

# Microprocessor Lab Report

Abhiroop Mukherjee  
Enrl. No: 510519109



Department of Computer Science and Technology  
Indian Institute of Engineering Science and Technology, Shibpur

## Contents

|    |   |    |
|----|---|----|
| 1  | Find out the sum of the first 30 natural numbers          | 1  |
| 2  | Find minimum and maximum number in 10-byte unsigned array | 2  |
| 3  | Delay Procedure   | 3  |
| 4  | Move block of data from location X to location Y          | 4  |
| 5  | Check if number odd                                       | 6  |
| 6  | Multi-Byte Addition                                       | 7  |
| 7  | Fast Multiplication Subroutine                            | 9  |
| 8  | Sort Subroutine   | 10 |
| 9  | Subroutine to save Register Status                        | 12 |
| 10 | POST to check stuck at 1 fault                            | 14 |
| 11 | Binary Search   | 16 |
| 12 | Time Period of a rectangular waveform signal              | 18 |

# 1 Assignment 1

## 1.1 Objective

Find out the sum of the first 30 natural numbers.

## 1.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

## 1.3 Procedure

We know that

$$1 + 2 + 3 + \dots + 29 + 30 = \frac{30 \times 29}{2} = 435 = 01D1H$$

This result is not possible to store in a single register, so we need to use register pair to store the result.

## 1.4 Program

```
1 ;end result is 465, more than 255 hence we need to do extended additions
2     MVI D,1E; setup D, the counter as 30
3     MVI C,01; setup BC as 1
4
5 L1:   DAD B; Double add BC to HL
6       INX B; extended increment BC
7       DCR D; decrement D
8       JNZ L1; if D becomes 0, Z flag becomes 0 and we break
9       HLT
10    ; Ans will be in HL
```

Listing 1: assembly program to find sum of the first 30 natural numbers

## 1.5 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register B  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register C  | 1F    | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register L  | D1    | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| Memory(M)   | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 1: Register configuration after execution (observe HL)

## 1.6 Conclusion

We see that after the execution of program, the data stored in HL register pair is 01D1H, which is the hexadecimal value of 435.

Hence the program is working as expected.

## 2 Assignment 2

### 2.1 Objective

From an array of 10-byte size integers (unsigned) find out the maximum and minimum.

### 2.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 2.3 Procedure

The idea is to linearly iterate through all the values of the arr, and update the register for minimum(C) and maximum(B) values.

### 2.4 Program

```
1 ;Actual Program
2 # ORG 5000H
3 # ARR: DB 5,2,3,4,F,C,7,A,B,1
4 # ORG 0000
5     LXI H,ARR
6     MOV B,M ;B is maximum val
7     MOV C,M ;C is minimum val
8     MVI D,0A
9 ;CMP R does (A - R) in background
10 ;If A - R > 0 then Cy = 0, Z = 0
11 ;If A - R = 0 then Cy = 0, Z = 1
12 ;If A - R < 0 then Cy = 1, Z = 1
13
14 LP:  MOV A,M
15     CMP B
16     JC MIN ;will Jump when Cy = 1, A - B < 0, A < B
17     MOV B,A ;will only happen if A > B
18
19 MIN:  CMP C
20     JNC SKIP ;will Jump when Cy = 0, A - C > 0, A > C
21     MOV C,A ;will only happen if A < C
22
23 SKIP: INX H
24     DCR D
25     JNZ LP
26     HLT
```

Listing 2: assembly program to find minimum and maximum number in 10-byte unsigned array

### 2.5 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register B  | 0F    | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Register C  | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 50    | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Register L  | 0A    | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| Memory(M)   | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 06    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Register B  | 0F    | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Register C  | 06    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 50    | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Register L  | 0A    | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| Memory(M)   | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(a) result for {5,2,3,4,F,C,7,A,B,1}

(b) result for {F,E,D,C,B,A,9,8,7,6}

Figure 2: Result for different inputs

### 2.6 Conclusion

We see that that after program execution, B has the maximum value of array, and C has the minimum value of the array.

Hence the program is working as expected.

## 3 Assignment 3

### 3.1 Objective

Write a routine that produces a delay. The delay value must be passed to register pair DE.

### 3.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 3.3 Procedure

Idea is to assign DE a very big value (say FFFF), and decrement it in a loop till DE becomes 0 to produce delay in execution.

### 3.4 Program

```
1      LXI D,FFFF
2      CALL DELAY
3      HLT
4      ;delay: this subroutine produces delay
5      ;in: value in DE pair
6      ;out: none
7      ;destroys: A
8
9      DELAY: DCX D      ;doesn't affect any flags, that's why doing OR
10         MOV A,E
11         ORA D      ;will give 0 only when both D and E 00
12         JNZ DELAY
13         RET
```

Listing 3: assembly program to produce delay

### 3.5 Conclusion

We see that the code runs for sometime, then completes its execution, signifying that the delay function worked and delayed execution of CPU for some time.

Hence the program is working as expected.

## 4 Assignment 4

### 4.1 Objective

Write a subroutine to move a block of bytes from location X to location Y.  
Note that the caller would specify

- X, the source address
- Y, the destination address
- Z, the block size

Note that X, Y and Z are 16-bit quantities.

### 4.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 4.3 Procedure

Start reading numbers from location X and save them to location Y, after each iteration, update address of X and Y to next byte. Do this Z times and the whole block is copied.

### 4.4 Program

```
1 ;setup of data
2 #ORG 2500
3 #ARR: DB 4,2,6,7,8
4
5 # DESTLOC EQU 4500
6 #ORG 0000
7 ;let BC = 5, X = 2500, Y = 4500
8 ;hence add data from 2500 to 2504
9     LXI B,0005
10    LXI D, ARR
11    LXI H,DESTLOC
12    CALL MOVE
13    HLT
14 ;MOVE: move Z number of bytes from loc (X to X+Z) to loc (Y to Y+Z)
15 ;Z store in BC
16 ;X store in DE
17 ;Y store in HL
18
19 MOVE: LDAX D    ;A = Mem[DE]
20       MOV M,A  ;Mem[HL] = A
21       INX H    ;HL++
22       INX D    ;DE++
23       DCX B    ;BC--
24 ;DCX doesn't set flags so do manual check by OR
25       MOV A,B
26       ORA C
27       JNZ MOVE
28       RET
```

Listing 4: assembly program to move block

## 4.5 Experimentation

| Memory Address | Value |
|----------------|-------|
| 0000           | 01    |
| 0001           | 05    |
| 0003           | 11    |
| 0005           | 25    |
| 0006           | 21    |
| 0008           | 45    |
| 0009           | CD    |
| 000A           | 0D    |
| 000C           | 76    |
| 000D           | 1A    |
| 000E           | 77    |
| 000F           | 23    |
| 0010           | 13    |
| 0011           | 0B    |
| 0012           | 78    |
| 0013           | B1    |
| 0014           | C2    |
| 0015           | 0D    |
| 0017           | C9    |
| 2500           | 04    |
| 2501           | 02    |
| 2502           | 06    |
| 2503           | 07    |
| 2504           | 08    |
| 4500           | 04    |
| 4501           | 02    |
| 4502           | 06    |
| 4503           | 07    |
| 4504           | 08    |
| FFFE           | 0C    |

  

| Memory Address | Value |
|----------------|-------|
| 0000           | 01    |
| 0001           | 0A    |
| 0003           | 11    |
| 0005           | 25    |
| 0006           | 21    |
| 0008           | 45    |
| 0009           | CD    |
| 000A           | 0D    |
| 000C           | 76    |
| 000D           | 1A    |
| 000E           | 77    |
| 000F           | 23    |
| 0010           | 13    |
| 0011           | 0B    |
| 0012           | 78    |
| 0013           | B1    |
| 0014           | C2    |
| 0015           | 0D    |
| 0017           | C9    |
| 2500           | 04    |
| 2501           | 02    |
| 2502           | 06    |
| 2503           | 07    |
| 2504           | 08    |
| 2505           | 01    |
| 2506           | 02    |
| 2507           | 04    |
| 2508           | 05    |
| 2509           | 06    |
| 4500           | 04    |
| 4501           | 02    |
| 4502           | 06    |
| 4503           | 07    |
| 4504           | 08    |
| 4505           | 01    |
| 4506           | 02    |
| 4507           | 04    |
| 4508           | 05    |
| 4509           | 06    |
| FFFE           | 0C    |

(a) result for {4,2,6,7,8}

(b) result for {4,2,6,7,8,1,2,4,5,6}

Figure 3: Result for different inputs

## 4.6 Conclusion

We see that all the data from location 2500(X) to (2500 + Z) has been copied to location 4500(Y) to (4500 + Z) [Z is 5 in 3a and 10 in 3b].

Hence the program is working as expected.

## 5 Assignment 5

### 5.1 Objective

Write a function isODD(unsigned n) in assembly that takes an unsigned integer (a byte) and determines if it is odd (returns 1) or 0 if it is even.

### 5.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 5.3 Procedure

Odd numbers will always be in the form of  $2x + 1$ , which means that they will have 1 as their LSB. So we just check if  $number \wedge 01$  is 1 or not. If the result is 1, then the number is odd, else it is even.

### 5.4 Program

```
1 # NUM EQU 49 ;73 in hex
2     MVI B,NUM
3     CALL ISODD
4     HLT
5 ;isODD(n): function which tell if n is odd or even
6 ;in: n = B
7 ;out: ans in C, if n even C = 0,else 1
8 ;destroys: A
9 ;idea: X AND 01 = 0 if X is even, 1 is X is Odd
10
11 ISODD: MOV A,B
12     ANI 01 ;A = A and 01
13     MOV C,A ;store result in C
14     RET
```

Listing 5: assembly program to check if given number isOdd or not

### 5.5 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register B  | 49    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| Register C  | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register L  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Memory(M)   | 06    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register B  | 4A    | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| Register C  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register L  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Memory(M)   | 06    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

(a) result for odd number(73)

(b) result for even number(73)

Figure 4: Result for both even and odd numbers

### 5.6 Conclusion

We see that in case of odd number, C is 1 after execution and in case of even number, C is 0 after execution.

Hence the program is working as expected.



## 6 Assignment 6

### 6.1 Objective

Write a function to add two multi-byte numbers stored in location X and Y. The result is stored in X. Pass a parameter Z indicating the no. of bytes to be added.

### 6.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 6.3 Procedure

We simulate the default way of adding numbers, we go from right to left, adding (with carry) the numbers and adding it to stack, then we keep popping the elements and save it in X.

### 6.4 Program

```
1  # ORG 8500
2  # NUM1: DB FF,FF,FF
3  # ORG 9000
4  # NUM2: DB FF,FF,FF
5  # NUMBYTE EQU 3
6  # ORG 0000
7      MVI C,NUMBYTE      ;C <- 03
8      LXI H,NUM1         ;HL <- 8500
9      LXI D,NUM2         ;DE <- 9000
10     CALL ADDDB
11     HLT
12 ;add Z bytes, first number starts from X, other number starts from Y
13 ;Z->C
14 ;X->HL
15 ;Y->DE
16 ;destroys -> B
17 # SAVELOC EQU 1500
18 ADDB: SHLD SAVELOC ;save start of X to 1500 addr to be read later
19     MVI B,00        ;set B as 0, will act as counter afterward
20 ;now we need to put HL and DE pair to back of array
21     XCHG            ;swap HL and DE
22     DAD B           ;HL = HL + BC [with B = 0] [HL now is DE]
23     DCX H
24     XCHG            ;swap HL and DE again
25     DAD B           ;HL = HL + BC
26     DCX H
27 ;now both DE and HL are in the end of their array
28 LOOP: LDAX D        ;A -> Mem[DE]
29     ADC M           ;A -> A + Mem[HL] + carry
30     PUSH PSW        ;push AF data to SP, first A, then F
31     INR B
32     DCX H
33     DCX D
34     DCR C
35     JNZ LOOP        ;jump to Loop till C!=0
36 ;now addition done and saved to Stack, need to check if carry exist
37     JNC SKIP
38 ;these will execute only when there is a carry
39     MVI A,01
40     PUSH PSW
41     INR B
42
43 SKIP: LHLD SAVELOC   ;read saved LH data to go to start of X
44
45 L1:   POP PSW        ;pop SP to AF, first F then A
46     MOV M,A
47     INX H
48     DCR B
49     JNZ L1
50     RET
```

Listing 6: assembly program to add multi-byte numbers

## 6.5 Experimentation

| Memory Address | Value |
|----------------|-------|
| 001F           | 17    |
| 0021           | D2    |
| 0022           | 28    |
| 0024           | 3E    |
| 0025           | 01    |
| 0026           | F5    |
| 0027           | 04    |
| 0028           | 2A    |
| 002A           | 15    |
| 002B           | F1    |
| 002C           | 77    |
| 002D           | 23    |
| 002E           | 05    |
| 002F           | C2    |
| 0030           | 2B    |
| 0032           | C9    |
| 1501           | 85    |
| 8500           | 01    |
| 8501           | FF    |
| 8502           | FF    |
| 8503           | FE    |
| 9000           | FF    |
| 9001           | FF    |
| 9002           | FF    |
| FFF6           | 55    |
| FFF7           | 01    |
| FFF8           | 85    |
| FFF9           | FF    |
| FFFA           | 85    |
| FFFB           | FF    |
| FFFC           | 91    |
| FFFD           | FE    |
| FFFE           | 0B    |

  

| Memory Address | Value |
|----------------|-------|
| 0027           | 04    |
| 0028           | 2A    |
| 002A           | 15    |
| 002B           | F1    |
| 002C           | 77    |
| 002D           | 23    |
| 002E           | 05    |
| 002F           | C2    |
| 0030           | 2B    |
| 0032           | C9    |
| 1501           | 85    |
| 8500           | 01    |
| 8501           | 01    |
| 8503           | FF    |
| 8504           | FE    |
| 9000           | FF    |
| 9001           | FE    |
| 9002           | FC    |
| 9003           | FA    |
| FFF4           | 55    |
| FFF5           | 01    |
| FFF6           | 11    |
| FFF7           | 01    |
| FFF8           | 55    |
| FFFA           | 84    |
| FFFB           | FF    |
| FFFC           | 80    |
| FFFD           | FE    |
| FFFE           | 0B    |

(a) result for {FF,FF,FF} + {FF,FF,FF}

(b) result for {01,02,03,04} + {FF,FE,FC,FA}

Figure 5: Result of multi-byte addition (start looking from address 8500)

## 6.6 Conclusion

We see that the result of multi-byte addition is correct.  
Hence the program is working as expected.

## 7 Assignment 7

### 7.1 Objective

Write a fast sub-routine to multiply 9 by 15.

### 7.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 7.3 Procedure

We use the Shift-and-Add Multiplication to fast multiply 15 and 9, as register size is 8 bits, we can do this multiplication by using a loop which runs 8 times, i.e  $O(1)$

This method is faster than default loop method, which run in  $O(\min(m, n))$ , where  $m$  and  $n$  are the numbers that will be multiplied.

### 7.4 Program

```
1      LXI D,000F      ;15 in hex
2      MVI A,09
3      LXI H,0000      ;result will be in HL
4      MVI B,08        ;8 bit data, 8 rotations to iterate through all the bits of A
5
6 LP:   DAD H          ;HL = HL + HL (multiply by 2)(assume left shift)
7      RAL            ;rotate A left(<-), leftmost value in C flag
8      JNC SKIP
9      DAD D          ;if 1 in A's bit, we add D also
10  SKIP: DCR B
11      JNZ LP
12      SHLD 2500
13      HLT
14 ;ans in HL register
```

Listing 7: assembly program to fast multiply 15 times 9

### 7.5 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 04    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Register B  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register C  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 0F    | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Register H  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register L  | 87    | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| Memory(M)   | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 6: Register configuration after execution (look at HL)

### 7.6 Conclusion

$$9 \times 15 = 135 = 0087H$$

We see that the result in HL register is same as what we expected.

Hence the program is working as expected.

## 8 Assignment 8

### 8.1 Objective

Write a subroutine to sort a 5-element byte array (Any algorithm will do)

### 8.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 8.3 Procedure

We use bubble sort algorithm to sort the array.

### 8.4 Program

```
1  # ORG 2500
2  # ARR: DB 4,2,5,6,7
3  # LEN EQU 5
4  # ORG 0000
5
6  START:      LXI H,ARR      ;start of array
7              MVI C,LEN
8              DCR C          ;c-- as we will only go through first 4
9              MVI D,00       ;d acts a bool, if any swap happend, d = 1
10
11 CHECK:      MOV A,M
12              INX H
13              CMP M          ;compare a[i] with a[i+1]
14 ;if a[i] - a[i+1] > 0 -> Cy = 0, Z = 0 (we swap)
15 ;if a[i] - a[i+1] = 0 -> Cy = 0, Z = 1 (we don't swap)
16 ;if a[i] - a[i+1] < 0 -> Cy = 1, Z = 0 (we don't swap)
17              JC NEXTBYTE
18              JZ NEXTBYTE
19 ;here swap occurs
20              MOV B,M ;a[i] = A, a[i+1] = B
21              MOV M,A ;a[i+1] = A
22              DCX H
23              MOV M,B ;a[i] = B
24              INX H
25              MVI D,01       ;set flag that swap occured
26
27 NEXTBYTE:   DCR C
28              JNZ CHECK
29              MOV A,D
30              CPI 01          ;comapare immediate A and 01
31 ;if A - 01 > 0 -> Cy = 0, Z = 0 -> A(D) is zero, no swap taken place, exit
32 ;if A - 01 = 0 -> Cy = 0, Z = 1 -> A or D is 1, swap taken place redo
33              JZ START
34              HLT
```

Listing 8: assembly program to bubble sort array

## 8.5 Experimentation

[illegible]

Figure 7: Register configuration after execution (look from address 2500)

## 8.6 Conclusion

We see that after program execution, values of address 2500 - 2504 is sorted in ascending order. Hence the program is working as expected.

## 9 Assignment 9

### 9.1 Objective

Write a sub-routine to store all the registers (A, F, B, C, D, E, H, L, I, SPL, SPH, PCL, PC, in that order) starting from location MYREGISTERS.

### 9.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 9.3 Program

```
1      MVI A,10      ;set A
2      LXI B,5092    ;set BC
3      LXI D,2794    ;set DE
4      LXI H,3792    ;set HL
5      SIM          ;set inturrept mask as value of A
6      LXI SP,F001
7      CALL DEBUG
8      HLT
9
10     # MYREGISTERS EQU 2000H
11     DEBUG: PUSH H
12           PUSH D
13           PUSH B
14           PUSH PSW
15           LXI H,MYREGISTERS
16           POP D
17           CALL STORE      ;store AF
18           POP D
19           CALL STORE      ;store BC
20           POP D
21           CALL STORE      ;store DE
22           POP D
23           CALL STORE      ;store HL
24           RIM
25           MOV M,A          ;store I
26           INX H
27           XCHG             ;swap HL and DE
28           LXI H,0000
29           DAD SP           ;now HL <- SP
30           XCHG             ;now HL-> save addr, DE<- SP
31     ;problem, SP also has this function call stuff, so we need to remove it's info from DE
32     ;(DE = DE + 2 [remember SP stack works reverse that's why +])
33           INX D
34           INX D
35     ;now store DE
36           CALL STORE
37     ;now we need to store PC, which will be in Stack due to function call
38           XCHG             ;swap HL and DE to store HL stuff in DE
39           XTHL             ;get PC data from SP (stack mem has garbage now)
40           XCHG             ;now HL has save addr, DE has data
41           CALL STORE
42           XCHG             ;PC data back in HL
43           XTHL             ;PC data back in stack
44           XCHG             ;save addr back in HL
45           RET
46
47     ;Procedure STORE
48     ;stores data in DE to memory whose address is in HL
49     STORE: MOV M,D
50           INX H
51           MOV M,E
52           INX H
53           RET
```

Listing 9: assembly program to store register configuration

## 9.4 Experimentation

[illegible]

Figure 8: Register configuration after execution (look from address 2000)

## 9.5 Conclusion

We see that after program execution, all the predefined value of registers are stored in memory starting from address 2000.

Hence the program is working as expected.

## 10 Assignment 10

### 10.1 Objective

Implement a POST or power-on-self-test where each RAM location is tested for stuck-at-zero or stuck-at-one fault. In your case the function takes the start address of the RAM block and the block size in bytes. The function sets CY in case of any error (else it is set to 0); HL contains the faulty location and Acc contains 0 for stuck at zero fault and 1 for stuck at one fault. [Note: usually there wont be any error as your RAM is not faulty, so direct checking may not set CY flag]

### 10.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 10.3 Procedure

We iteratively go through STARTLOC to (STARTLOC + LEN) and check if any value is 1 or not, if it is, we exit from there setting the C flag.

### 10.4 Program

```
1  #STARTLOC EQU 6400
2  #LEN EQU 05
3  ;let position 6403 have stuck at one fault
4      LXI H,6403
5      MVI M,01
6  ;let we start out search from 6400 an search 05 addresses
7      LXI H,STARTLOC
8      MVI B,LEN
9      CALL POST
10     HLT
11
12 POST:  MOV A,M
13        CPI 01
14 ;A will be 0 or 1, if 1 we need to exit
15 ;A zero -> A - 1 < 0, Cy =1
16 ;A one -> A - 1 = 0, Z = 1, we exit here
17     JZ ERR ;if 1 found, we exit are set Cy 1 directly
18     INX H
19     DCR B
20     JNZ POST
21 ;we will reach here only if there is no 1 found, so we need to set Cy 0
22     JC REVERT ;if Cy 1 , we revert, else we direct return
23     RET
24 ;we reach here when 1 found
25
26 ERR:   STC ;set Cy 1
27     RET
28 ;we reach here if no 1 found but Cy 1 due to previous CMP stuff
29
30 REVERT: CMC ;complements Cy, if Cy 1, Cy becomes 0
31     RET
```

Listing 10: assembly program for POST for stuck at 1 fault



## 10.5 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Register B  | 02    | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Register C  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register H  | 64    | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| Register L  | 03    | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Memory(M)   | 01    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

  

| Resister      | Value | S | Z | * | AC | * | P | * | CY |
|---------------|-------|---|---|---|----|---|---|---|----|
| Flag Resister | 55    | 0 | 1 | 0 | 1  | 0 | 1 | 0 | 1  |

  

| Type                     | Value |
|--------------------------|-------|
| Stack Pointer(SP)        | 0000  |
| Memory Pointer (HL)      | 6403  |
| Program Status Word(PSW) | 0155  |
| Program Counter(PC)      | 000D  |
| Clock Cycle Counter      | 221   |
| Instruction Counter      | 29    |

 | Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |-------------|-------|---|---|---|---|---|---|---|---| | Accumulator | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Register B  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Register C  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Register D  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Register E  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Register H  | 64    | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | Register L  | 05    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | | Memory(M)   | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | Resister      | Value | S | Z | * | AC | * | P | * | CY | |---------------|-------|---|---|---|----|---|---|---|----| | Flag Resister | 54    | 0 | 1 | 0 | 1  | 0 | 1 | 0 | 0  |     | Type                     | Value | |--------------------------|-------| | Stack Pointer(SP)        | 0000  | | Memory Pointer (HL)      | 6405  | | Program Status Word(PSW) | 0054  | | Program Counter(PC)      | 0008  | | Clock Cycle Counter      | 266   | | Instruction Counter      | 37    | |

(a) result of the program

(b) result of the program if we comment line 4 and 5

Figure 9: Result of Program Execution

## 10.6 Conclusion

We see that if there exist a 1 in some memory location, the program sets Cy flag and HL register to stuck address; when there is no 1 in whole array, C is unset. Hence the program is working as expected.

## 11 Assignment 11

### 11.1 Objective

Implement a binary search — the function would take the start address and no. of elements in the array. If successful the function resets CY flag and the HL pair points to the element found else CY is set and the value in HL pair is irrelevant.

### 11.2 Tool/Experimental setup considered

- Jubin's 8085 Simulator

### 11.3 Program

```
1  # ORG 2000H
2  # ARR : DB 2,4,7,8,10
3  # ORG 0000H
4  # N EQU 5
5  # X EQU 7
6
7      MVI C,N          ;Number of elements in the Array
8      DCR C            ;Now we will be using C as the high pointer of Binary Search
9      MVI B,00         ;It is the low pointer of Binary Search
10     MVI D,X           ;Number we are looking for
11     CALL SEARCH
12     LXI H,ARR
13     ADD L
14     MOV L,A
15     JNC NAD
16     INR H
17 NAD:   MOV A,D
18     CMP M
19     JZ ND
20     JC ND
21     HLT
22 ND:    CMC             ;Complement the carry flag
23     HLT               ;Halt
24
25 SEARCH: MOV A,B
26     CMP C             ;Checking if low < high
27     RNC               ;Return if not carry
28     ADD C             ;Now acc has B+C
29     JNC NOCARRY
30     CMC
31
32 NOCARRY: RAR           ;Rotate Accumulator right by 1 bit (A = A >> 1) i.e A = (B+C)/2
33     MOV E,A
34     LXI H,ARR         ;H-L now has 2000h
35     ADD L             ;Accumulator now has the offset to the mid point
36     MOV L,A           ;H-L now points to the mid
37     JNC NOPE
38     INR H
39
40 NOPE:   MOV A,D        ;Accumulator now has the number we are looking for
41     CMP M             ;Compare the number we are looking for with the mid point
42     JC LEFT          ;If less than mid point, go left
43     JZ ND             ;If equal, return
44     MOV B,E           ;E had the previous mid point index
45     INR B             ;Now looking for [M+1:]
46     JMP SEARCH       ;Recall search
47
48 LEFT:   MOV C,E
49     DCR C
50     JNZ SEARCH
51     JZ ND
52     RET
53
54 ;CMP M
55 ;If A less than (R/M), the CY flag is set and Zero flag is reset.
56 ;If A equals to (R/M), the Zero flag is set and CY flag is reset.
57 ;If A greater than (R/M), the CY and Zero flag are reset.
```

Listing 11: assembly program for the implementation of Binary Search

## 11.4 Experimentation

| Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|---|---|---|---|---|---|---|---|
| Accumulator | 07    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| Register B  | 00    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register C  | 04    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Register D  | 07    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| Register E  | 02    | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Register H  | 20    | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Register L  | 02    | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Memory(M)   | 07    | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

  

| Resister      | Value | S | Z | * | AC | * | P | * | CY |
|---------------|-------|---|---|---|----|---|---|---|----|
| Flag Resister | 55    | 0 | 1 | 0 | 1  | 0 | 1 | 0 | 1  |

  

| Type                     | Value |
|--------------------------|-------|
| Stack Pointer(SP)        | FFFE  |
| Memory Pointer (HL)      | 2002  |
| Program Status Word(PSW) | 0755  |
| Program Counter(PC)      | 001D  |
| Clock Cycle Counter      | 148   |
| Instruction Counter      | 22    |

 | Register    | Value | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |-------------|-------|---|---|---|---|---|---|---|---| | Accumulator | 1A    | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | | Register B  | 04    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | Register C  | 04    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | Register D  | 1A    | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | | Register E  | 03    | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | Register H  | 20    | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | Register L  | 04    | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | Memory(M)   | 10    | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |     | Resister      | Value | S | Z | * | AC | * | P | * | CY | |---------------|-------|---|---|---|----|---|---|---|----| | Flag Resister | 04    | 0 | 0 | 0 | 0  | 0 | 1 | 0 | 0  |     | Type                     | Value | |--------------------------|-------| | Stack Pointer(SP)        | 0000  | | Memory Pointer (HL)      | 2004  | | Program Status Word(PSW) | 1A04  | | Program Counter(PC)      | 001B  | | Clock Cycle Counter      | 682   | | Instruction Counter      | 106   | |

(a) result of the program when X is in array

(b) result of the program when X is not in array

Figure 10: Result of Program Execution

## 11.5 Conclusion

We see that when X is in array, Cy flag is set and HL points to the data's address in array; but then X is not present, Cy flag is not set.

Hence the program is working as expected.

## 12 Assignment 12

### 12.1 Objective

Suppose that you are reading a bit of an input port (say, PORT 0) to which the output of a function generator, producing a rectangular wave, is connected. Measure the ON of this repetitive rectangular waveform time in terms of millisecond.

### 12.2 Procedure

1. We first wait in an infinite loop, waiting for the signal to turn 0 (in case we started program when signal was already 1).
2. Then when the signal turns 0, the actual count procedure starts, we again keep waiting in an infinite loop, waiting for the signal to go 1 again.
3. When the signal goes 1 now, we start counting the amount of time the signal is 1 in another infinite loop.
  - The two register pair is too small to count the number of machine cycles it took for the signal to go to 0.
  - so we added a delay (1 ms) after every count, now the count signifies the amount of (1ms) delays it took for signal to go to 0.
4. now the signal is back to 0, and we have the amount of delays it took for signal to go to 0, we write that data to SAVELOC.

### 12.3 Program

```
1  # ORG 1000H
2  ;delay -> 1 ms
3  DELAY:    MVI D,DE
4  DELAYLoop: DCR D
5             JNZ DELAYLoop
6             RET
7
8  # ORG 0000
9  # PORTX EQU 10
10 INPUT:    IN PORTX      ;input data goes to A
11            RAR
12            JC INPUT      ;if already high read again
13 ;now input port data 0 (actual low->high timing cont begins now)
14 L1:        IN PORTX
15            RAR
16            JNC L1         ;if LOW read again
17 ;now input HIGH -> start counting
18            LXI B,0000
19 COUNT:     INX B          ;6T
20            CALL DELAY
21            IN PORTX      ;10T
22            RAR           ;4T
23            JC COUNT      ;10T -> jump till input is 1
24 ;now input is 0 we save counter
25 ;time taken = BC*(30*1/3 + 1ms) = BC*11ms +- 2 ms
26 # SAVELOC EQU 5000H
27            LXI H,SAVELOC
28            MOV M,B
29            INX H
30            MOV M,C
31            HLT
```

Listing 12: assembly program to calculate time period of rectangular wave

### 12.4 Conclusion

We created a program which will calculate the time period of a rectangular wave as follows

$$T = BC * (30 * 1/3 + 1ms) = BC * 11ms \pm 2ms$$