Exp No: 5

Date: 08/10/2020

# Matrix operations

Name: Swetha Saseendran

Reg No: 185001183

## Aim:

To write and execute 8086 programs for Matrix operations (Addition & Subtraction).

## Programs:

### (i) Matrix Addition

## Algorithm:

- Declare the data segment.
- Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
- Close the data segment.

- Declare the code segment.
- Set a preferred offset (preferably 100)
- Load the data segment content into AX register.
- Transfer the contents of AX register to DS register.
- Compare row1 and row2, if not equal then exit the program.
- Compare col1 and col2, if not equal then exit the program.
- Position SI at matrix1, and DI at matrix2.
- Multiply row1 and col1 to find length len of the matrix.
- Move the len to CL register.
- Till CL goes to zero:
    - Add values at SI and DI and push it into the stack.
    - Increment SI and DI.
    - Decrement CL.
- Move SI to end of resultant matrix.
- Till CL goes to zero:
    - Pop the value from top of the stack and put it at SI.
    - Decrement SI.

| Program | Comments |
|---|---|
| **assume** *cs:code, ds:data* | Using assume directive to declare data, extra and code segment |
| ***data segment*** | Using assume directive to declare data, extra and code segment |
| *mat1 db 23h,24h,55h,11h* | |
| *mat2 db 21h,44h,57h,22h* | |
| *row1 db 02h* | |
| *col1 db 02h* | |
| *row2 db 02h* | |
| *col2 db 02h* | |
| *len db 00h* | |
| *resi dw ?* | |
| *data ends* | |
| | |
| *code segment* | Start the code segment. |
| *org 0100h* | Initialize an offset address. |
| ***start:*** *mov ax, data* | Transfer data from "data" to AX. |
| *mov ds, ax* | Transfer data from memory location AX to DS. |
| | |
| *mov al, row1* | Move row1 to AL |
| *mov bl, row2* | Move row2 to BL |
| *cmp al, bl* | Comparing row count of both matrices. |
| *jne break* | Exiting if not same. |
| *mov al, col1* | Move col1 to AL |
| *mov bl, col2* | Move col2 to BL |
| *cmp al, bl* | Comparing col count of both matrices. |
| *jne break* | Exiting if not same. |
| *mov si, offset mat1* | Set SI to point to Matrix 1's starting index. |
| *mov di, offset mat2* | Set DI to point to Matrix 2's starting index. |
| *mov al, row1* | Move row1 to AL |
| *mov bl, col1* | Move row2 to BL |
| *mul bl* | AL has the value of row1 * col1. |
| *mov len, al* | Move len to AL |
| *mov cl, len* | Finding no. of elements in the matrix. |
| *mov ch, 00h* | Clear CH. |
| *mov ax, 0000h* | Clear AX. |
| | |
| ***looper:*** *mov al, [si]* | Pushing each element-wise sum into stack |
| *mov ah, 00h* | AH <- 00H |
| *mov bl, [di]* | |
| *mov bh, 00h* | BH <- 00H |
| *add ax, bx* | Add the 2 elements from each matrix. |

| | |
|---|---|
| *push ax* | |
| *inc si* | Move to next element in matrix 1. |
| *inc di* | Move to next element in matrix 2. |
| *dec cx* | Decrement counter by 1. |
| *jz prewrk* | If addition is over, jump to prewrk |
| *jmp looper* | Repeat addition for all elements. |
| | |
| ***prewrk:*** *mov si, offset resi + 0001h* | Set the SI to store values in result matrix "resi" properly. |
| *mov cl, len* | Set counter to length of the matrix. |
| *mov ch, 00h* | Clear CH. |
| *add si, cx* | Set SI to point to the last location of the matrix. |
| | |
| ***retloop:*** *pop ax* | Popping each element from stack into resultant matrix. |
| *mov [si], al* | Move AL to [SI] |
| *dec si* | Decrement SI. |
| *mov [si], ah* | Move AL to [SI] |
| *dec si* | Decrement SI |
| *dec cx* | Decrement counter by 1. |
| *jz break* | Stop popping if all elements are popped (CX = 0) |
| *jmp retloop* | Pop the next element and put it in the matrix. |
| | |
| ***break:*** *mov ah, 4ch* | Moves the hexadecimal value 4c to ah. |
| *int 21h* | When Software interrupt 21 is called with AH=4C, then current process terminates. (i.e., These two instructions are used for the termination of the process). |
| *code ends* | |
| *end start* | |

## Unassembled Code:

```
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A00800        MOV     AL,[0008]
076B:0108 8A1E0A00      MOV     BL,[000A]
076B:010C 38D8          CMP     AL,BL
076B:010E 7551          JNZ     0161
076B:0110 A00900        MOV     AL,[0009]
076B:0113 8A1E0B00      MOV     BL,[000B]
076B:0117 38D8          CMP     AL,BL
076B:0119 7546          JNZ     0161
076B:011B BE0000        MOV     SI,0000
076B:011E BF0400        MOV     DI,0004
```

## Snapshot of sample input and output:

INPUT:

```
-d 076A:0000
076A:0000  23 24 55 11 21 44 57 22-02 02 02 02 00 00 00 00   #$U.!DW".......
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

OUTPUT:

```
-g

Program terminated normally
-d 076A:0000
076A:0000  23 24 55 11 21 44 57 22-02 02 02 02 04 00 00 00   #$U.!DW".......
076A:0010  44 00 68 00 AC 00 33 00-00 00 00 00 00 00 00 00   D.h...3.........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

# (ii) Matrix Subtraction

## Algorithm:

- Declare the data segment.
- Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
- Close the data segment.

- Declare the code segment.
- Set a preferred offset (preferably 100)
- Load the data segment content into AX register.
- Transfer the contents of AX register to DS register.
- Compare row1 and row2, if not equal then exit the program
- Compare col1 and col2, if not equal then exit the program
- Position SI at matrix1, and DI at matrix2.
- Multiply row1 and col1 to find length len of the matrix.
- Move the len to CL register.
- Till CL goes to zero:
    - Subtract values at SI and DI and push it into the stack.
    - Increment SI and DI.
    - Decrement CL.
- Move SI to end of resultant matrix.
- Till CL goes to zero:
    - Pop the value from top of the stack and put it at SI.
    - Decrement SI.

| *Program* | *Comments* |
|---|---|
| ***assume*** *cs:code, ds:data* | Using assume directive to declare data, extra and code segment |
| ***data segment*** | Using assume directive to declare data, extra and code segment |
| *mat1 db 23h,24h,55h,11h* | |
| *mat2 db 21h,44h,57h,22h* | |
| *row1 db 02h* | |

| | |
|---|---|
| *col1 db 02h* | |
| *row2 db 02h* | |
| *col2 db 02h* | |
| *len db 00H* | |
| *resi dw ?* | |
| ***data ends*** | |
| | |
| ***code segment*** | Start the code segment. |
| *org 0100h* | Initialize an offset address. |
| ***start:*** *mov ax, data* | Transfer data from "data" to AX. |
| *mov ds, ax* | Transfer data from memory location AX to DS. |
| | |
| *mov al, row1* | Move row1 to AL |
| *mov bl, row2* | Move row2 to BL |
| *cmp al, bl* | Comparing row count of both matrices. |
| *jne break* | Exiting if not same. |
| *mov al, col1* | Move col1 to AL |
| *mov bl, col2* | Move col2 to BL |
| *cmp al, bl* | Comparing col count of both matrices. |
| *jne break* | Exiting if not same. |
| *mov si, offset mat1* | Set SI to point to Matrix 1's starting index. |
| *mov di, offset mat2* | Set DI to point to Matrix 2's starting index. |
| *mov al, row1* | Move row1 to AL |
| *mov bl, col1* | Move row2 to BL |
| *mul bl* | AL has the value of row1 * col1. |
| *mov len, al* | Move len to AL |
| *mov cl, len* | Finding no. of elements in the matrix. |
| *mov ch, 00h* | Clear CH. |
| *mov ax, 0000h* | Clear AX. |
| | |
| ***looper:*** *mov al, [si]* | Pushing each element-wise sum into stack |
| *mov ah, 00h* | AH <- 00H |
| *mov bl, [di]* | |
| *mov bh, 00h* | BH <- 00H |
| *sub ax, bx* | Subtract the 2 elements from each matrix. |
| *push ax* | |
| *inc si* | Move to next element in matrix 2. |
| *inc di* | Move to next element in matrix 1. |
| *dec cx* | Decrement counter by 1. |
| *jz prewrk* | If addition is over, jump to "prewrk" |
| *jmp looper* | Repeat addition for all elements. |

| | | |
|---|---|---|
| ***prewrk:*** | *mov si, offset resi + 0001h* | Set the SI to store values in result matrix “resi” properly. |
| | *mov cl, len* | Set counter to length of the matrix. |
| | *mov ch, 00h* | Clear CH. |
| | *add si, cx* | Set SI to point to the last location of the matrix. |
| | *add si, cx* | |
| | | |
| ***retloop:*** | *pop ax* | Popping each element from stack into resultant matrix. |
| | *mov [si], al* | Move AL to [SI] |
| | *dec si* | Decrement SI. |
| | *mov [si], ah* | Move AL to [SI] |
| | *dec si* | Decrement SI |
| | *dec cx* | Decrement counter by 1. |
| | *jz break* | Stop popping if all elements are popped (CX = 0) |
| | *jmp retloop* | Pop the next element and put it in the matrix. |
| | | |
| ***break:*** | *mov ah, 4ch* | Moves the hexadecimal value 4c to ah. |
| | *int 21h* | When Software interrupt 21 is called with AH=4C, then current process terminates. (i.e., These two instructions are used for the termination of the process). |
| | ***code ends*** | |
| | ***end start*** | |

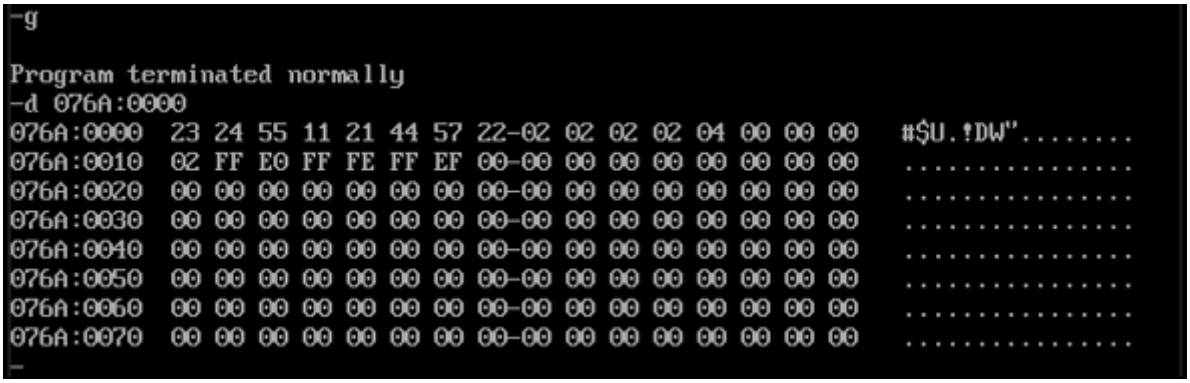
## Unassembled Code:

```
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A00800        MOV     AL,[0008]
076B:0108 8A1E0A00      MOV     BL,[000A]
076B:010C 38D8          CMP     AL,BL
076B:010E 7551          JNZ     0161
076B:0110 A00900        MOV     AL,[0009]
076B:0113 8A1E0B00      MOV     BL,[000B]
076B:0117 38D8          CMP     AL,BL
076B:0119 7546          JNZ     0161
076B:011B BE0000        MOV     SI,0000
076B:011E BF0400        MOV     DI,0004
```

## Snapshot of sample input and output:

INPUT:


OUTPUT:




## Result:

The assembly level programs were written to perform the above specified matrix operations and the result was verified.