EAST WEST UNIVERSITY

Subject Code: CSE360

Section: 2

Spring 2020

Group Project

Topic: Implementing Binary Search Using 8085 Assembly Language.

Submitted To:

Dr. Ahmed Wasif Reza

Associate Professor

Department of CSE

East West University

Submitted By:

Kafi Khan

ID: 2017-2-60-078

Yeasir Arafat Shahed

ID: 2017-2-60-089

Sharif Mulla Mahin

ID: 2017-2-60-023

Date of Submission

14/05/2020

Implementing Binary Search Using 8085 Assembly Language.

Objective: The goal of the project is to learn fundamentals of assembly language. In doing so, we also want to learn how a computer operates internally. These are the primary questions that we want to be able to answer after finishing the project.

a. How data is represented in memory and other external devices.
b. How the processor accesses and executes instruction.
c. How instructions access and process data.

Theory: 8085 assembly language primarily operates on 8-bit data. It does so by using Registers, Flags, Memory and I/O ports.

Registers: There are 14 registers in 8085 processors. Here is the description of the primary ones,

A - The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU). After performing arithmetic or logical operations, the result is stored in the accumulator.

B, C, D, E, H, and L - General-purpose registers. These registers are used to store or copy temporary data, by using instructions, during the execution of the program.

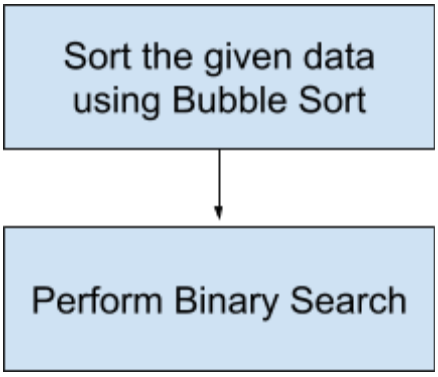
PC - Program counter. This register is used to sequence the execution of the instructions.

SP - Stack pointer. It points to a memory location in read/write memory, called the stack. It is always incremented/decremented by 2 during push and pop operation. There are two more, Program Status Word and Int-Reg.

Flags: There are five flag bits as follows; Sign Flag, Zero Flag, Auxiliary Carry Flag, Parity Flag and Carry Flag.

Memory: There are 65536 memory addresses from 0 to 65535, each having 8-bits.

Design:



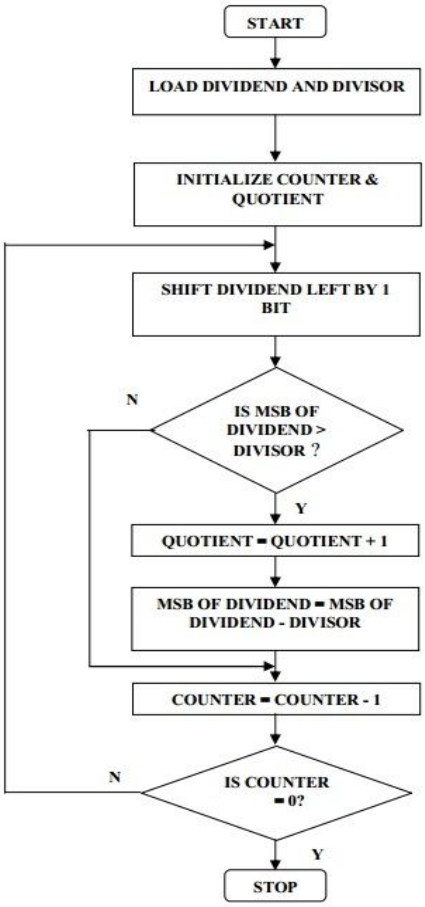
The algorithm for Bubble sort is as follows:

```
for i from 1 to N
    for j from 0 to N-1
        if a[j]>a[j+1]
          swap(a[j], a[j+1])
```

The algorithm for Binary Search is as follows:

```
while x not found
    if upperBound < lowerBound
      EXIT: x does not exist.
    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2
    if A[midPoint] < x
      set lowerBound = midPoint + 1
    if A[midPoint] > x
      set upperBound = midPoint - 1
    if A[midPoint] = x
      EXIT: x found at location midPoint
 end while
```

The algorithm for division of two numbers:



Implementation:

Code for Bubble Sort is as follows:

MVI L, 03H ; move 03h(hexadecimal) into L register

MOV A, L ; move the data of l into accumulator

STA 000BH ; stores the data in accumulator into 000BH memory address

LOOP2: MVI B, 00H

MVI C, 00H ; the array of data starts from 0000H location.

MVI D, 00H

MVI E, 01H

MVI L, 03H

LOOP1: LDAX B ; loads the data found in the address stored in BC register in A register

MOV H, A

LDAX D ; loads the data found in the address stored in DE register in A register

CMP H ; compares the data in A with the data of H

; If A < M, Carry Flag is set to 1, and Zero Flag is set to 0

; If A = M, Carry Flag is set to 0, and Zero Flag is set to 1

; If A > M, Carry Flag is set to 0, and Zero Flag is set to 0

CC GREATER ; If carry flag is set to 1, hence A < M, then jump to a subroutine called ‘greater’

INR E ; increment the data in e

INR C

MOV A, L

DCR A ; decrement the data in a

MOV L, A

JNZ LOOP1 ; if the data in a is not zero, then jump to the subroutine ‘loop1’

LDA 000BH

DCR A

STA 000BH

JNZ LOOP2

At first BC and DE holds the addresses of the data of n and n+1. BC’s data goes to H and DE’s data resides in A. They get compared. So if (n+1)th data is smaller than nth data, they get swapped in ‘greater’ subroutine. Then the BC and DE hold the address of n+1 and n+2. Since for an N number of data, we have to iterate two loops N time. So loop1 and loop2 have an iterator with initial value N, and on each iteration they get decremented until the value is zero and the data is sorted.

The code for Binary Search: LOOP3: MVI A, 00H

MVI B, 00H

MVI D, 00H

```
MVI H, 00H
MVI C, 0DH; 0000DH hols the the lower bound
MVI E, 0EH; 000EH holds the median
MVI L, 0FH; 000FH holds the upper bound
LDAX B
MOV C, A
MOV A, M; M is an imaginary register that can hold the value of the address placed in HL
;register
MOV L, A
MOV A, L
CMP C ; compares the upper bound with lower bound.
JC HALT ; if upper bound is lower than lower bound, ‘halt’ subroutine terminates the program
MOV A, L
SUB C
STA 0031H
CALL DIVISION ; this subroutine stores the value quotient into 0033H memory location
LDA 0033H
MOV D, A
LDA 000DH
ADD D ; calculates midian
STA 000EH; stores the midian into 000EH location
LDA 0EH; loads median into accumulator
MOV E, A
MVI D, 00H
LDAX D ; loads the data of median into accumulator
MOV B, A
LDA 000CH
CMP B ; compares the searched number with the median
```

```
CNC GREATEROREQUAL ; if the number is greater or equal, the program calls the subroutine
;‘greaterorequal’
CC SMALLER ;
JMP LOOP3 ; keeps jumping to loop3 until the searched value is found or lower bound is >
;upper bound
```

The subroutines are as follows:

```
DIVISION: LHLD 31H ;
MVI B,02H
MVI C, 08
UP: DAD H
MOV A, H
SUB B
JC DOWN
MOV H, A
INR L
DOWN: DCR C
JNZ UP
SHLD 0033H
RET
HALT: MVI A, 100
STA 000AH
HLT
INCREMENTLOWER: LDA 000EH
INR A
STA 000DH
RET ;CHANGE IT
EQUAL: MVI A, 200
STA 000AH
HLT
```

GREATEROREQUAL: CZ EQUAL

CNZ INCREMENTLOWER

RET

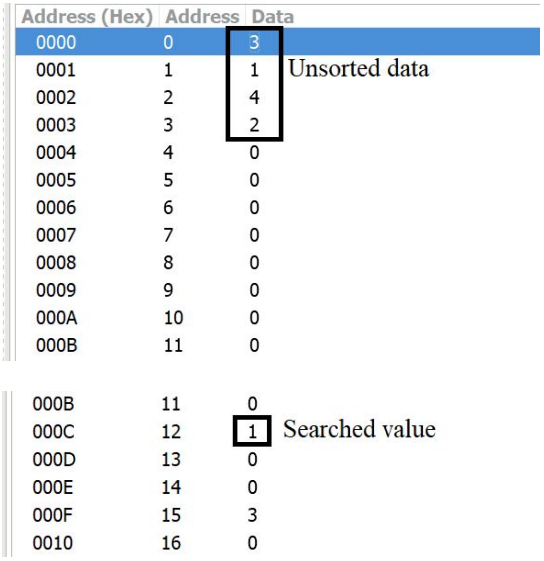
SMALLER: LDA 000EH

DCR A

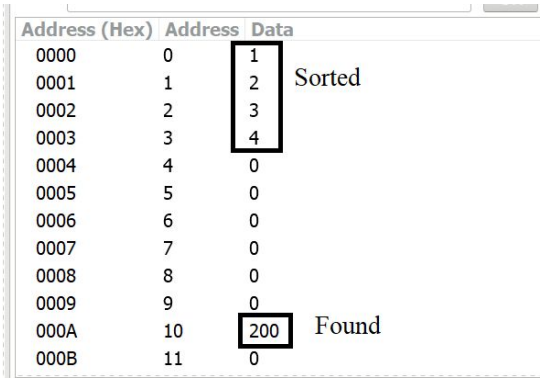
STA 000FH

RET

Debugging-Test-run: Test Case 1:

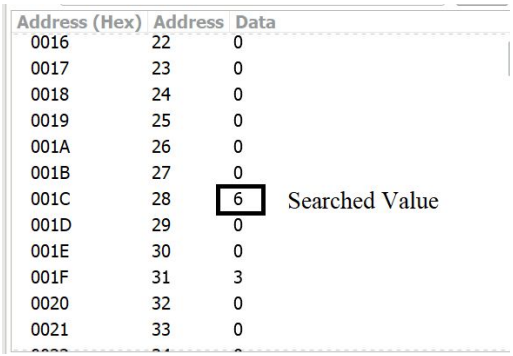


Address (Hex)	Address	Data	
0000	0	3	
0001	1	1	Unsorted data
0002	2	4	
0003	3	2	
0004	4	0	
0005	5	0	
0006	6	0	
0007	7	0	
0008	8	0	
0009	9	0	
000A	10	0	
000B	11	0	
000B	11	0	
000C	12	1	Searched value
000D	13	0	
000E	14	0	
000F	15	3	
0010	16	0	

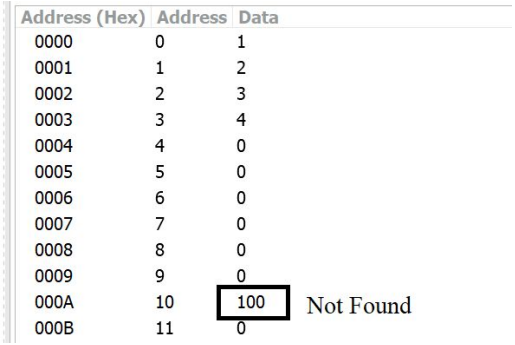


Address (Hex)	Address	Data	
0000	0	1	
0001	1	2	Sorted
0002	2	3	
0003	3	4	
0004	4	0	
0005	5	0	
0006	6	0	
0007	7	0	
0008	8	0	
0009	9	0	
000A	10	200	Found
000B	11	0	

Test Case 2:



Address (Hex)	Address	Data	
0016	22	0	
0017	23	0	
0018	24	0	
0019	25	0	
001A	26	0	
001B	27	0	
001C	28	6	Searched Value
001D	29	0	
001E	30	0	
001F	31	3	
0020	32	0	
0021	33	0	



Address (Hex)	Address	Data	
0000	0	1	
0001	1	2	
0002	2	3	
0003	3	4	
0004	4	0	
0005	5	0	
0006	6	0	
0007	7	0	
0008	8	0	
0009	9	0	
000A	10	100	Not Found
000B	11	0	

Here are two cases. If the value is found, 200 is set on 000AH memory location. If the value is not in the array, the 100 is set on 000AH memory location.

Results analysis: Time Complexity of the algorithm: Time Complexity of Bubble Sort + Time Complexity of Binary Search = O(n^2) + log(n) = O(n^2)

Knows Bugs: 1. This program cannot work with 16-bit numbers

2. It cannot work with negative numbers

3. If the searched value is lower than the lowest value in array, the program crashes.

Conclusion and Future Improvements: The primary significance of Binary Search is its time complexity which is much lower than brute force search. So, a good sorting algorithm is required to take advantage of binary sort. Unfortunately, we have used bubble sort, which has the time complexity of O(n^2) which defeats the whole purpose of Binary Search. Any divide and conquer algorithm would be perfect; but they are much harder to implement on an 8085 assembly language.

Furthermore, working with 16-bit numbers is extremely plausible in an 8085 processor. For example, BC registers can work together to hold 16-bit numbers. But, for time shortages, we could not implement it.

Bibliography:

1. Astha Singh. (2020, February 27). Registers of 8085 microprocessor. Retrieved from https://www.geeksforgeeks.org/registers-8085-microprocessor/
2. Bubble sort. (n.d.). Retrieved from https://algorithmist.com/wiki/Bubble_sort
3. 8085 Program to Divide two 8 Bit numbers. (n.d.). Retrieved from https://www.tutorialspoint.com/8085-program-to-divide-two-8-bit-numbers

Project Link: https://drive.google.com/open?id=1Jq3IdyKS_PdHtrHDGim7yuxE4aerfEPS