Mini Project Report on

Implementation of Matrix Multiplication using Asssembly language

by

Mangesh (18CE1062)

Priyanshu (18CE1059)

Pramey (18CE1045)

Prathamesh (18CE1054)

Under the guidance of

Mr. Prathmesh Gunjgur

Department of Computer Engineering

Ramrao Adik Institute of Technology

Dr. D. Y. Patil Vidyanagar, Nerul, Navi Mumbai

University of Mumbai

April 2020

Ramrao Adik Institute of Technology

Dr. D. Y. Patil Vidyanagar, Nerul, Navi Mumbai

CERTIFICATE

This is to certify that Mini Project report entitled

Implementation of matrix multiplication using assembly language

by

Mangesh Deshmukh

(18CE1062)

Priyanshu Dubey

(18CE1059)

Pramey Dongre

(18CE1045)

Prathamesh Gade

(18CE1054)

is successfully completed for Second Year of Computer Engineering as prescribed by University of Mumbai.

Supervisor

(Mr. Prathmesh Gunjgur)

DECLARATION

We declare that this written submission represents our ideas and does not involve plagiarism. We have adequately cited and referenced the original sources wherever other's ideas or words have been included. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/ source in our submission. We understand that any violation of the above will be cause for disciplinary action against me by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 19-03-2020

Mangesh Deshmukh
(18CE1062)
Priyanshu Dubey
(18ce1059)
Pramey Dongre
(18ce1045)
Prathamesh Gade
(18CE1054)

Contents

1	**Title**	**5**
2	**Objective**	**5**
3	**Hardware & Software Requirements**	**5**
4	**Theory**	**5**
5	**Design**	**6**
6	**Implementation**	**7**
7	**Result Analysis**	**16**
8	**Conclusion**	**17**
9	**References**	**17**

1. **Title : Implementation of Matrix Multiplication using Assembly Language**

2. **Objective :** To implement matrix multiplication using low level approach

3. **Hardware & Software Requirements : Nasm assembler, Text Editor, Linux System**

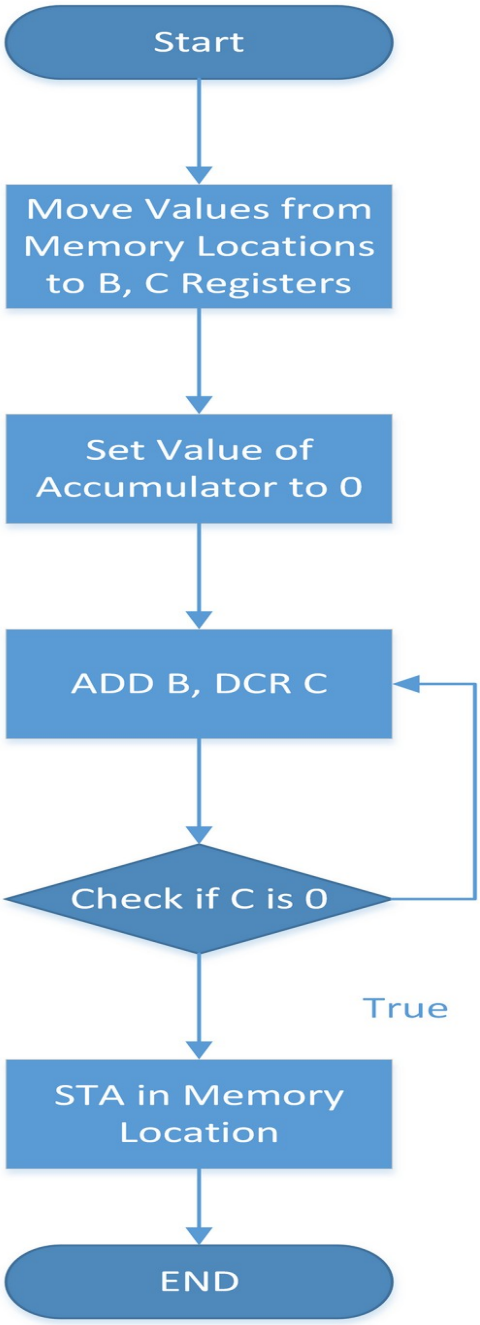
4. **Theory :** We know how to add vectors and how to multiply them by scalars. Together, these operations give us the possibility of making linear combinations. Similarly, we know how to add matrices and how to multiply matrices by scalars. In this section we mix all these ideas together and produce an operation known as “matrix multiplication.” This will lead to some results that are both surprising and central. We begin with a definition of how to multiply a vector by a matrix.

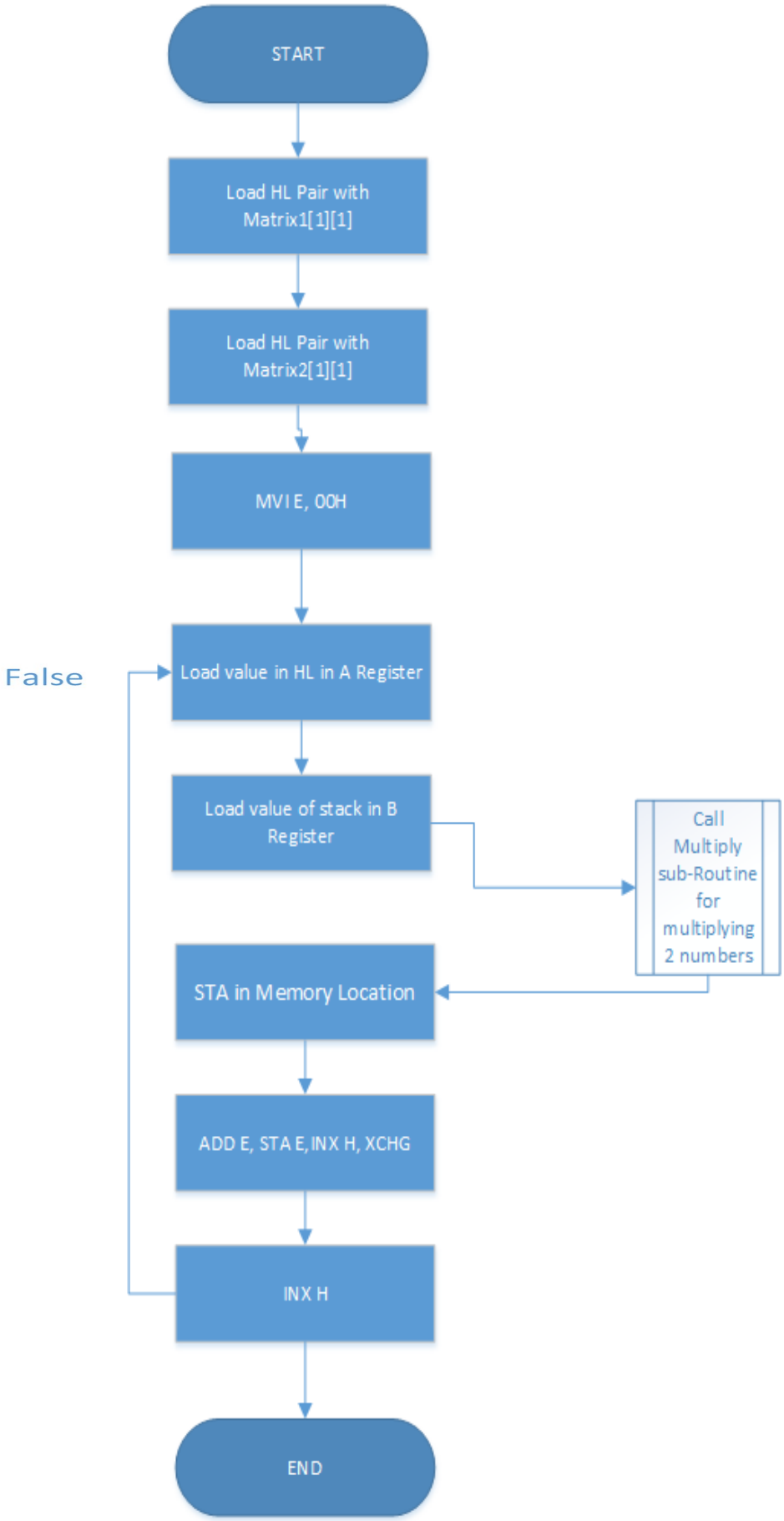
Suppose A is an m×n matrix with columns $A_1, A_2, A_3, \ldots, A_n$ and u is a vector of size n. Then the *matrix-vector product* of A with u is the linear combination

$$Ai = [u]_1 A_1 + [u]_2 A_2 + [u]_3 A_3 + \ldots + [u]_n A_n$$

So, the matrix-vector product is yet another version of “multiplication,” at least in the sense that we have yet again overloaded juxtaposition of two symbols as our notation. Remember your objects, an m×n matrix times a vector of size n will create a vector of size m. So if A is rectangular, then the size of the vector changes. With all the linear combinations we have performed so far, this computation should now seem second nature.

5. **Design :**


Start
Move Values from Memory Locations to B, C Registers
Set Value of Accumulator to 0
ADD B, DCR C
Check if C is 0
True
STA in Memory Location
END


START
Load HL Pair with Matrix1[1][1]
Load HL Pair with Matrix2[1][1]
MVI E, 00H
Load value in HL in A Register
False
Load value of stack in B Register
Call Multiply sub-Routine for multiplying 2 numbers
STA in Memory Location
ADD E, STA E, INX H, XCHG
INX H
END

6. Implementation :

```
%include "./src/io.mac"

.DATA

        msgRow db "Enter the number of rows : ",0

        msgCol db "Enter the number of cols : ",0

        msgMat1 db "Matrix 1 : ",0xA,0xD,0

        msgMat2 db "Matrix 2 : ",0xA,0xD,0

        msgEle db "Enter the elements : ",0xA,0xD,0

        msgWrong db "Wrong size of matrices",0xA,0xD,"Columns in 1st should be equal to Rows in 2nd",0

        space db " ",0

        answer db "Resultant Matrix : ",0xA,0xD,0

        com db ",",0


.UDATA

        n resw 1

        m resw 1

        p resw 1

        q resw 1

        matA resw 100

        sizeA resw 1

        matB resw 100

        sizeB resw 1

        matC resw 100
```

```
.CODE
        .STARTUP

        PutStr msgMat1
        PutStr msgRow
        GetInt [n]
        PutStr msgCol
        GetInt [m]
        mov eax,[n]
        imul eax,[m]
        mov [sizeA],eax

        push word[sizeA]
        push matA
        call readMatrix

        PutStr msgMat2
        PutStr msgRow
        GetInt [p]
        PutStr msgCol
        GetInt [q]
        mov eax,[p]
        imul eax,[q]
```

```
mov [sizeB],eax

push word[sizeB]
push matB
call readMatrix

mov ax,word[m]
mov bx,word[p]
cmp ax,bx
jne wrong

PutStr msgMat1
push word[m]
push word[n]
push matA
call printMatrix

PutStr msgMat2
push word[q]
push word[p]
push matB
call printMatrix
```

```
        push word[q]
        push word[m]
        push word[n]
        push matA
        push matB
        push matC
        call mulMatrix

        PutStr answer
        push word[q]
        push word[n]
        push matC
        call printMatrix

        jmp done

wrong:
        PutStr msgWrong
        nwln

done:
        .EXIT

readMatrix:
```

```
        enter 0,0
        ;ebp+8 : address of mat,+12 : size of matrix

        PutStr msgEle
        xor ax,ax
        mov ebx,[ebp+8]
        readLoop:
          GetInt cx
          mov [ebx],cx
          add ebx,2
          inc ax
          cmp ax,[ebp+12]
        jne readLoop
        readLoop_end:
          leave
          ret 6

printMatrix:
        enter 0,0
        ;ebp+8 : address of mat,+12 : number of rows
        ;ebp+14 : number of cols

        xor bx,bx
        mov ecx,[ebp+8]
```

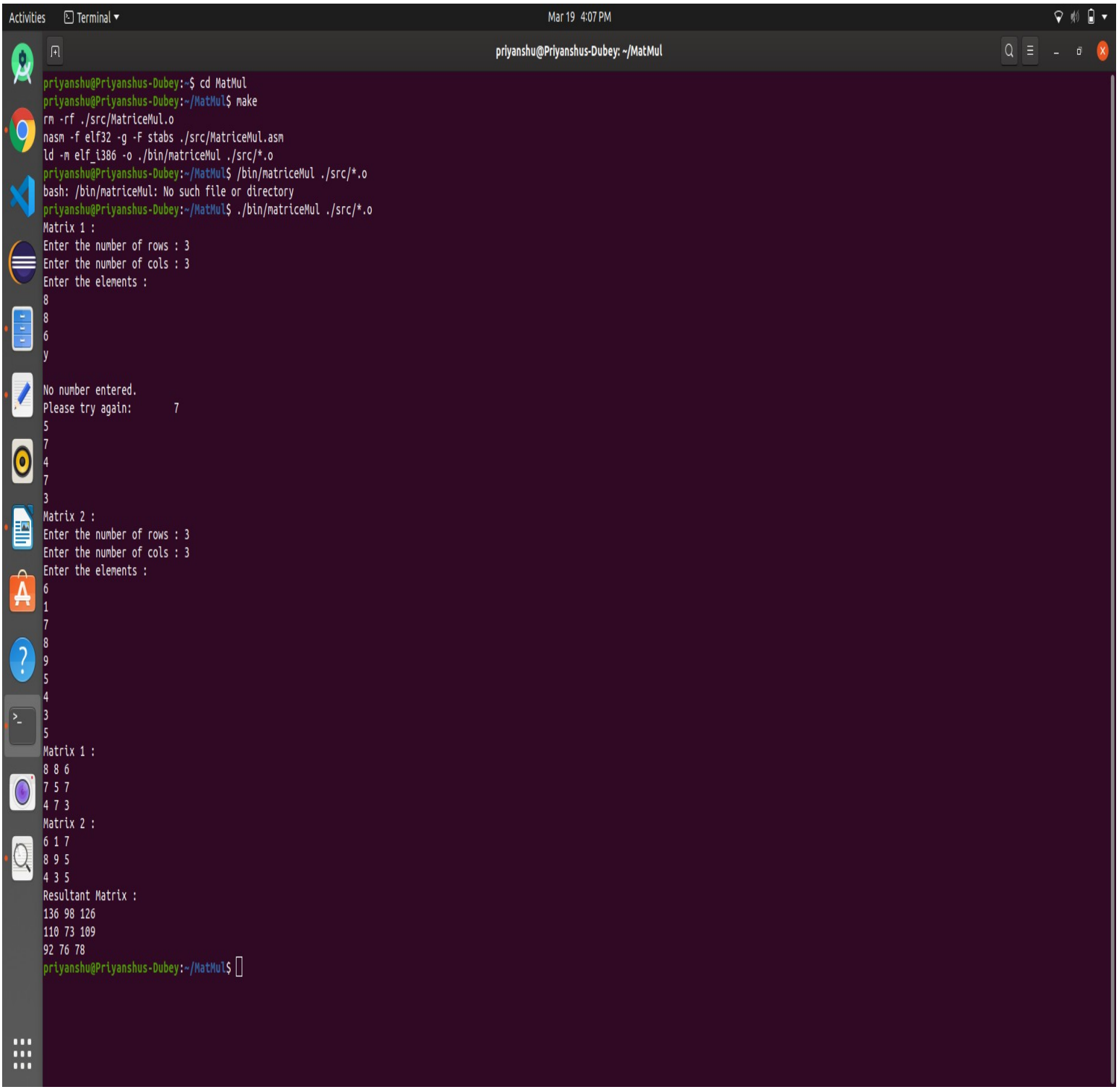
```
        printLoop:
          xor ax,ax
          printRow:
                PutInt word[ecx]
                PutStr space
                add ecx,2
                inc ax
                cmp ax,[ebp+14]
          jne printRow
          nwln
          inc bx
          cmp bx,[ebp+12]
        jne printLoop
        printLoop_end:
          leave
          ret 8


mulMatrix:
        enter 0,0
        ;+8: address of c,+12: address of b
        ;+16: address of a,+20: rows in 1
        ;+22: cols/rows in 1/2,+24: cols in 2
        ;+26: size of a,+28: size of b
```

```
segment .data
        i dw 0
        j dw 0
        k dw 0
        sum dw 0
        ind1 dd 0
        ind2 dd 0

segment .text

        mov eax,[ebp+16]
        mov ebx,[ebp+12]
        mov ecx,[ebp+8]
        mulLoop:
          xor dx,dx
          mov word[j],dx

          mulLoopRow:
                xor dx,dx
                mov word[k],dx
                mov word[sum],dx
                mulLoopAdd:
                        mov dx,word[i]
                        imul dx,[ebp+22]
```

```
add dx,word[k]
mov [ind1],dx

mov dx,word[k]
imul dx,[ebp+24]
add dx,word[j]
mov [ind2],dx

add eax,[ind1]
add eax,[ind1]
add ebx,[ind2]
add ebx,[ind2]

mov dx,[eax]
imul dx,[ebx]
add [sum],dx

sub eax,[ind1]
sub ebx,[ind2]
sub eax,[ind1]
sub ebx,[ind2]

inc word[k]
mov dx,word[k]
```

```
                cmp dx,[ebp+22]
            jne mulLoopAdd

            mov dx,word[sum]
            mov word[ecx],dx
            add ecx,2
            inc word[j]
            mov dx,word[j]
            cmp dx,[ebp+24]
  jne mulLoopRow
  inc word[i]
  mov dx,word[i]
  cmp dx,[ebp+20]
jne mulLoop
mulLoop_end:
  leave
  ret 18
```

7. **Result Analysis (if any) :**



8. Conclusion : We can perform matrix Multiplication using Assembly language but the code for it is too lengthly and time consuming as compared to C/C++, Java or Python.

9. References :

[1] Microprocessor Architecture, Programming, and Applications with the 8085 - S Gaonkar

[2] 8080/8085 Assembly Language Programming Manual Copyright c 1977, 1978 Intel Corporation

[3] http://en.wikipedia.org/wiki/Matrix multiplication