INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

MICROPROCESSOR LAB REPORT

PROFESSOR	PROFESSOR	SUB CODE:
PROF. BIPLAB K. SIKDAR	PROF. AMIT KUMAR DAS	CS3171

NAME: ARNAB SEN
EN.NO: 510519006
GSUITE: 510519006.arnab@students.iiests.ac.in
SEM: 5TH
BATCH: 2019-2023

Table of Contents

Assignment 1

Question 1

Objective

Find out the sum of the **first 30 natural numbers**.

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

The sum of first 30 natural numbers is `435` i.e `0x01D1` . Since the value is greater than `256` we need two register pair to store the value.

Program

```
        LXI H,0000   ; Load H-L pair with 0000H, it will be used as an accumulator
        MVI D,1E     ; Move immediate data to register, D storing 0x1E = 30
        MVI C,01     ; B-C pair storing 00-01

L1:     DAD B        ; Double ADd
        INX B        ; INcrementeXtended register, increments B-C
        DCR D        ; DeCRement, decrements D
        JNZ L1       ; Jump Not Zero
        SHLD 8085    ; Store HLpair using Direct addressing, storing the data in 8085
        HLT  ; Halt
```

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

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)	01D1
Program Status Word(PSW)	0054
Program Counter(PC)	0010
Clock Cycle Counter	942
Instruction Counter	125

Conclusion

Finally the data stored in HL register pair is `0x01D1` i.e `435` . This is the sum of first 30 natural numbers.**Hence the program is working as expected.**

Question 2

Objective

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

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

Loop through the entire array of integers, and store the maximum and minimum values in the registers.

Program

```
# ORG 4200H
# ARR: DB 9, 1, 8, 4, 8, A, F, 5
# LEN EQU 08
# ORG 0000H
        LXI H,ARR
        MVI B,LEN ; number of elements
        MOV D,M  ; storing the min in D
        MOV C,M  ; storing the max in C
        DCR B

LOOP:
        INX H
        MOV A,M
        CMP D
        JNC MAX
        MOV D,M

MAX:
        CMP C
        JC AHEAD
        MOV C,A

AHEAD:
        DCR B
        JNZ LOOP
        MOV A,D
```

```
STA 4300
MOV A,C
STA 4400
HLT
```

Experimentation

Register	Value	7	6	5	4	3	2	1	0
Accumulator	0F	0	0	0	0	1	1	1	1
Register B	00	0	0	0	0	0	0	0	0
Register C	0F	0	0	0	0	1	1	1	1
Register D	01	0	0	0	0	0	0	0	1
Register E	00	0	0	0	0	0	0	0	0
Register H	42	0	1	0	0	0	0	1	0
Register L	07	0	0	0	0	0	1	1	1
Memory(M)	05	0	0	0	0	0	1	0	1

4200	09
4201	01
4202	08
4203	04
4204	08
4205	0A
4206	0F
4207	05
4300	01
4400	0F

Conclusion

After the execution the maximum value is stored in register C and the minimum in register D.

Question 3

Objective

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

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

We can simulate the delay by running a long loop. The loop will be executed for the number of times specified in DE register pair.

Program

```
      LXI D, E000H
      CALL DELAY
      HLT
DELAY: DCX D
      MOV A, D
      ORA E
      JNZ DELAY
      RET
```

Conclusion

Will notice the code runs for sometime and the value of DE register pair is decremented. Hence the delay is produced.

Assignment 2

Question 1

Objective

Write a subroutine to move a block of bytes from location X to location Y. Note that the caller would specify X and Y; the source and destination along with the block size, say, Z. X, Y and Z are 16-bit quantities.

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

For this program, I am

- Storing Z in B-C
- Storing X in D-E
- Storing Y in H-L

Now, we will start reading from X and writing to Y, Z times and the entire block will be copied.

Program

```
; Storing Z in B-C
; Storing X in D-E
; Storing Y in H-L

# ORG 3000H
# ARR: DB 1, 2, 3, 4
```

```
# ORG 0000H
LXI B, 0004 ; count
LXI D, 3000 ; source
LXI H, 4000 ; destination
CALL MOVE
HLT

; Moving 4 bytes from 3000 -> 4000

MOVE:
  LDAX D  ; Stores the content of D-E in A
  MOV M, A ; M -> HL
  INX H
  INX D
  DCX B
  MOV A, B
  ORA C
  JNZ MOVE
  RET
```

Experimentation

0017	C9
3000	01
3001	02
3002	03
3003	04
4000	01
4001	02
4002	03
4003	04
FFFE	0C

Conclusion

We can see that values from `4000` to `4004` is copied from `3000` to `3004`. Here Z is `4`, X is `3000`, and Y is `4000`. Hence, the block of bytes is copied.

Question 2

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.

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

Odd numbers always have their 0-th bit set. So, we can use the AND instruction to check if the 0-th bit is set.

Program

```
# NUM EQU 7
MVI A, NUM
CALL ISODD
HLT

// Will store 1 in register B if odd, else 0.
ISODD:
  ANI 01    ; AND Immediate with the data and store the result in Acc
```

```
MOV B,A
RET
```

Experimentation

When the `NUM` is 7 (Odd).

Register	Value	7	6	5	4	3	2	1	0
Accumulator	01	0	0	0	0	0	0	0	1
Register B	01	0	0	0	0	0	0	0	1
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)	3E	0	0	1	1	1	1	1	0

When the `NUM` is 8 (Even).

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	00	0	0	0	0	0	0	0	0
Register L	00	0	0	0	0	0	0	0	0
Memory(M)	3E	0	0	1	1	1	1	1	0

Conclusion

In case the `NUM` was odd, the register C was set. And, when the `NUM` was even, the register C was not set. Hence, the function returns 1 if the `NUM` was odd and 0 if the `NUM` was even.

Question 3

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.

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

Looped through the no. of bytes to be added. Each time we add the content of X and Y and store the result in X. For that we use the `ADC` which adds the content along with the carry. Finally, we check if there is a carry and if yes, we add 1 to the result.

> **Note:** Here we are assuming the numbers are written in Little-Endian format i.e the most significant bit is stored in the largest address.

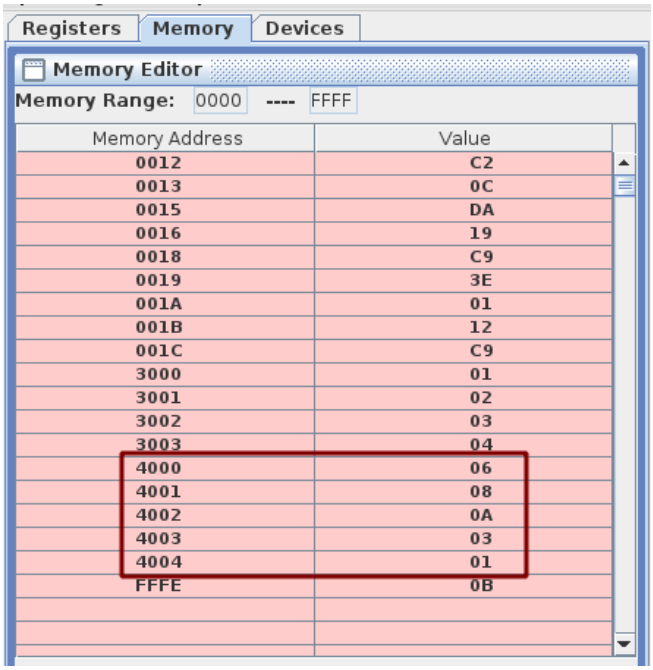
Program

```
# ORG 3000h
# ARR: DB 01,02,03,04

# ORG 4000h
# ARR: DB 05,06,07,FF

# LEN EQU 04

# ORG 0000h
LXI H,3000H ; Y
```

```
LXI D,4000H ; X
MVI B, LEN  ; Z
CALL SUM
HLT

SUM:
  LDAX D ;copies the contents of that memory location D-E into the accumulator
  ADC M  ;ADd with Carry
  STAX D ;contents of the A are copied into the memory location
  INX D
  INX H
  DCR B
  JNZ SUM
  JC CARRY ;If there is some left out carry
  RET

CARRY:
  MVI A, 01h
  STAX D
  RET
```

Experimentation

Adding `01,02,03,04` and `05,06,07,FF`



Conclusion

Result of adding `01,02,03,04` and `05,06,07,FF` is `06,08,0A,03,01` and that is exactly what we got. Hence, the function returns the correct result.

Assignment 3

Question 1

Objective

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

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

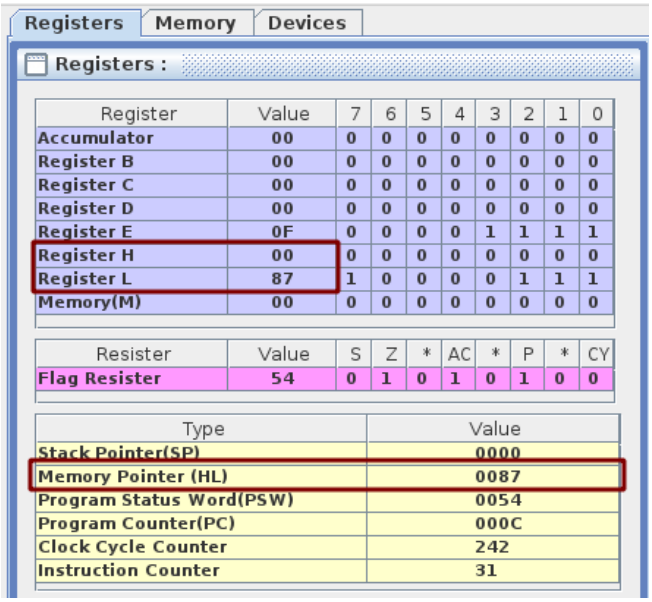
Procedure

We can add 15, 9 times to get 9*15 because multiplication is nothing but repeated addition.

Program

```
LXI H,0000H
MVI E,0FH
MVI B,09H
LOOP:
  DAD D ; HL <- HL + DE
  DCR B ; Decrease B (1-byte)
  JNZ LOOP
HLT
```

Experimentation



Conclusion

9*15 = 135 which in hex is `0x87` . Above we can see that the value in the H-L pair is `0087h` . Hence, our program was able to calculate 9 times 15.

Question 2

Objective

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

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

Performing a simple bubble sort, where we compare adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

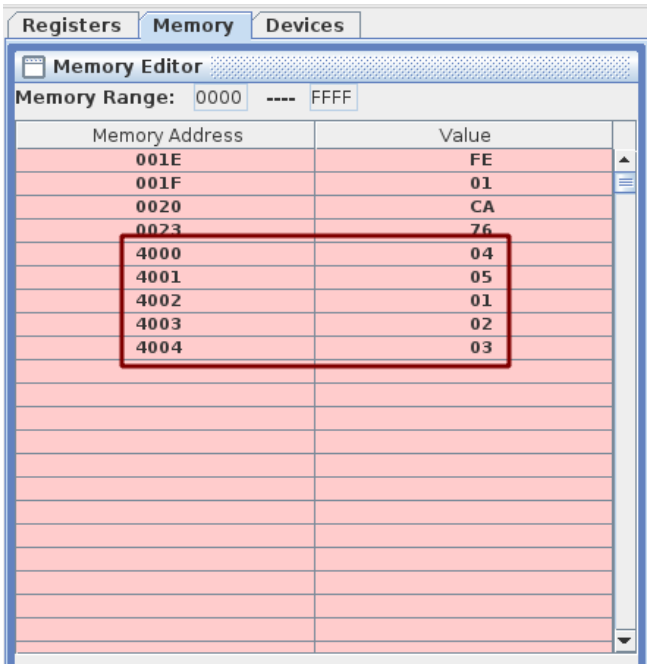
Program

```
# ORG 4000h
# ARR: DB 04,05,01,02,03
# LEN EQU 5

# ORG 0000h

MAIN:
  LXI H,ARR
  MVI D,00
  MOV C,LEN
  DCR C
CHECK:
  MOV A,M
  INX H
  CMP M
  JC NEXT
  JZ NEXT
  CALL SWAP
SWAP:
  MOV B,M
  MOV M,A
  DCX H
```

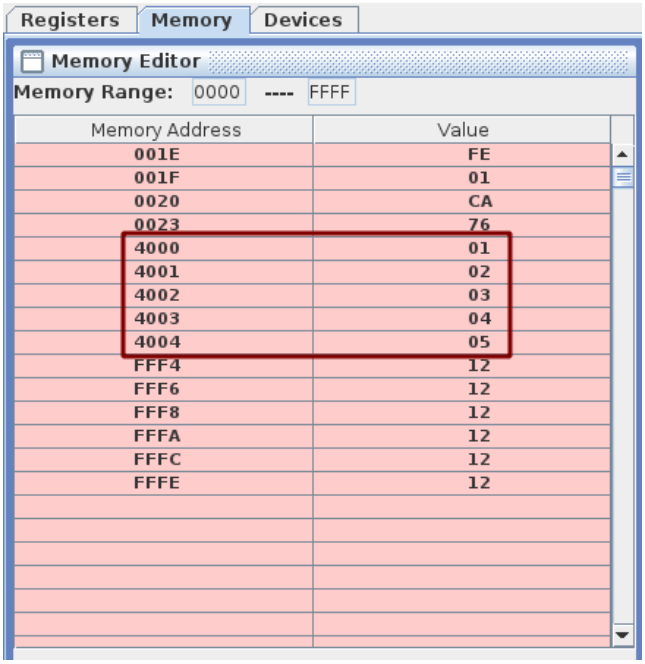
```
    MOV M,B
    INX H
    MVI D,01

  NEXT:
    DCR C
    JNZ CHECK
    MOV A,D
    CPI 01
    JZ MAIN
    HLT
```

Experimentation

Before sorting (just assembling the code)



After sorting



Conclusion

Finally we can see in the memory range `4000` to `4004` that the numbers are all sorted, hence our code worked.

Question 3

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` .

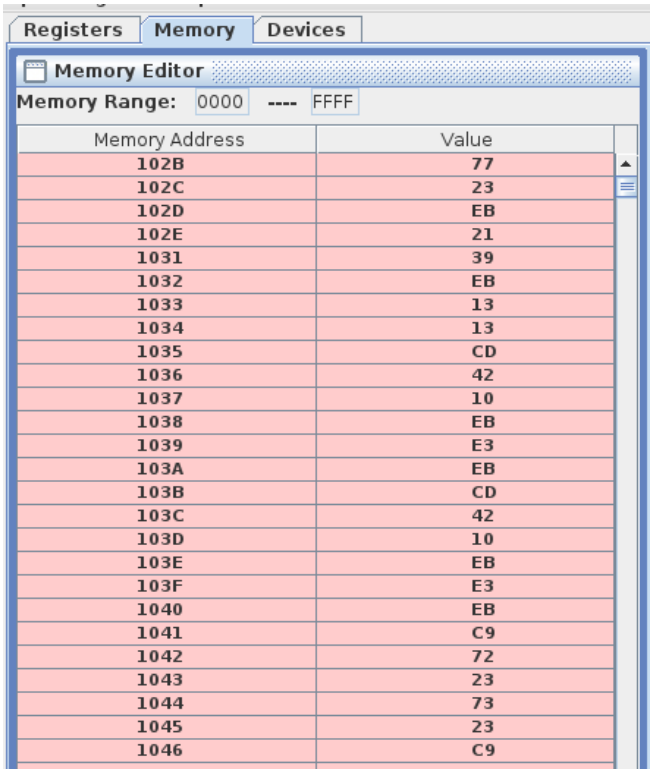
Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Program

```
# ORG 1000H
# MYREG EQU 2000H
        MVI A,10
        LXI B,1001
        LXI D,2002
        LXI H,3003
        SIM
        LXI SP,F001h
        CALL MAIN
        HLT

MAIN:
  PUSH H
  PUSH D
  PUSH B
  PUSH PSW
  LXI H,MYREG
  POP D
  CALL STORE
  POP D
  CALL STORE
  POP D
  CALL STORE
  POP D
  CALL STORE
  RIM
  MOV M,A
  INX H
  XCHG
  LXI H,0000
  DAD SP
  XCHG
  INX D
  INX D
  CALL STORE
  XCHG
  XTHL
  XCHG
  CALL STORE
  XCHG
  XTHL
  XCHG
  RET

STORE:
  MOV M,D
  INX H
  MOV M,E
  INX H
  RET
```

Experimentation



Conclusion

It is storing all the registers in that order mentioned in the question.

4/4.md

Assignment 4

Question 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.]

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

We iterate through the block of memory and check if the value is 1. Once we get 1 we exit setting the C flag.

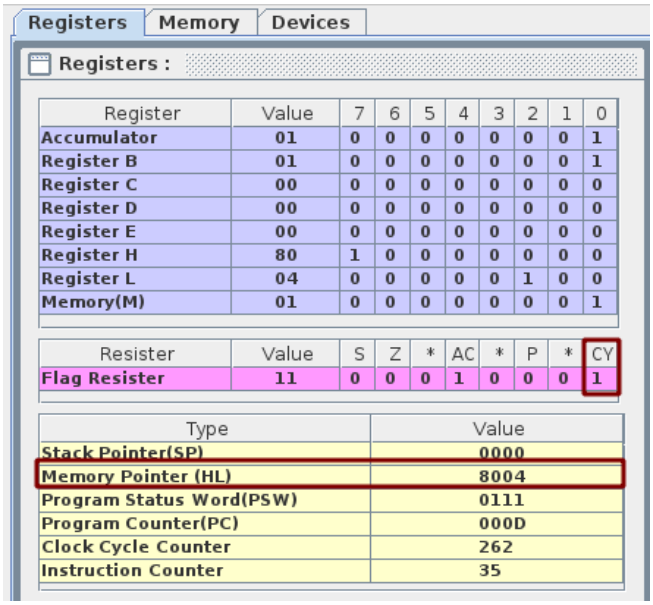
Program

```
// Assuming 8004 is faulty
LXI H,8004
MVI M,01
// Checking from 805 till 5 bytes
LXI H,8000
MVI B,05
CALL CHECK
HLT

CHECK:
    MOV A,M
    ANI 01       // Doing ANDImmediate with 01
    JNZ FAULTY   // If Z is set
    INX H
    DCR B
    JNZ CHECK
    CMC
```

```
    STC
    RET

FAULTY:
    STC    // set Cy 1
    RET
```

Experimentation

Assuming that `8004` is faulty.



Registers : Memory : Devices

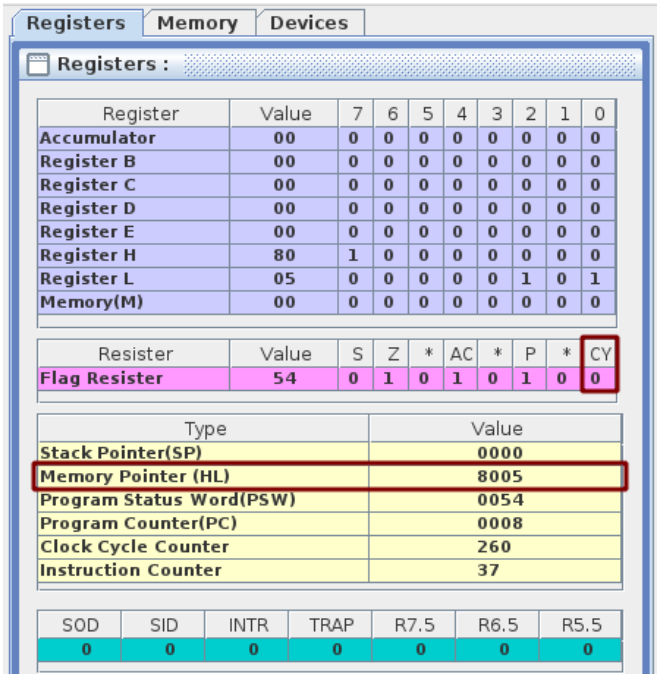
Registers :

Register	Value	7	6	5	4	3	2	1	0
Accumulator	01	0	0	0	0	0	0	0	1
Register B	01	0	0	0	0	0	0	0	1
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	80	1	0	0	0	0	0	0	0
Register L	04	0	0	0	0	0	1	0	0
Memory(M)	01	0	0	0	0	0	0	0	1

Resister	Value	S	Z	*	AC	*	P	*	CY
Flag Resister	11	0	0	0	1	0	0	0	1

Type	Value
Stack Pointer(SP)	0000
Memory Pointer (HL)	8004
Program Status Word(PSW)	0111
Program Counter(PC)	000D
Clock Cycle Counter	262
Instruction Counter	35

When nothing is faulty.



Registers : Memory : Devices

Registers :

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	80	1	0	0	0	0	0	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)	8005
Program Status Word(PSW)	0054
Program Counter(PC)	0008
Clock Cycle Counter	260
Instruction Counter	37

SOD	SID	INTR	TRAP	R7.5	R6.5	R5.5
0	0	0	0	0	0	0

Conclusion

We can see that the memory pointer is at `8004` and the C flag is `1` hence it suggests that the memory address `8004` is faulty. In the second scenario, where we had no faulty, we can see that C flag remains `0` .

Hence, our code is working properly.

Question 2

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.

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

We keep two pointers, one low(Register `B`) and another high(Register `C`). We will check if the mid of this two pointer is the value we are looking for. If we found the value then we set the `CY` flag and break, else we keep the `CY` flag unset.

> **NOTE:** We are assuming that the array is sorted

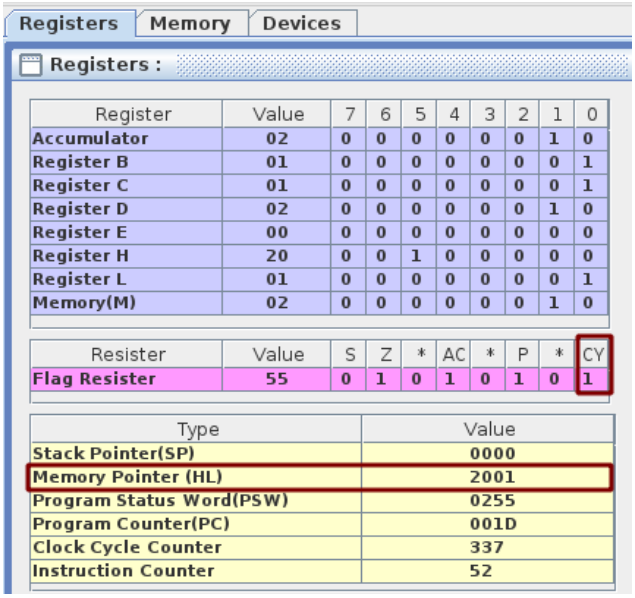
Program

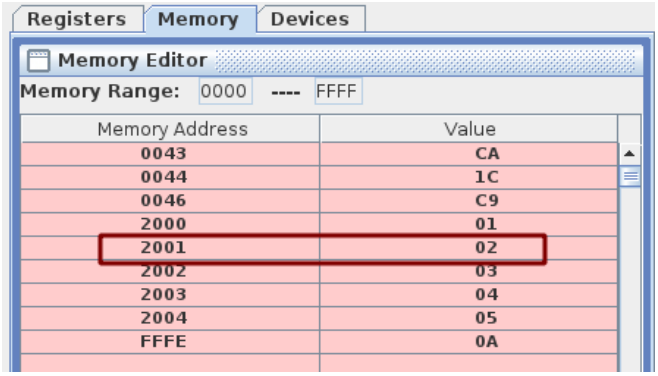
```
# ORG 2000H
# ARR : DB 1,2,3,4,5
# ORG 0000H
# N EQU 5
# X EQU 2

MVI C, N
DCR C
MVI B,00
MVI D, X
CALL SEARCH
LXI H,ARR
ADD L
MOV L,A
JNC NAD
INR H


NAD:
  MOV A,D
  CMP M
  JZ END
  JC END
  HLT

END:
  CMC
  HLT

SEARCH:
  MOV A,B
  CMP C
  RNC
  ADD C
  JNC NOCARRY
  CMC

NOCARRY:
  RAR
  MOV E,A
  LXI H,ARR
  ADD L
  MOV L,A
  JNC NOTFOUND
  INR H

NOTFOUND:
  MOV A,D
  CMP M
  JC LEFT
  JZ END
```

```
    MOV B,E
    INR B
    JMP SEARCH

  LEFT:
    MOV C,E
    DCR C
    JNZ SEARCH
    JZ END
    RET
```

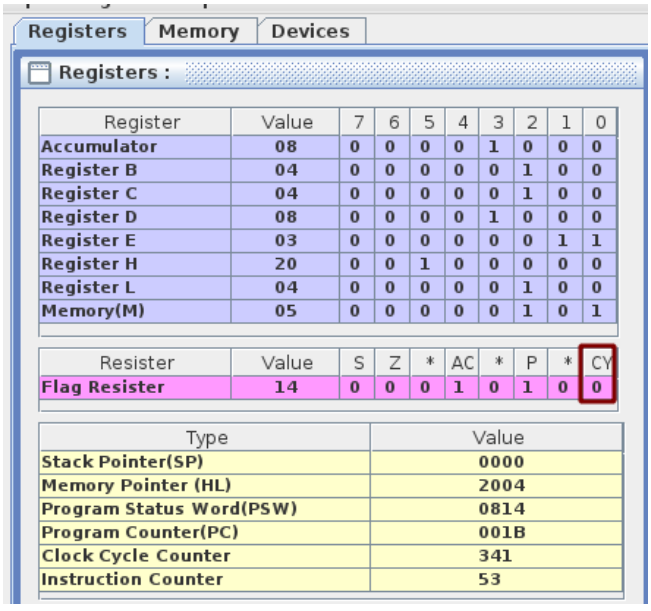
Experimentation

When the array is `{1, 2, 3, 4, 5}` and we are looking for `2`.





When the array is `{1, 2, 3, 4, 5}` and we are looking for `8`.



Conclusion

Clearly, when the element is found in the array the `CY` flag was set, and the address pointed by the memory pointer was the value we are looking for. On the other hand when the element is not there in the array we see that the `CY` flag remains unset. Hence, our program is working as expected.

Assignment 5

Question 1

Objective

Using auto vectored input RST 7.5 prepare a scheme to count the number of key-press done at this interrupting input.

> The main routine after initialisation of the interrupt mechanism waits in an infinite loop waiting for the key-press. On a key-press (that simulates as if you have excited the RST 7.5 input) it increases a counter at a predefined memory location (used to hold the count value). You may exit from this routine and then check the counter value

Tool / Experimental setup considered

- Used Jubin's 8085 Simulator.

Procedure

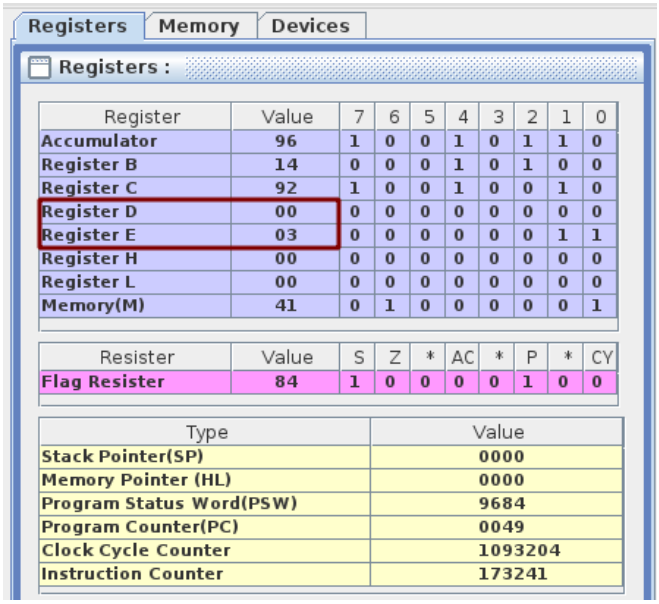
We will use an RST 7.5 interrupt line to call a procedure every time we get an interrupt.

> To prevent multiple interrupts being registered at the same time, we are using a small delay.

Program

```
# ORG 0000H
    MVI A,0B // To enable R7.5
    SIM // Set Interrupt Mask
    EI  // Enables Interrupt

LOOP:
    MVI A,01
    JNZ LOOP

# ORG 3C
    DI
    INX D
    CALL DEL80 // Adding some delay
    EI
    RET

DEL80:     LXI B,28AF

DEL80LOOP:
    DCX B
    MOV A,B
    ORA C
    JNZ DEL80LOOP
    RET
```

Experimentation



Conclusion

After pressing the interrupt 3 times, we can see that the `D-C` register has the value 3. After trying it out for multiple interrupts we are getting accurate results. Hence, our program is working.