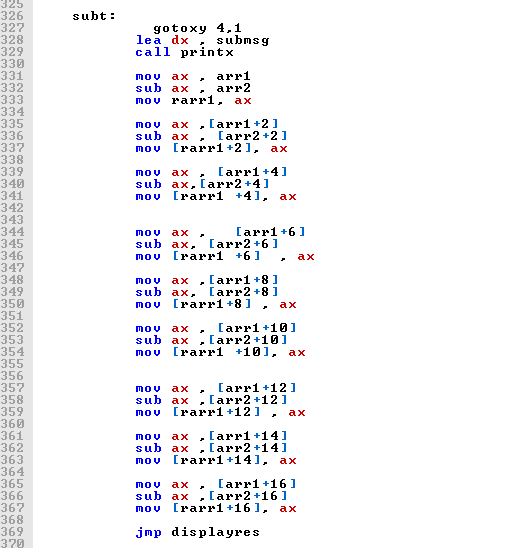
There are three operations that can be carried out

1. Addition



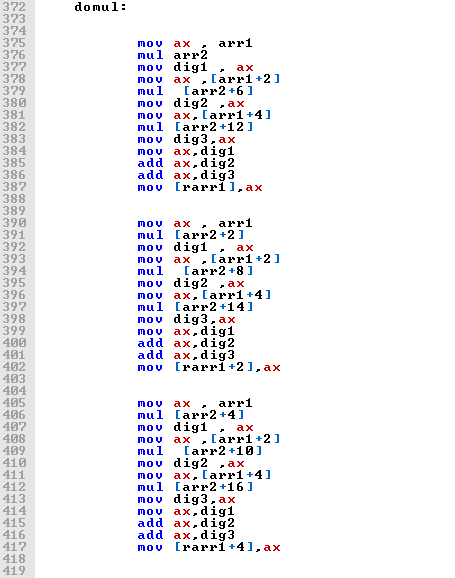
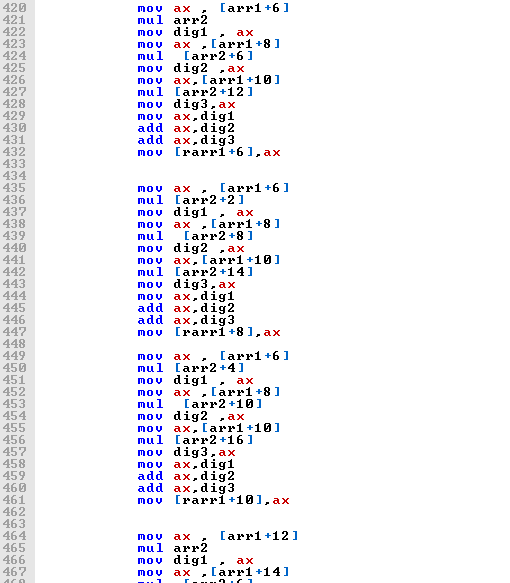
Matrix 1 is added to matrix 2 and the result is stored in rarr1(result array)

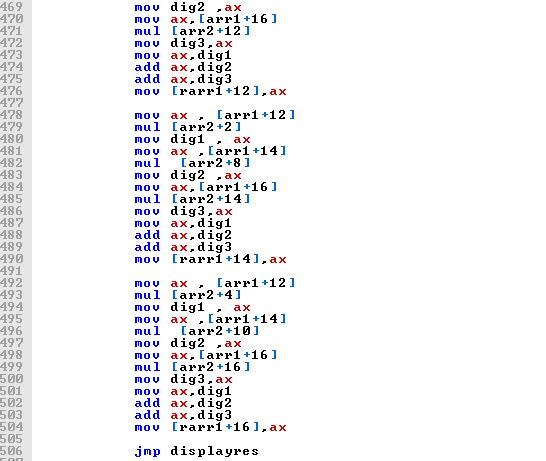
1. Subtraction



Matrix 2 is subtracted from matrix 1 and the result stored in result array (rarr1).

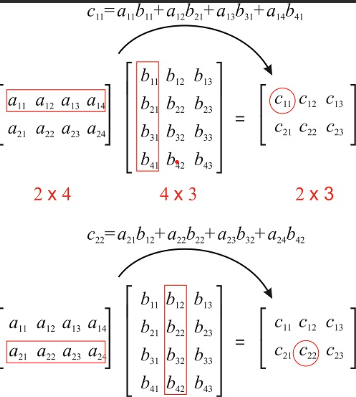
1. Multiplication



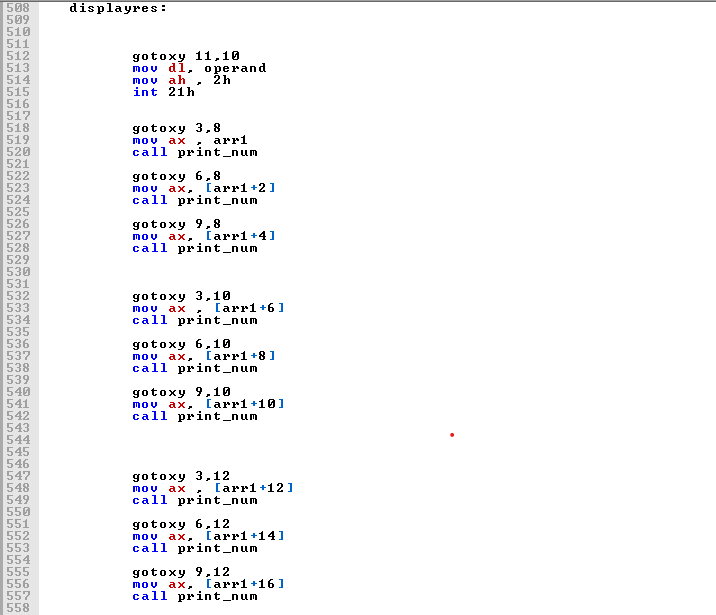
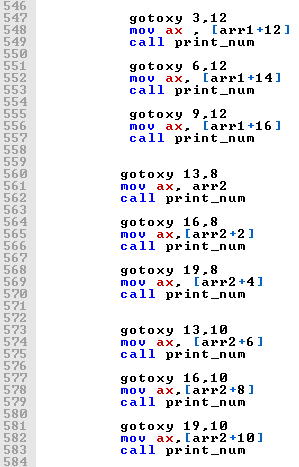
Matrix 1 and 2 are multiplied using the general laws of multiplication of matrices.

Dig 1 ,2 and 3 were used to temporarily store results of the operation



**5. Displaying the result**

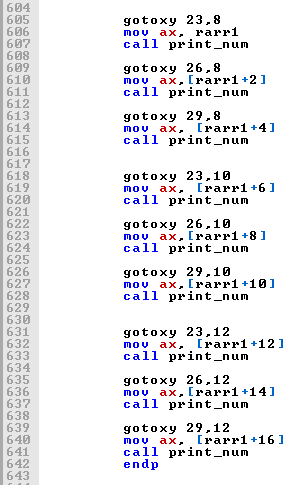
After an operation has been performed a function called displayres is jumped into.This function is defined and is used to print out the results of the operation by displaying the two matrices, the operation symbol and the resulting array.

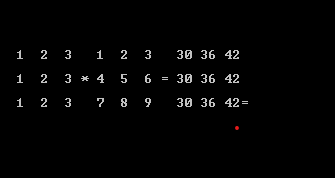
 

Displaying Array 1 and Array 2

A function called **print\_num** defined in **include emu8086.inc** is used to display the numbers on the console

Displaying the result





Defined functions, macros and procedures used

**GOTOXY col, row** - macro with 2 parameters, sets cursor position.

**Scan\_num**

procedure that gets the multi-digit SIGNED number from the keyboard, and stores the result in **CX** register. To use it declare: **DEFINE\_SCAN\_NUM** before **END** directive.

**Print\_num**

It’s also a common function found in the emu8086.inc library procedure that prints a signed number in **AX** register. To use it declare: **DEFINE\_PRINT\_NUM** and **DEFINE\_PRINT\_NUM\_UNS** before **END** directive.