ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

ALGEBRAIC OPERATIONS ON HIGHER-DIMENSIONAL MATRICES USING ASSEMBLY LANGUAGE PROGRAM IN INTEL 8085 MICROPROCESSOR

[1]Mohammed Ibrahim .A & [2]Malarvizhi .V

[1]*Master Student*
Department of Physics
Pachaiyappa's College, Shenoy Nagar, Chennai-600030. India
Email: mi551212550@hotmail.com

[2]*Associate Professor*
Department of Physics
Pachaiyappa's College, Shenoy Nagar, Chennai-600030. India
Email: malarveliah@gmail.com

Abstract

The algebraic operations on lower-order matrices are a simple and easy one. But for higher dimensional matrix the operations are trivial to do manually. In this project we intend to do algebraic operations like addition, subtraction, multiplication of higher dimensional matrix (maximum of 16×16) by using Intel 8085 microprocessor assembly language program (ALP). The reason for choosing the above dimensional limit is that we have small memory sized RAM. The elements of the matrices are restricted to a positive real hexadecimal number since the registers can hold an only positive hexadecimal number.

Keywords*: intel-8085, matrix algebra, programmer model, hexadecimal.*

INTRODUCTION

Matrix algebra is very important as it has a wide application in various fields. An algebraic operation like addition, subtraction, multiplication, scalar multiplication and inverse can be done on matrices with certain rules and regulations. The dimension of a matrix is represented by (**m×n**), where m and n represent the number of rows and column respectively. Also known as the order of a matrix. Where,

$$\mathbf{m, n} \in \mathbb{Z}+$$

The Intel 8085 microprocessor is an 8-bit processor. The programmer model shown in Fig.1 helps the programmer to understand the memory organization of internal memories. It consists of two

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

blocks namely, register block and memory block. The register block consists of an accumulator, six general purpose register which is 8-bit. Besides they have stack pointer and program counter which are 16-bit. The memory block consists of 2K EPROM where the utility programs are stored and 2K RAM where the user programs are stored. Each memory in the memory block can hold an 8-bit value which is addressed by 16-bit numbers. These two blocks are connected by parallel lines namely address bus, data bus and control bus.

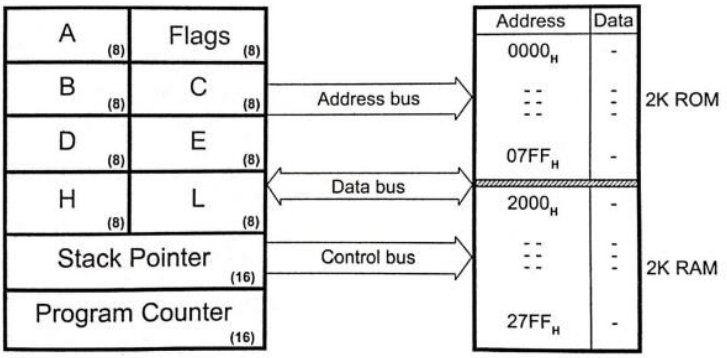


Fig. 1: Programmer model of 8085

CONSTRAINTS

Since the memory addresses are named with hexadecimal numbers and the data hold by memories and registers are (8-bit) 2 digit hexadecimal number, we have to formulate the problem with certain rules. They are,

- The elements of the matrix and the indices described in this project are hexadecimal numbers.
- The order of the matrix is given in decimal number.
- The starting address of RAM (user's program area) is considered to be 4000_H.
- The maximum order of the matrix is 16×16.
- The program can be modified to do operations of matrices of dimension within the above limit. An example we can do operations on (2×10) matrix and so on.

MATRIX ADDITION AND SUBTRACTION

In general, the element of a matrix is represented by "$x_{\mu\vartheta}$" and the matrix itself represented as

$$\mathbf{X} = [x_{\mu\vartheta}]_{m\times n.}$$

Where the m×n is the order of the matrix **X** and μ, ϑ are the indices which represent the μ^{th} row and ϑ^{th} column.

i.e. $X = [x_{\mu\vartheta}]_{m\times n}$ and $Y= [y_{\mu\vartheta}]_{m\times n}$ then $X\pm Y = [x_{\mu\vartheta} \pm y_{\mu\vartheta}]_{m\times n.}$

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Doing with ALP:

Memory mapping: In order to distribute the matrix elements in the memory of RAM, we can consider the last two number of the memory address represent the row and column such that "ZZμϑ".

For example,

Memory address	Data
4151	$0A_H$

$0A_H$ is the element of the matrix of 5th row and 1st column of the corresponding matrix. Here μ and ϑ are indices such that,

$$\mu, \vartheta \in 0,1,2,\ldots\ldots..9,A,B,C,D,E,F.$$

Let us consider an addend/subtrahend matrix **X** of order (16×16), whose elements are located in memory 4100_H to $41FF_H$. There are 256 elements in the matrix.

$$X = \begin{bmatrix} x_{00} & x_{01} & x_{02} & . & . & . & . & x_{0F} \\ x_{10} & x_{11} & x_{12} & . & . & . & . & . \\ x_{20} & x_{21} & x_{22} & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & x_{\mu\vartheta} & . & . \\ . & . & . & . & . & . & . & . \\ x_{F0} & . & . & . & . & . & . & x_{FF} \end{bmatrix}$$

Here how the elements are distributed in memories for this matrix is given in Table.1.

Memory address	**Data**
4100	x_{00}
4101	x_{01}
4102	x_{02}
4103	x_{03}
.	.
.	.
41μϑ	$x_{\mu\vartheta}$
.	.
41FF	x_{FF}

Table. 1

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Similarly, let the augend/minuend matrix $\mathbf{Y} = [\mathbf{y_{\mu\vartheta}}]_{\mathbf{16\times16}}$ of the same order and their elements are located in memory 4200_H to $42FF_H$. For the sum/difference matrix **S,** elements are located in memory 4300_H to $43FF_H$. Where,

$$\mathbf{S} = \mathbf{X} + \mathbf{Y} = [x_{\mu\vartheta} + y_{\mu\vartheta}]_{16\times16} \text{ (for addition operation).}$$

$$\mathbf{S} = \mathbf{Y} - \mathbf{X} = [y_{\mu\vartheta} - x_{\mu\vartheta}]_{16\times16} \text{ (for subtraction operation).}$$

If the result of operation exceeds the value FF_H in the case of addition or if the result of an operation is negative in the case of subtraction then the carry or borrow will be stored in a matrix $\mathbf{G} = [g_{\mu\vartheta}]_{16\times16}$ of same order respectively. The memory location allotted for this is from 4400_H to $44FF_H$.

ALGORITHM FOR ADDITION/SUBTRACTION

Step 1: Load the number 'n' in the E register. Where, $n = \{(\mu\times10_H) - 1_H\}$, n should be a hexadecimal number. μ is the number of rows of the matrix.

Step 2: Load the address of the 1st element of the addend/subtrahend matrix in the BC register pair (data pointer) pair.

Step 3: Copy the element specified by data pointer to the accumulator and then to H register.

Step 4: Move the data pointer to the first element of the augend/minuend matrix.

Step 5: Copy the element to accumulator and proceed with the addition/subtraction operation with the H register.

Step 6: Check if there is a carry/borrow. If there is no carry/borrow jump to Step 10.

Step 7: If carry/borrow produced, then store the result of the addition/subtraction by incrementing the data pointer to the sum/difference matrix **S**.

Step 8: Now clear the accumulator and increment it by one.

Step 9: Again increment the data pointer to the carry/borrow matrix **G** and store the corresponding Carry/borrow and return to sum/difference matrix **S**. and then jump to Step 11.

Step 10: Increment the data pointer to the sum/difference matrix **S** and store the result.

Step 11: Decrement the data pointer to the addend/subtrahend matrix **X**.

Step 12: Now increment the data pointer to the next element of the same matrix.

Step 13: Now decrement the value of 'n' in the E register.

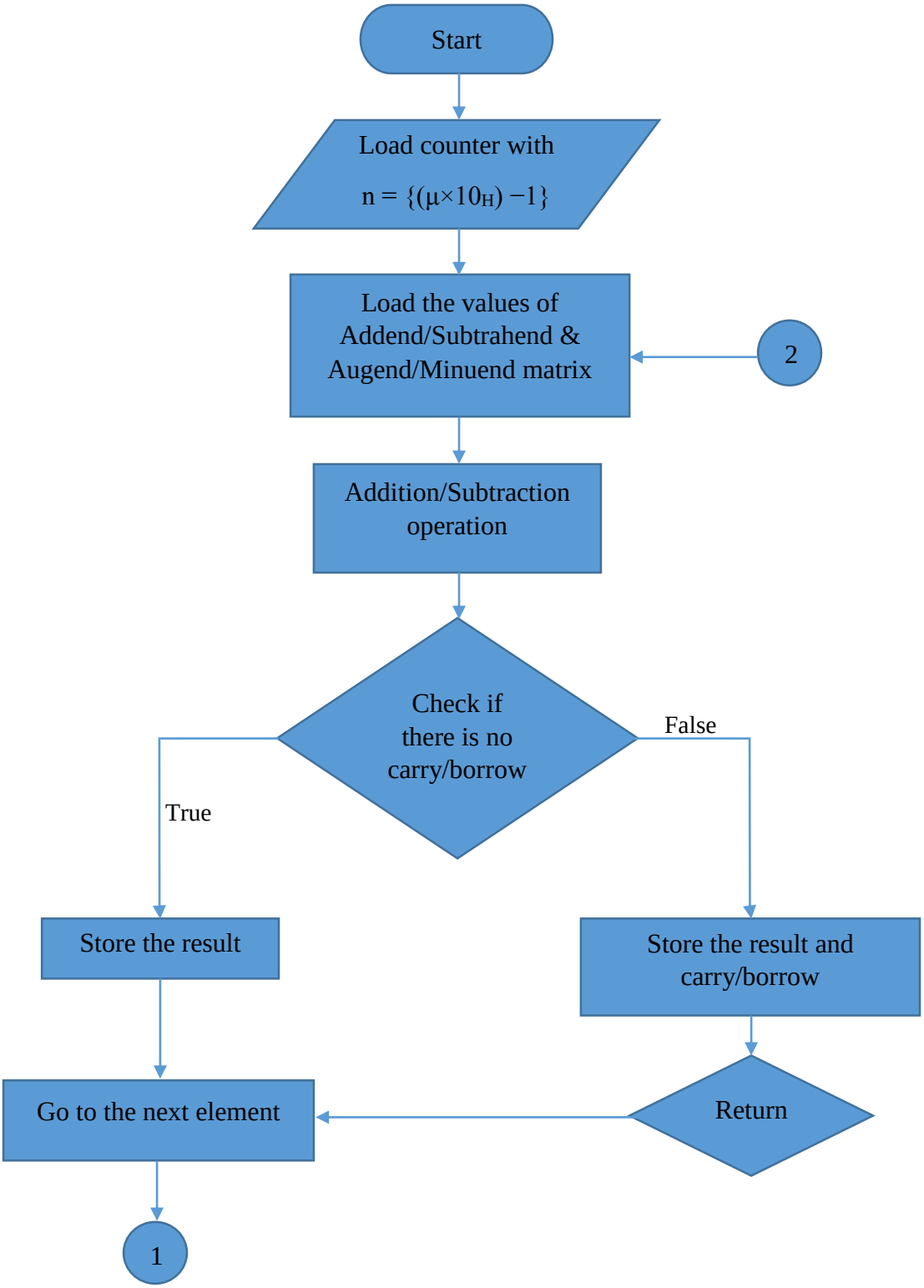
Step 14: Check if n is zero or non-zero. If it is a non-zero go to step 3 or else continue to next step.

Step 15: Once again repeat from Step 3 to Step 11.

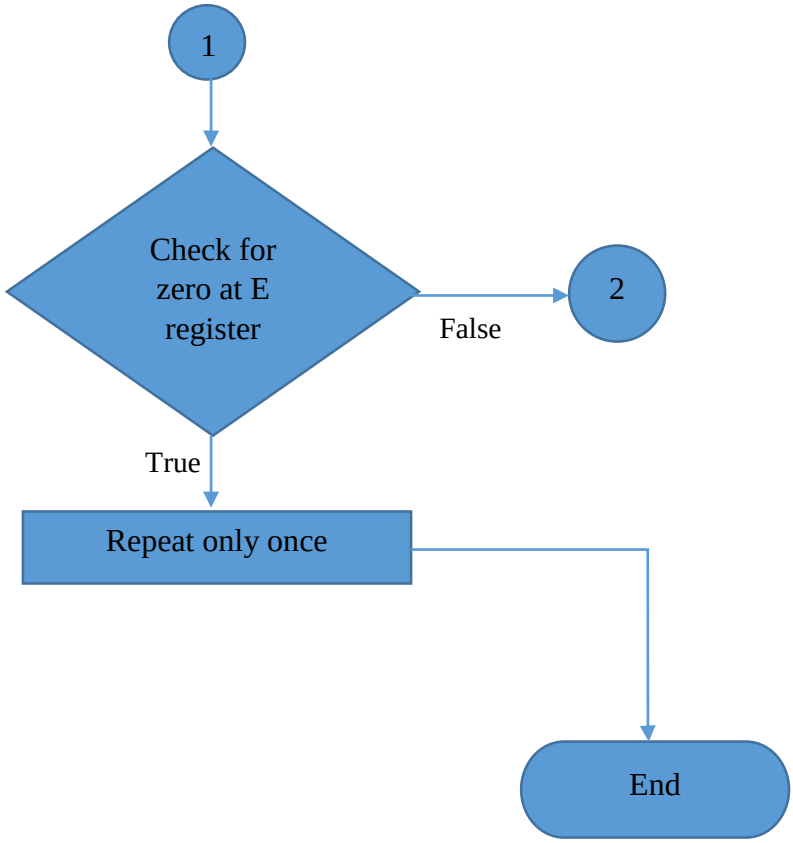
ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Step 16: End the program.

Flow chart:



ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)



CODE FOR ADDITION/SUBTRACTION

```
    MVI E,FF_H
    LXI B,4100_H
J3: LDAX B
    MOV H,A
    INR B
    LDAX B
    ADD H / SUB H
    JNC J1
    INR B
    STAX B
    INR B
    XRA A
    INR A
    STAX B
    DCR B
    JMP J2
J1: INR B
    STAX B
J2: DCR B
    DCR B
    INX B
    DCR E
    JNZ J3
    LDAX B
    MOV H,A
    INR B
```

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

```
        LDAX B          DCR B
        ADD H  / SUB H  JMP J5
        JNC J4          J4: INR B
        INR B           STAX B
        STAX B          J5: DCR B
        INR B           DCR B
        XRA A           HLT
        INR A
        STAX B
```

What if a matrix is other than (16×16)?

Whatever the type of matrix, the way of distributing the elements are the same as in matrix **X**. The only thing matters are the number of rows. The number of rows of the required matrix should be taken as μ (maximum 10_H) in the formula given in step 1 of the algorithm.

The program is verified using 8085 simulator v 1.0 (designed by J-Tech Software).

INPUT FOR ADDITION

For simplicity, Matrix **X** can be considered to have all the elements as 01_H. The simulation input of matrix **X** is shown in Fig. 2. And the matrix **Y** is considered to have 02_H as its elements except for the diagonal elements. The diagonal elements are loaded with FF_H to check whether we get a diagonal carry matrix. The simulation input of matrix **Y** is shown in Fig. 3.

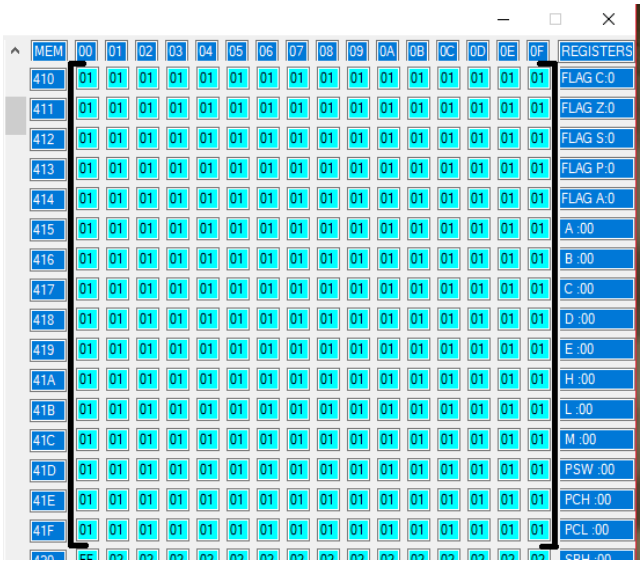


Fig. 2: Matrix X

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

The black colour square bracket is marked in the figure to indicate the matrix bracket. It was not present in the simulator.

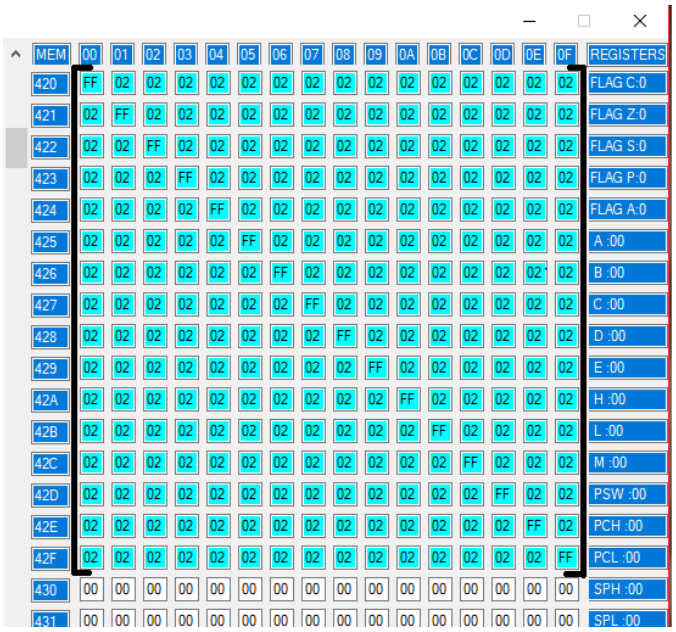


Fig. 3: Matrix Y

OUTPUT FOR ADDITION

The sum matrix **S** and the carry matrix **G** which we obtained are shown in Fig.4 and Fig.5 respectively. As we expect, the numbers on non-diagonal elements are 03_H. Since, $01_H + 02_H = 03_H$. And we obtained the carries at the diagonal as we expected. Since we have entered FF_H in the diagonals of augend matrix **Y**. Therefore $01_H + FF_H = 0100_H$. (01_H at carry matrix and 00_H at sum matrix).

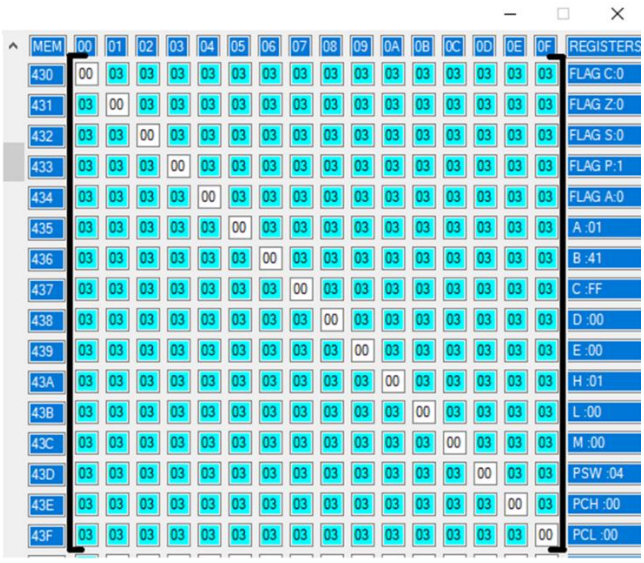


Fig. 4: Sum Matrix S

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

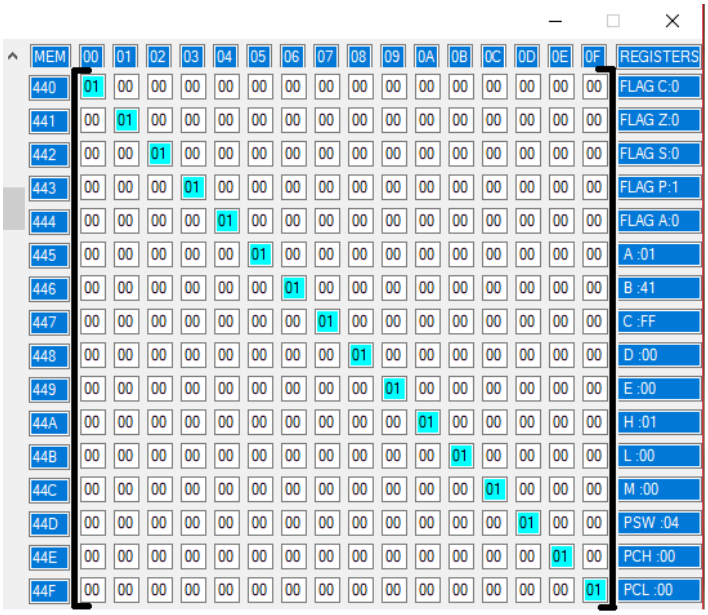


Fig. 4: Carry matrix G

INPUT FOR SUBTRACTION

Matrix **X** can be considered to have all the non-diagonal elements as 01_H and the diagonal elements as 02_H. The simulation input of matrix **X** is shown in Fig. 6. And the matrix **Y** is considered to have 02_H as its non-diagonal elements and the diagonal elements are loaded with 01_H to check whether we get a diagonal (negative) borrow matrix. But the difference should be still 01_H. The simulation input of matrix **Y** is shown in Fig. 7.

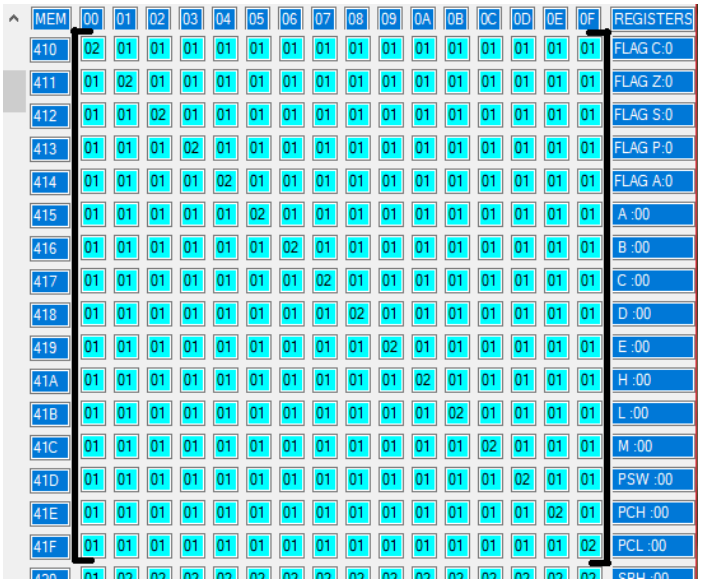


Fig. 5: Matrix X

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

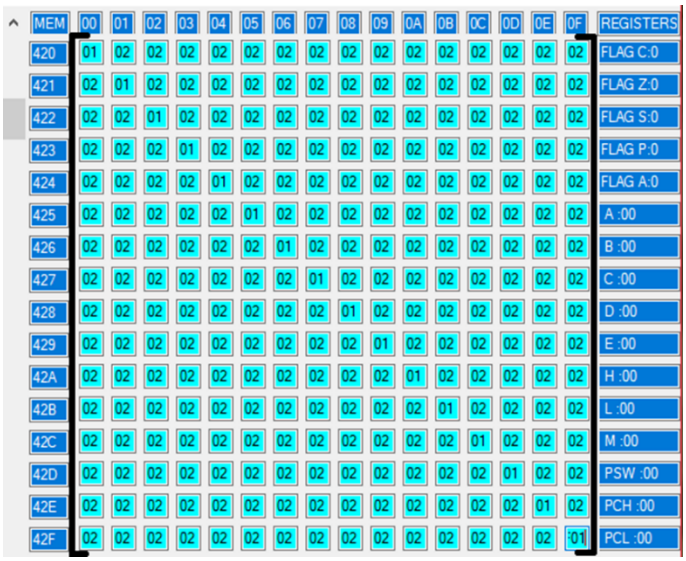


Fig. 6: Matrix Y

OUTPUT FOR SUBTRACTION

The difference matrix **S** and borrow matrix **G** which we obtain are shown in Fig. 8 and Fig. 9 respectively.

As we expect, all the elements of difference matrix **S** are 01_H. Since, the difference ($02_H \sim 01_H$) is 01_H. We obtained the borrow matrix **G** with diagonal elements 01_H, which indicates that the diagonal elements in difference matrix **S** are negative.

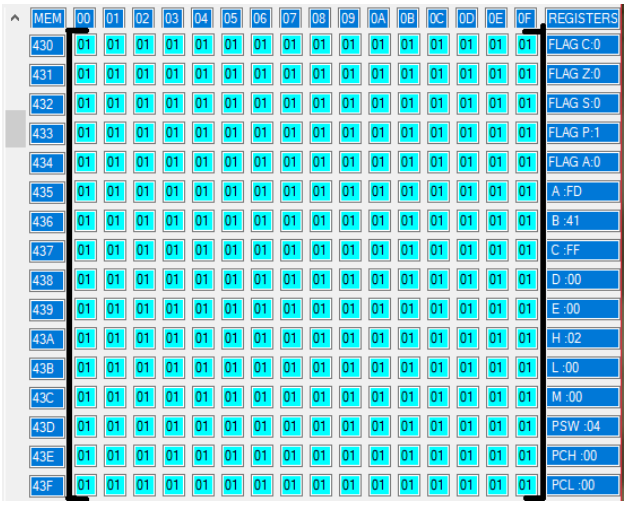


Fig. 7: Difference Matrix S

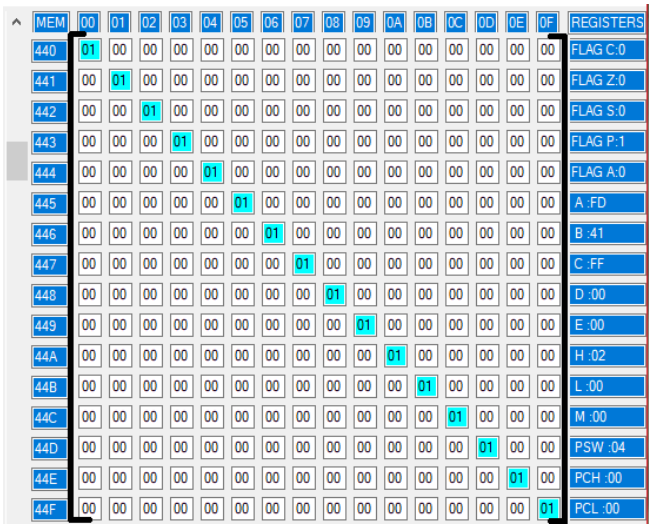


Fig. 8: Borrow Matrix

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

MATRIX MULTIPLICATION

Let $\mathbf{X} = [x_{\mu\vartheta}]_{m\times n}$ be the matrix of order (m×n) and $\mathbf{Y} = [y_{\mu\vartheta}]_{n\times p}$ be the matrix of order (n×p) then the product matrix **XY** will be the order of (m×p). The actual procedure to do matrix multiplication is row-by-column multiplication rule. In 8085 system the data pointer moves linearly. Therefore, we require a very large number of instructions to do multiplication. So we design the code to follow the row-by-row multiplication instead of doing actual procedure. We can get the correct result by entering $\mathbf{Y^T}$ instead of the **Y** matrix which is easy for the user to enter the data.

Doing with ALP:

Memory Mapping: The mapping procedure is similar to the previous case except for multiplicand matrix **Y**. In this case, we restrict our self that elements of the matrices are 4-bit hexadecimal numbers. So that the resultant element is within 16-bit.

Let us consider a multiplier matrix $\mathbf{X} = [x_{\mu\vartheta}]_{16\times16}$ whose 4-bit elements are located in memory 4100_H to $41FF_H$ and multiplicand transpose matrix $\mathbf{Y^T} = [y_{\mu\vartheta}]^T_{16\times16} = [y_{\vartheta\mu}]_{16\times16}$ whose 4-bit elements are located in memory 4200_H to $42FF_H$.

$$Y^T = \begin{bmatrix} y_{00} & y_{10} & y_{20} & \cdot & \cdot & \cdot & \cdot & y_{F0} \\ y_{01} & y_{11} & y_{21} & \cdot & \cdot & \cdot & \cdot & \cdot \\ y_{02} & y_{12} & y_{22} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & y_{\vartheta\mu} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ y_{0F} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & y_{FF} \end{bmatrix}$$

And the distribution of elements of $\mathbf{Y^T}$ is given in Table. 2.

Memory address	Data
4200	y_{00}
4201	y_{10}
4202	y_{20}
.	.
.	.
.	.
42μϑ	$y_{\vartheta\mu}$
.	.
.	.
42FF	y_{FF}

Table. 2

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

For the product matrix **XY**, elements are defined as,

$$XY = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdot & \cdot & \cdot & \cdot & p_{0F} \\ p_{10} & p_{11} & p_{12} & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{20} & p_{21} & p_{22} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & p_{\mu\vartheta} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{F0} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & p_{FF} \end{bmatrix}$$

Here, the resultant elements are obtained in memory locations described in the Table. 3.

Memory address	Data	Memory address	Data	Memory address	Data	…
4300	p_{00}	4400	p_{10}	4500	p_{20}	…
4310	p_{01}	4410	p_{11}	4510	p_{21}	…
4320	p_{02}	4420	p_{12}	4520	p_{22}	…
4330	p_{03}	4430	p_{13}	4530	p_{23}	…
.	.	.	.	.	.	•••
.	.	.	.	.	.	•••
.	.	.	.	.	.	•••
43ϑ0	$p_{0\vartheta}$	44ϑ0	$p_{1\vartheta}$	45ϑ0	$p_{2\vartheta}$	…
.	.	.	.	.	.	•••
.	.	.	.	.	.	•••
.	.	.	.	.	.	•••
43F0	p_{0F}	44F0	p_{1F}	45F0	p_{2F}	…

Table. 3

If the product element exceeds FF_H then the overlapping number is obtained in the next memory of the corresponding element.

Programmer defined extra-memories

Apart from the memory location for program code and input/output data, there should be a 6 extra-memory location where the description of the matrix should be entered by the user as given in the Table. 4.

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Memory address	Description	Value
5200	Storage counter (constant)	02_H
5201	No. of columns of matrix **X**	10_H
5202	No. of columns of product matrix minus one	$\boldsymbol{\beta_H} = \boldsymbol{\alpha_H} - 1$
5203	No. of columns of the product matrix	$\boldsymbol{\alpha_H}$
5300	Temporary storage (for the program itself)	Forbidden for user
5301	Temporary address storage (for the program itself)	Forbidden for user

Table. 4

ALGORITHM FOR MULTIPLICATION

Main program

Step 1: Load DE register pair with $40F0_H$ (starting address-10_H) of **X**.

Step 2: HL pair with 0010_H and then double add the DE and HL register pair.

Step 3: Exchange the DE (data pointer-1) and HL register pairs.

Step 4: Store the value of E register in the temporary address storage location.

Step 5: Load BC (data pointer-2) and go to the subroutine program.

Step 6: Recover the value of E register from temporary address storage and go to subroutine program.

Step 7: Load the value from address 5202 and decrement it by one. Store it in address 5202. Check the value ($\beta_H - 1$), if it is non-zero then go to step-6 or else continue to next step.

Step 8: Load the actual value of β_H in the memory address 5202.

Step 9: Increment the storage counter.

Step 10: Recover the value of E register from temporary address storage.

Step 11: Load the value from address 5203 and decrement it by one. Store it in address 5203. Check the value ($\alpha_H - 1$), if it is non-zero then go to step-2 or else continue to next step.

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Step 12: Now load E register with 'δ_H'.

Where $\delta_H = \{(10_H \times \omega_H) - (10_H - \omega_H)\}$ and ω_H is the number of rows of product matrix in hexadecimal.

Step 13: Load the HL pair with the 16-bit address which is obtained by adding 10_H with the address of the last element of the product matrix.

Step 14: Clear A and B register and load c register with 10_H.

Step 15: Decrement the HL pair and add memory with A register.

Step 16: If there is a carry, store the carry in the B register. And decrement C register.

Step 17: If the value in C is non-zero go to step-15. Else continue to next step.

Step 18: Store the sum in memory. Increment the HL pair and store the carry in memory and then decrement HL pair.

Step 19: Decrement the E register. If the value in E is non-zero go to step-14. Else continue to next step.

Step 20: Repeat from step-14 to step-18 once.

Step 21: End the program.

Subroutine program

Step 1: Load the content of the DE register pair.

Step 2: check if the value is zero. If it is zero, then jump to step-11. Else continue to the next step.

Step 3: Move the value to the L register.

Step 4: Load the content of the BC register pair.

Step 5: Move the value to H register.

Step 6: Multiply the content of H and L register by multiple additions.

Step 7: Store the result in temporary storage.

Step 8: load the storage counter value from address 5200 to A and decrement it. Increment the B register by the value in A register.

Step 9: Recover the result from temporary storage and store the result in the area denoted by BC.

Step 10: Decrement the BC to previous value by using storage counter.

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Step 11: Increment the BC and DE by one.

Step 12: Load the value of columns in A from address 5201 and decrement it by one.

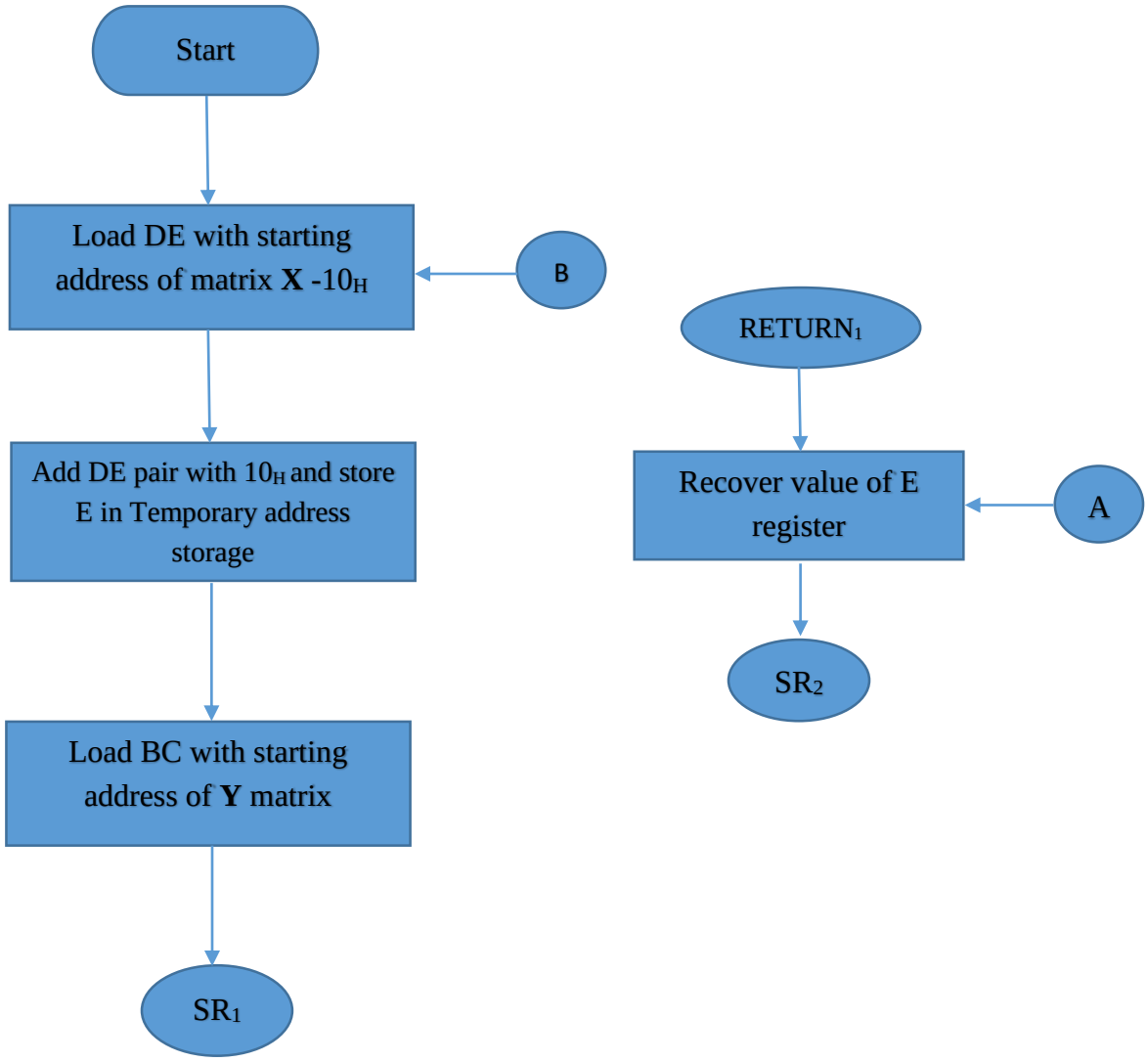
Step 13: Check for non-zero. If it is non-zero then go to step-1 or else continue to next step.

Step 14: Store the actual number of columns in address 5201.

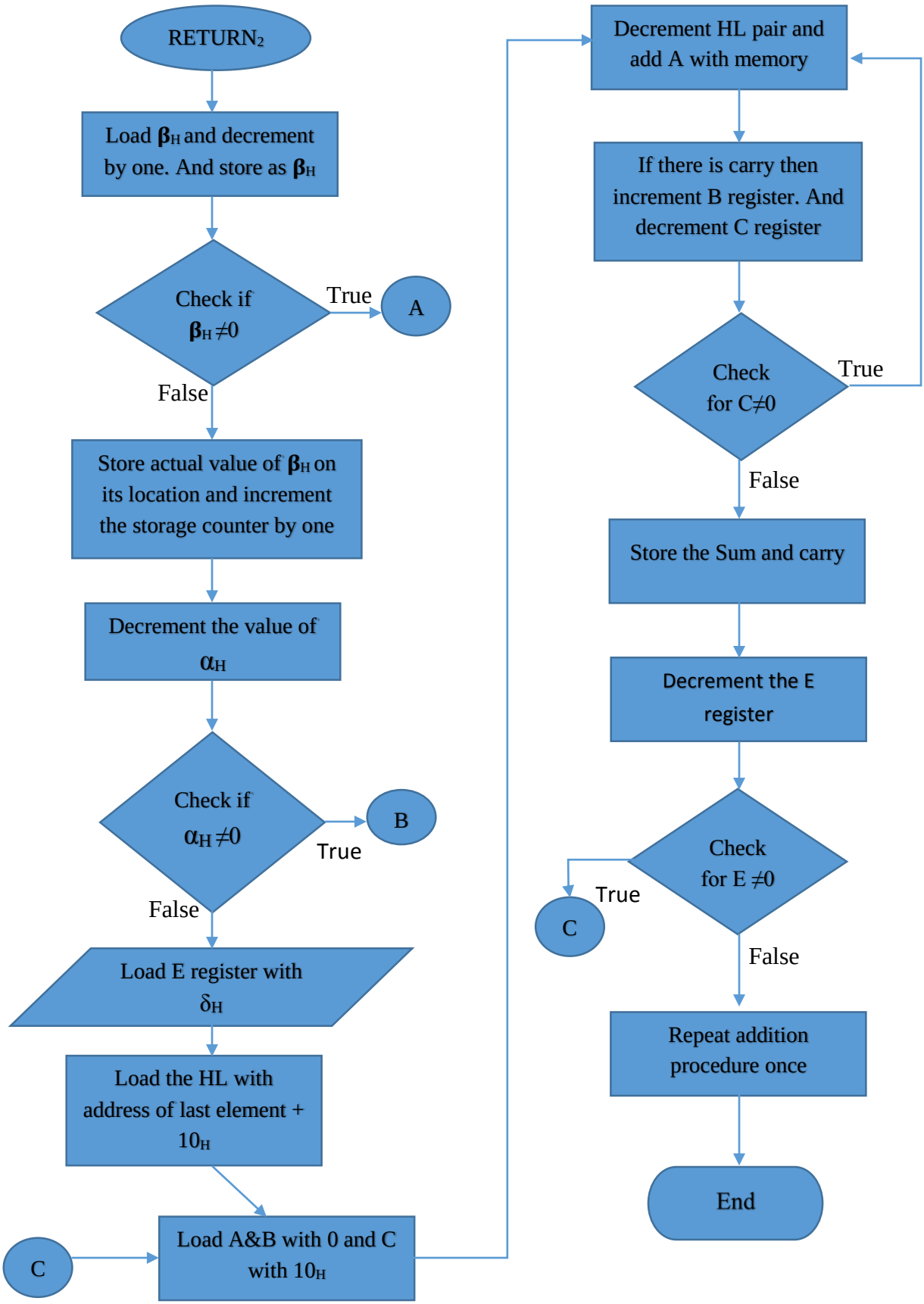
Step 15: Return to the main program.

FLOW CHART

Main program:

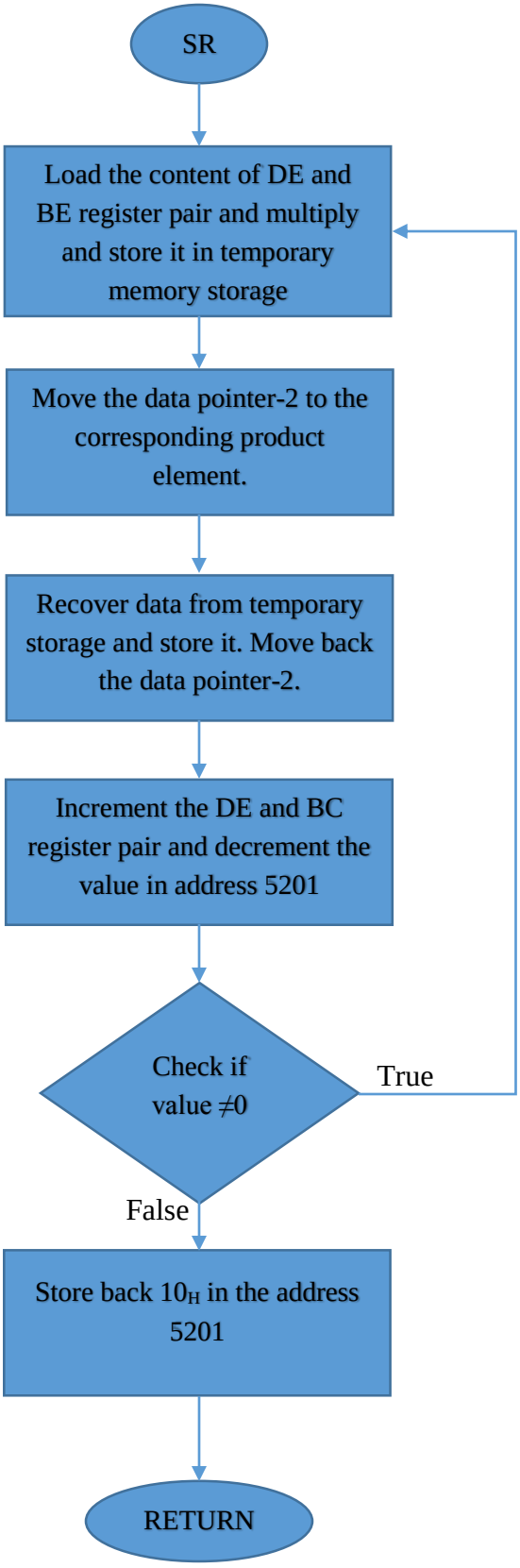


ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)



ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Subroutine program:



ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

CODING FOR MULTIPLICATION.

Main program

```
    LXI D,40F0H
J2: LXI H,0010H
    DAD D
    XCHG
    MOV A,E
    STA 5301H
    LXI B,4200H
    CALL S1
J1: LDA 5301H
    MOV E,A
    CALL S1
    LDA 5202H
    DCR A
    STA 5202H
    JNZ J1
    MVI A,02H
    STA 5202H
    LDA 5200H
    INR A
    STA 5200H
    LDA 5301H
    MOV E,A
    LDA 5203H
    DCR A
    STA 5203H
    JNZ J2
    MVI E,23
    LXI H,4530H
J5: XRA A
    MOV B,A
    MVI C,10H
J4: DCX H
    ADD M
    JNC J3
    INR B
J3: DCR C
    JNZ J4
    MOV M,A
    INX H
    MOV M,B
    DCX H
    DCR E
    JNZ J5
    XRA A
    MOV B,A
    MVI C,10H
J7: DCX H
```

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

```
    ADD M
    JNC J6
    INR B
J6: DCR C
    JNZ J7
    MOV M,A
    INX H
    MOV M,B
    DCX H
    HLT
```

Subroutine program

```
S1: LDAX D
    INR A
    DCR A
    JZ L1
    MOV L,A
    LDAX B
    MOV H,A
    XRA A
L2: ADD H
    DCR L
    JNZ L2
    STA 5300H
    LDA 5200H
L4: DCR A
    JZ L3
    INR B
    JMP L4
L3: LDA 5300H
    STAX B
    LDA 5200H
L5: DCR A
    JZ L1
    DCR B
    JMP L5
L1: INX B
    INX D
    LDA 5201H
    DCR A
    STA 5201H
    JNZ S1
    MVI A,10H
    STA 5201H
    RET
```

INPUT

For simplicity we can consider multiplier matrix **X** of order (3×16) and multiplicand matrix **Y** of order (16×3) with all the element as 01_H, In order to get a product matrix **XY** of order (3×3).

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

The program code given above written for this dimension by using conditions mentioned in Table. 4 and main algorithm step 12&13. Therefore the values of the parameters are:

$\alpha_H = 03_H$

$\boldsymbol{\beta}_H = 02_H$

$\omega_H = 03_H$

$\delta_H = 23_H$

But still, the program is capable of performing multiplication of maximum limit of (16×16) by changing the limits in Table 4 and the main algorithm step 12&13.

The reason we have chosen this kind of input matrix **X** and **Y** is to get the output as sooner as possible. If we compute the multiplication operation for maximum dimensional limit, the simulator will take a larger amount of time.

The simulation input of Matrix **X** is given in Fig. 10.

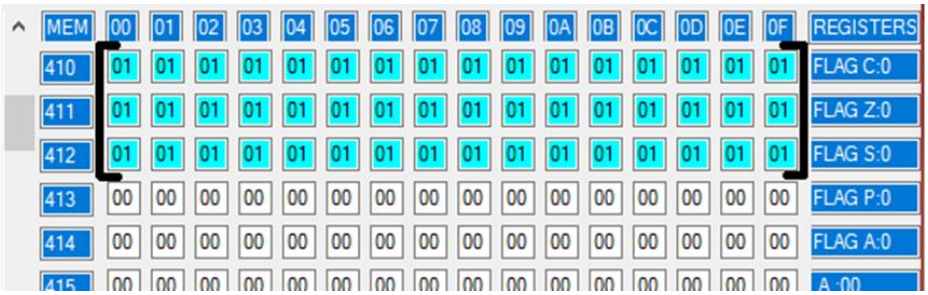


Fig. 9: Matrix X

The simulation input of Matrix $\mathbf{Y}^{\mathbf{T}}$ is given in Fig. 11.

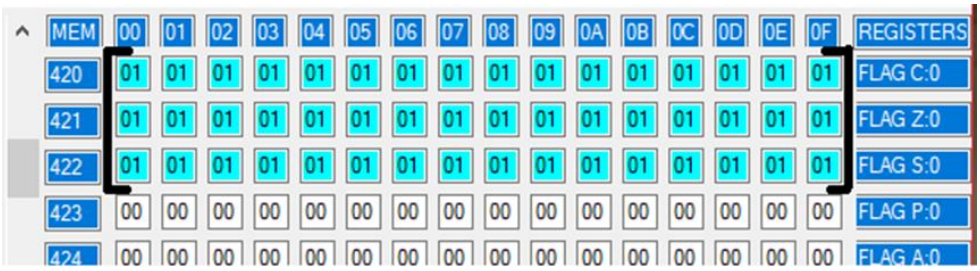


Fig. 10: Matrix Y^T

OUTPUT

The simulation output we obtained is shown in Fig.12, Fig.13, and Fig.14. Each Figure represents the corresponding row of the product matrix.

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

Fig. 11

Fig. 12

Fig. 13

Now we can use the Table. 3 to figure out the elements from the resultant values of the memories. The black-coloured box is marked to indicate the required memory area for the product matrix. It was not present in the simulator. The second (white) column inside the box indicates the overlapping value. It will be the first two digits of the corresponding element of the product matrix. By using the Table. 3 we can figure out the product matrix as

$$XY = \begin{bmatrix} 0010_H & 0010_H & 0010_H \\ 0010_H & 0010_H & 0010_H \\ 0010_H & 0010_H & 0010_H \end{bmatrix}$$

Therefore we successfully got the correct result.

ADVANTAGE OF THIS HARDWARE COMPUTING

- It is easy to add on the memory size, therefore we can do computations on n-dimensional matrices.

ISSN 2688-8300 (Print) ISSN 2644-3368 (Online)

- The size of the number we computing here is 8-bit, but it is physically easy to increase the size of the data bus.
- We can include decimal points and negative sign by designing a special flip-flop in each register and with few changes in ALU.
- Parallel computing can be done easily with fewer interfaces.

CONCLUSION

The Assembly language program for computing addition, subtraction and multiplication on higher dimensional matrix has been successfully designed. The possible operational limit of this algorithm is from (2×2) to (16×16) matrices. The time taken to complete execution is very long in actual 8085 microprocessor since we have small clock frequency oscillator. If we apply the same algorithm to the latest hardware systems we may get the result quickly.

REFERENCES:

1. V Vijayendran. "Fundamentals of Microprocessor-8085 Architecture Programming & Interfacing", S. Viswanathan Printers & Publishers Pvt. Ltd., Chennai, India (2009)
2. 8080/8085 Assembly Language Programming Manual Copyright © 1977, 1978 Intel Corporation, California, USA