|  |  |  |
| --- | --- | --- |
|  | MATRIX OPERATIONS |  |
| Exp No.: 5 |  | **Name:** S Vishakan |
| Date: 23-09-2020 |  | **Reg. No:** 18 5001 196 |

**AIM:**

To write assembly language programs to perform the following matrix operations:

1. Matrix Addition.
2. Matrix Subtraction.

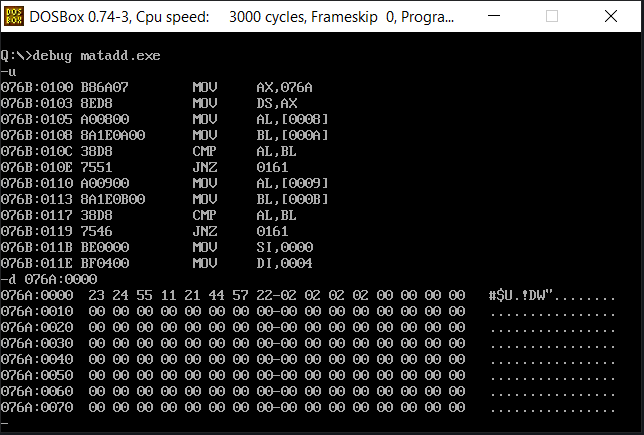
**PROGRAM – 1: MATRIX ADDITION:**

**ALGORITHM:**

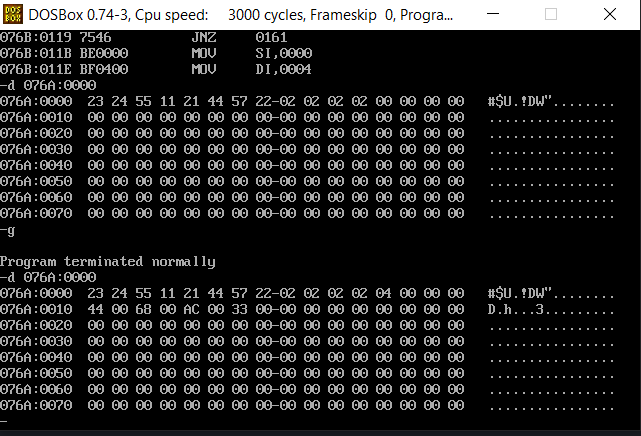
1. Begin.
2. Declare the data segment.
3. Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Compare row1 and row2, if not equal then exit the program.
10. Compare col1 and col2, if not equal then exit the program.
11. Position SI at matrix1, and DI at matrix2.
12. Multiply row1 and col1 to find length len of the matrix.
13. Move the len to CL register.
14. Till CL goes to zero:
    1. Add values at SI and DI and push it into the stack.
    2. Increment SI and DI.
    3. Decrement CL.
15. Move SI to end of resultant matrix.
16. Till CL goes to zero:
    1. Pop the value from top of the stack and put it at SI.
    2. Decrement SI.
17. Introduce an interrupt for safe exit. (INT 21h)
18. Close the code segment.
19. End.

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| mat1 db 23h,24h,55h,11h | Matrix 1. |
| mat2 db 21h,44h,57h,22h | Matrix 2. |
| row1 db 02h | Row count of matrix 1. |
| col1 db 02h | Column count of matrix 1. |
| row2 db 02h | Row count of matrix 2. |
| col2 db 02h | Column count of matrix 2. |
| len db 00h | Length of matrix. |
| resi dw ? | Result matrix. |
| 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 | Comparing row count of both matrices. |
| mov bl, row2 |  |
| cmp al, bl |  |
| jne break | Exiting if not same. |
| mov al, col1 | Comparing column count of both matrices. |
| mov bl, col2 |  |
| cmp al, bl |  |
| 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 |  |
| mov bl, col1 |  |
| mul bl | AL has the value of row1 \* col1. |
| mov len, 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 |  |
| mov bl, [di] |  |
| mov 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 |  |
| dec si | Decrement SI. |
| mov [si], ah |  |
| dec 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 |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



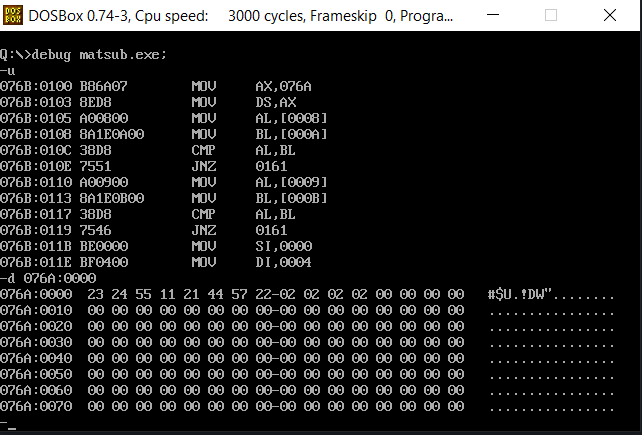
**PROGRAM – 2: MATRIX SUBTRACTION:**

**ALGORITHM:**

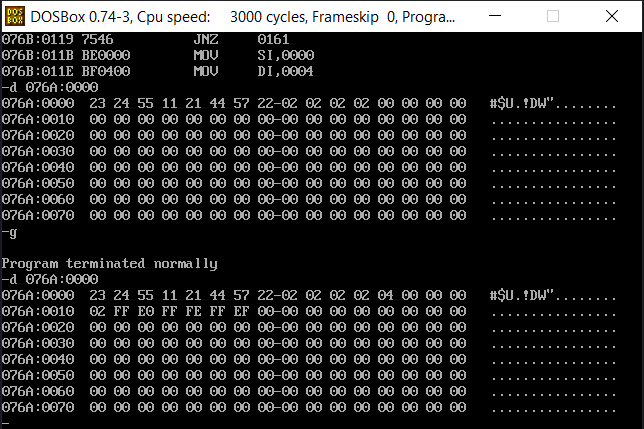
1. Begin.
2. Declare the data segment.
3. Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Compare row1 and row2, if not equal then exit the program
10. Compare col1 and col2, if not equal then exit the program
11. Position SI at matrix1, and DI at matrix2.
12. Multiply row1 and col1 to find length len of the matrix.
13. Move the len to CL register.
14. Till CL goes to zero:
    1. Subtract values at SI and DI and push it into the stack.
    2. Increment SI and DI.
    3. Decrement CL.
15. Move SI to end of resultant matrix.
16. Till CL goes to zero:
    1. Pop the value from top of the stack and put it at SI.
    2. Decrement SI.
17. Introduce an interrupt for safe exit. (INT 21h)
18. Close the code segment.
19. End.

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| mat1 db 23h,24h,55h,11h | Matrix 1. |
| mat2 db 21h,44h,57h,22h | Matrix 2. |
| row1 db 02h | Row count of matrix 1. |
| col1 db 02h | Column count of matrix 1. |
| row2 db 02h | Row count of matrix 2. |
| col2 db 02h | Column count of matrix 2. |
| len db 00h | Length of matrix. |
| resi dw ? | Result matrix. |
| 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 | Comparing row count of both matrices. |
| mov bl, row2 |  |
| cmp al, bl |  |
| jne break | Exiting if not same. |
| mov al, col1 | Comparing column count of both matrices. |
| mov bl, col2 |  |
| cmp al, bl |  |
| 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 |  |
| mov bl, col1 |  |
| mul bl | AL has the value of row1 \* col1. |
| mov len, 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 |  |
| mov bl, [di] |  |
| mov 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 |  |
| dec si | Decrement SI. |
| mov [si], ah |  |
| dec 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 |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



**RESULT:**

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