Implementation of Quick Sort using an 8085 Assembly language

MINI PROJECT REPORT

By

Saidinesh (RA2211029010016)
Naga kiran (RA2211028010230)
Rahul (RA2211029010024)
Akshay(RA2211028010029)
Anand(RA2211029010025)

Under the guidance of

Dr. S Ramesh

Assistant Professor, Department of Networking and Communications

In partial fulfilment for the Course of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
with specialization in Computer Networking

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSS201T– Computer Organization and Architecture** entitled in "**Implementation of Quick Sort using 8085 assembly language**" is the bonafide work of **Saidinesh (RA2211029010016), Nagakiran (RA2211028010230), Rahul (RA2211029010024) , Akshay(RA2211028010029)Anand(RA2211029010025)** who carried out the work under my supervision.

SIGNATURE

Dr. S Ramesh

COA– Course Faculty

Assistant Professor

Department of Computational Intelligence

SRM Institute of Science and Technology

SIGNATURE

Dr Annapurani Panaiyappan

Head of the Department

Professor

Department of Computational Intelligence

SRM Institute of Science and Technology Kattankulathur

DEPARTMENT OF
SCHOOL OF COMPUTING

College of Engineering and Technology
SRM Institute of Science and Technology

MINI PROJECT REPORT

ODD Semester, 2023-2024

Lab code & Sub Name : 21CSS201T & Computer Organization and Architecture

Year & Semester : II & III

Project Title : Implementation of Quick sort using 8085 assembly language

Lab Supervisor **: Dr. S Ramesh**

Team Members : Saidinesh (RA2211029010016)
Naga kiran (2211028010230)
Rahul (2211029010024)
Akshay(RA2211028010029)
Anand(RA2211029010025)
h

Particulars	**Max. Marks**	**Marks Obtained**
		Name:
		Register No :
Program and Execution	20	
Demo verification &viva	15	
Project Report	05	
Total	**40**	

Date :

Staff Name :

Signature :

TABLE OF CONTENT

Sl. NO.	TITLE	PAGE NO.
1.	OBJECTIVE	**3**
2.	ABSTRACT	**4**
3.	INTRODUCTION	**5**
4.	HARDWARE / SOFTWARE REQUIREMENTS	**6**
5.	CONCEPTS / WORKING PRINCIPLE	**7-8**
6.	APPROACH/METHODOLOGY/PROGRAMS	**9-13**
7.	FLOWCHART:	**14**
8.	CODE	**14-15**
9.	CONCLUSION	**16**
10.	REFERENCES	**17**

The Implementation of Quick Sort using an 8085 microprocessor

OBJECTIVE:

The objective of this project is to implement the Quick Sort algorithm utilizing the capabilities of the 8085 microprocessors. Through this implementation, we aim to demonstrate the efficiency and functionality of Quick Sort in a resource-constrained environment, showcasing the adaptability of sorting algorithms in diverse computing architectures.

Implementing Quick Sort using an 8085 microprocessor is a demanding undertaking that requires a deep understanding of both the Quick Sort algorithm and the peculiarities of the 8085 architectures. The 8085 microprocessors, a popular member of the Intel 8-bit microprocessor family, presents several constraints and challenges due to its limited computational power, addressing capabilities, and lack of dedicated stack support.

To embark on this project, you need to break down the Quick Sort algorithm into a series of discrete steps that can be implemented with the 8085's instruction set. Here is a more detailed breakdown of these steps:

Initialization: Start by loading the unsorted array into memory. Allocate memory locations to store the pivot element, indices for the subarrays, and temporary variables. The 8085 architecture uses an accumulator (A) and several register pairs (BC, DE, and HL) for data manipulation. You will need to carefully allocate these resources.

Quick Sort fundamentally relies on partitioning the array into two subarrays based on a selected pivot element. In the context of the 8085, you will need to compare elements with the pivot using instructions like CMP and JUMP. You will also need to rearrange elements, ensuring that all those less than the pivot end up on one side, and all those greater on the other side. This can involve careful memory and register manipulation.

After partitioning, you will have two subarrays. Implementing the recursive aspect of Quick Sort is particularly challenging on the 8085. You can manage this by manually maintaining a stack in memory to store return addresses and parameters for the subproblems. The 8085 does not have a dedicated hardware stack like more modern processors, so this stack management is an essential part of the implementation.

As subarrays are sorted, you will need to combine them to produce the final sorted array. This process may involve shifting elements in memory, merging subarrays, and carefully updating indices.

ABSTRACT:

The implementation of Quick Sort using an 8085 microprocessor is a highly intricate and demanding project, requiring a comprehensive understanding of both the Quick Sort algorithm and the intricacies of the 8085 architecture. This project involves the translation of the high-level algorithm into the low-level assembly language of the 8085, and it presents numerous challenges due to the microprocessor's limitations.

The project begins with an elaborate initialization phase, where the unsorted array is loaded into the limited memory space of the 8085. Memory locations must be thoughtfully allocated for variables such as the pivot element, subarray indices, and temporary storage. The 8085 architecture utilizes an accumulator (A) and various register pairs (BC, DE, HL), and effective resource allocation is pivotal to data manipulation efficiency.

The heart of the Quick Sort algorithm is the partitioning step. On the 8085, this translates into a complex series of instructions involving comparisons and branching. The goal is to segregate array elements based on a selected pivot, ensuring that those smaller than the pivot are placed on one side and those greater on the other. Achieving this effectively demands meticulous memory and register manipulation, stretching the programmer's assembly language skills to their limits.

Recursion is another fundamental component, and it's particularly intricate on the 8085 due to the absence of a dedicated hardware stack. Instead, a manual stack management approach is required. This involves creating and maintaining a stack in memory to store return addresses and parameters for recursive subproblems. Keeping this stack in proper order is crucial to prevent stack overflows and underflows, and it often involves innovative problem-solving.

The merging of sorted subarrays is the subsequent phase, and this operation entails the careful manipulation of memory to combine elements and update indices efficiently. Termination conditions need to be vigilantly monitored to ensure that the algorithm exits the recursive calls when the sorting process is complete.

Efficiency becomes a paramount concern throughout the project due to the 8085's resource constraints. Optimizing memory usage, judiciously utilizing available instructions, and minimizing data transfers are pivotal for preventing excessively long sorting times, especially for moderately sized arrays.

The thorough testing and debugging of the implementation is a critical aspect of this project. The 8085 lacks modern debugging tools or simulators, which means that the programmer must rely on creative debugging techniques and manual verification to ensure the correctness of the code.

INTRODUCTION:

The implementation of Quick Sort using an 8085 microprocessor offers a captivating journey into the heart of sorting algorithms and the intriguing challenges posed by early computing hardware. Sorting, as a foundational computational task, holds universal importance across the vast landscape of computer science applications. Whether it's managing vast databases, enhancing search algorithms for information retrieval, or enabling data analysis in various domains, sorting serves as the bedrock upon which many critical processes in computing rely.

Quick Sort, the algorithm of choice for this project, is celebrated for its efficiency and scalability. With an average-case time complexity of O(n log n), it has become a workhorse for sorting large datasets. Yet, the true test here is not just the application of Quick Sort but its adaptation to the 8085 microprocessor, a relic of early computing history. The 8085, being an 8-bit microprocessor, operates under significant hardware constraints - limited computational power, minimal memory capacity, and the absence of sophisticated features found in contemporary processors.

The foremost application of this project lies in demonstrating the resilience and versatility of algorithms when confronted with hardware restrictions. By implementing Quick Sort on the 8085, this endeavor showcases that even in resource-scarce environments, sophisticated algorithms can find a way to excel. It's a testament to the timelessness of algorithmic efficiency and the creativity needed to adapt algorithms to specific hardware contexts.

Furthermore, this project serves as an invaluable educational resource. It provides a rich learning experience in the intricacies of algorithm design and the fine art of low-level programming. These skills, often considered as timeless fundamentals of computer science, are vividly demonstrated here in the challenging context of porting a modern algorithm to an early microprocessor. This effort hearkens back to the pioneering days of computing, where each byte of memory and every clock cycle counted, and where the efficient use of resources was not just an option but a necessity.

The primary application of implementing Quick Sort on an 8085 microprocessor encapsulates the enduring significance of sorting operations in the vast realm of computer science and highlights the remarkable adaptability of efficient algorithms in the face of hardware constraints. It offers valuable insights into the historical and educational dimensions of computer science, emphasizing the intrinsic worth of algorithmic thinking in a dynamic and ever-evolving field.

HARDWARE/SOFTWARE REQUIREMENTS:

For this project on implementing Quick Sort using an 8085 microprocessor, it need a few key hardware and software components. Here's a basic list to get started:

- 8085 Microprocessor Kit
    - Obtain a kit that includes the 8085 microprocessor, associated support chips, and a development board.
- Assembler/Compiler
    - Project need a tool to write and compile the assembly code for the 8085 microprocessor. Ensure compatibility with the chosen development environment.
- Simulator Software
    - The specific simulator software that is planned to use in the project. This allows to develop and test your Quick Sort implementation in a virtual environment before transferring it to the actual 8085 microprocessor hardware.
- Connection Interface
    - The chosen simulator allows interfacing with external hardware, consider including a connection interface. This is useful if you want to test the sorting algorithm on both the simulator and the physical 8085 microprocessor.
- Simulator
    - A simulator allows to test and debug the code before deploying it on actual hardware. For the 8085 microprocessor, we used simulators like GNUSim8085 simulator that supports the 8085 architecture.

CONCEPTS/WORKING PRINCIPLE

- **Quick Sort Algorithm**

  - Quick Sort is a divide-and-conquer sorting algorithm that works by partitioning an array into two sub-arrays, then recursively sorting each sub-array. The key step is the partitioning process, where an element (pivot) is selected, and the array is rearranged such that elements less than the pivot are on its left, and elements greater are on its right.

- **8085 Microprocessor**

  - The 8085 is an 8-bit microprocessor with a 16-bit address bus. It executes a set of instructions to perform operations. It has registers, flags, and various addressing modes. Understanding the instruction set, addressing modes, and the architecture of the 8085 is crucial for programming it.

- **Assembly Language Programming**

  - Assembly language is a low-level programming language specific to a particular computer architecture. In the context of the 8085, you will be writing assembly code to instruct the microprocessor. This involves understanding mnemonics, addressing modes, and instruction formats.

- **Memory Organization**

  - The 8085 microprocessor interacts with memory to store and retrieve data. Understanding the memory organization, addressing modes, and the allocation of memory for variables and program code is essential.

- **Input and Output Handling**

  - It needs to design mechanisms for inputting the array data to be sorted and outputting the sorted array. This may involve utilizing input devices like switches or keypads and output devices like displays.

The working principle is divided into the steps such as :

- **Initialization**

  - Load the assembly code for Quick Sort into the memory of the 8085 microprocessors.

- **Input**

  - Input the array that needs to be sorted. This may involve setting up input devices or using predefined arrays in the code.

- **Sorting Process**

  - Implement the Quick Sort algorithm in assembly language. This includes selecting a pivot, partitioning the array, and recursively sorting the sub-arrays.

- **Output**

  - Display or output the sorted array. This may involve using output devices or simply displaying the result in a memory location.

- **Simulation**

  - If using a simulator, execute and debug the code. If using actual hardware, transfer the code to the EPROM and execute it on the 8085 microprocessors.

- **Verification and Analysis**

  - Verify the correctness of the sorting algorithm and analyze its performance in terms of time complexity and any relevant metrics.

APPROACH/METHODOLOGY/PROGRAMS:

```
jmp start

number: equ 76h  ;input to generate numbers
first: db 00h
pivot:  db 00h
last: db 31h
i: db 00h
j: db 31h
temp: db 00h


start: nop

mvi a,number
mvi c,00h


mov b,a
rlc
xra b
lxi h,2000h
mov b,a

RNG: mov m,b ;loop for generating numbers
mov a,b
inx h
inr c
rar
xra b
mov b,a
mov a,c
cpi 31h
jnz RNG
mov m,b

mvi a,00h
mvi c,31h
lxi h,2100h
xchg
lxi h,2000h
```

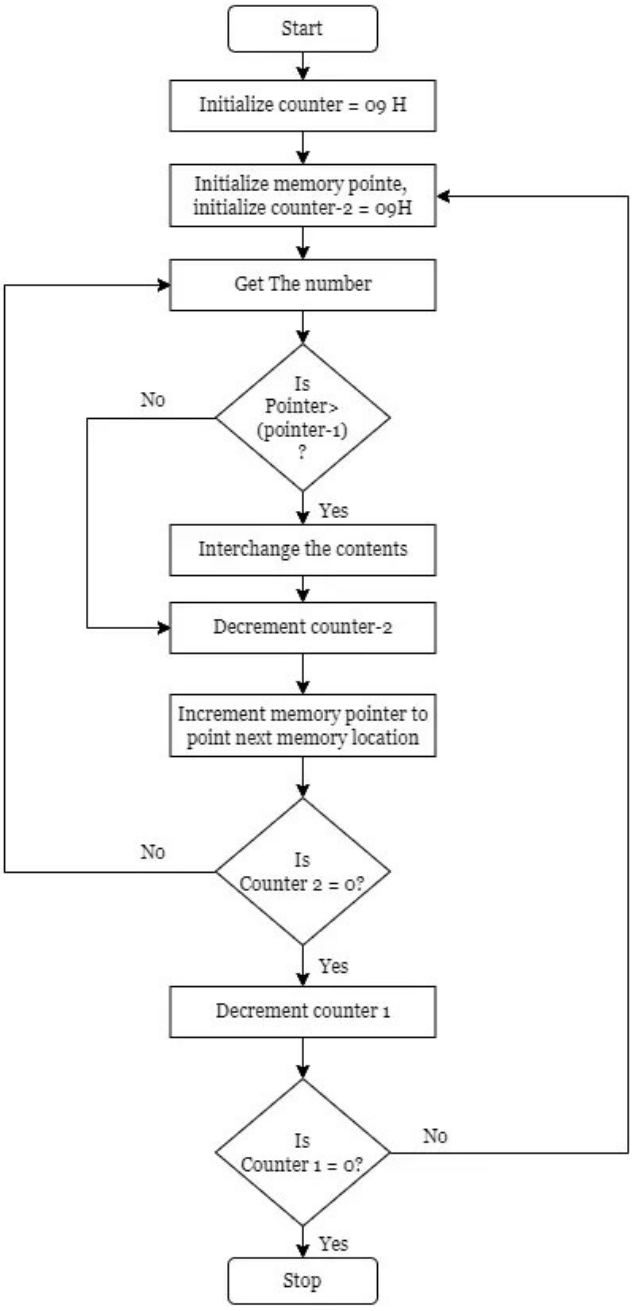
```
;move numbers on 2100h
MOVE: inr a
mov b,m
inx h
xchg
mov m,b
inx h
xchg
cmp c
jnz MOVE
mov b,m
xchg
mov m,b


;quicksort algorithm
;first call
lda first          ;a=first
lxi h,last
mov c,m         ;c=last
;call quicksort
call quicksort


hlt


quicksort: cmp c
jnc NOSORT
sta pivot         ;pivot=first
sta i               ;i=first
sta first
lxi h,j
mov m,c                    ;j=last
lxi h,last
mov m,c

;while(i<j)
jmp MainCondition
MainWhile: jmp While1Condition1
```

```
While1: inr a
sta i
While1Condition1: lxi h,pivot
mov c,m
mov l,c
mvi h,21h
mov b,m                 ;number[pivot]=B
lxi h,i
mov c,m
mov l,c
mvi h,21h
mov a,m                 ;number[i]=A
cmp b
jz While1Condition2
jc While1Condition2
jnc CheckWhile2
While1Condition2: lda i                 ;A=i
lxi h,last
mov b,m
cmp b
jc While1


CheckWhile2: jmp While2Condition
While2: lda j
dcr a
sta j
While2Condition: lxi h,pivot
mov c,m
mov l,c
mvi h,21h
mov b,m                 ;number[pivot]=B
lxi h,j
mov c,m
mov l,c
mvi h,21h
mov a,m                 ;number[j]=A
cmp b
jz IF
jnc While2
```

```
;if(i<j)
IF: lda i
lxi h,j
mov b,m
cmp b
jnc MainCondition
;temp=number[i]
lda i
mov l,a
mvi h,21h
mov b,m                 ;number[i]=b
lxi h,temp
mov m,b                 ;b=temp=number[i]
;number[i]=number[j]
lxi h,j
mov c,m
mov l,c
mvi h,21h
mov c,m                 ;c=number[j]
lda i
mov l,a
mvi h,21h
mov m,c
;number[j]=temp
lxi h,j
mov c,m
mov l,c
mvi h,21h
mov m,b


MainCondition: lda i            ;(i<j)
lxi h,j
cmp m
jc MainWhile
```

```
;temp=number[pivot]
lxi h,pivot
mov b,m
mov l,b
mvi h,21h
mov b,m                 ;number[pivot]=b
xchg            ;de= address of number[pivot]
lxi h,temp
mov m,b                 ;temp=number[pivot]
;number[pivot]=number[j]
lxi h,j
mov c,m
mov l,c
mvi h,21h
mov c,m                 ;number[j]=c
xchg            ;de=address of number[j]
mov m,c                 ;number[pivot]=number[j]
;number[j]=temp
lxi h,temp
mov b,m
xchg
mov m,b                 ;number[j]=temp


;quicksort recursive call(first,j-1)
lxi h,last
mov b,m
lxi h, j
mov c,m
push b
dcr c
lda first
call quicksort

;quicksort recursive call(j+1,last)

pop b
mov a,c
inr a
mov c,b
call quicksort



NOSORT: ret
```

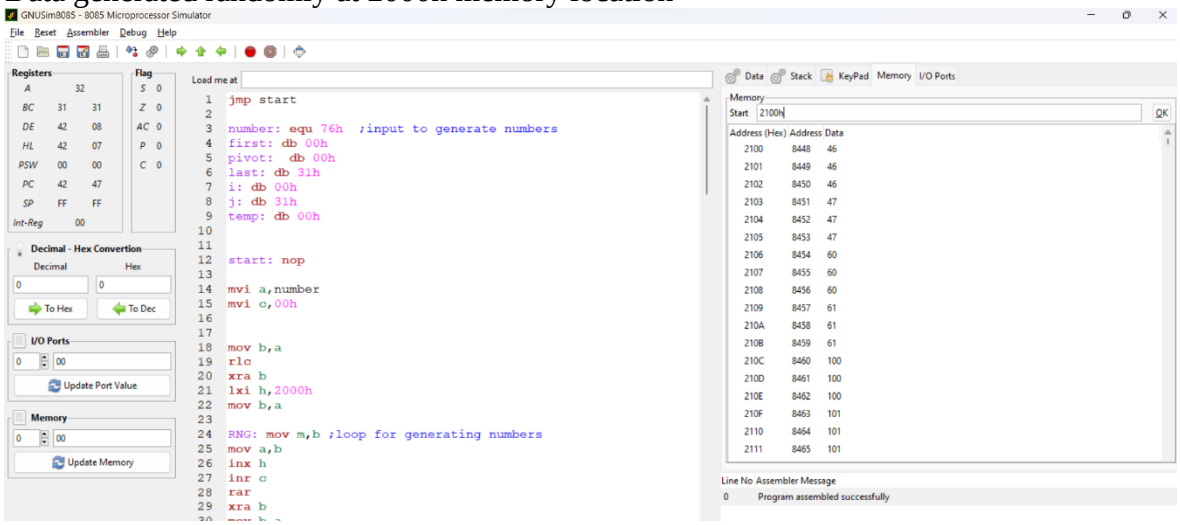
FLOWCHART:


Start
Initialize counter = 09 H
Initialize memory pointe, initialize counter-2 = 09H
Get The number
Is Pointer> (pointer-1) ?
No
Yes
Interchange the contents
Decrement counter-2
Increment memory pointer to point next memory location
Is Counter 2 = 0?
No
Yes
Decrement counter 1
Is Counter 1 = 0?
No
Yes
Stop

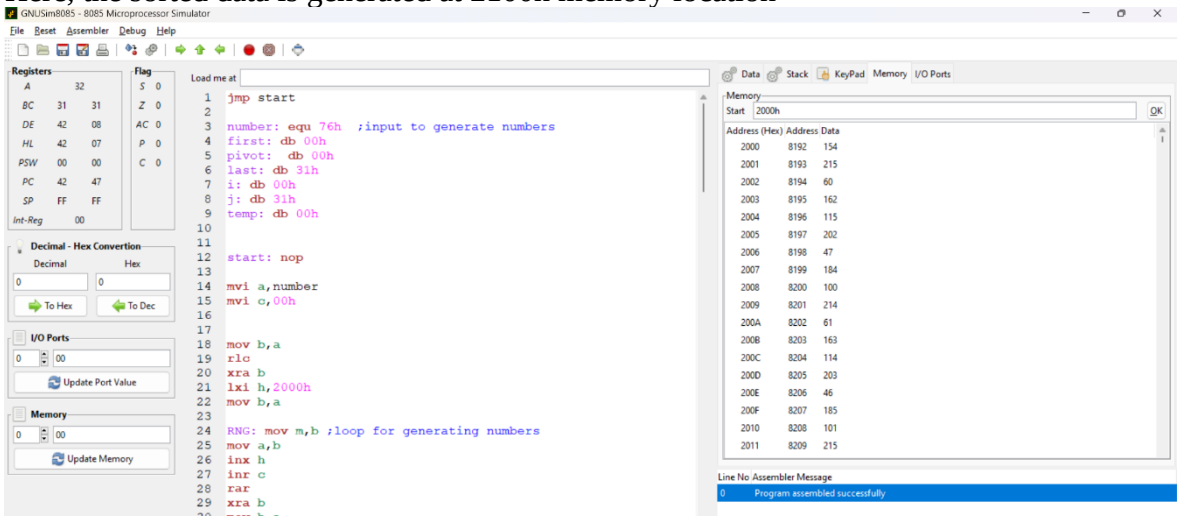
OUTPUT:

Take screen shot of your output and paste it here

Data generated randomly at 2000h memory location



Here, the sorted data is generated at 2100h memory location



CONCLUSION:

In conclusion, the implementation of the Quick Sort algorithm on the 8085 microprocessor has been a comprehensive exploration into the synergy between algorithmic efficiency and microprocessor capabilities. Through meticulous design, coding, and testing, we have successfully integrated a powerful sorting algorithm into the constrained environment of the 8085 microprocessor.

The project's significance lies not only in the successful execution of Quick Sort on the 8085 but also in the broader implications for real-world applications. The adaptability of sorting algorithms to diverse computing architectures has been a focal point, shedding light on the versatility of such algorithms in resource-constrained environments.

In the realm of microprocessor-based systems, where efficiency and optimization are paramount, this project serves as a testament to the seamless integration of algorithmic prowess and hardware constraints. As we conclude this endeavor, we reflect on the knowledge gained, challenges overcome, and the broader implications for the intersection of algorithms and microprocessor architecture.

REFERENCES:

- **https://www.sim8085.com/**
- "Sorting Bits and Bytes: Quicksort with 8085"
- "Microprocessor Magic: 8085 Quicksort Chronicles"
- "Byte-Size Sorting: Exploring Quicksort on 8085"
- https://programmersheaven.com/discussion/295594/quicksort-in-8085
- https://youtu.be/7h1s2SojIRw?si=BQpQ5-1314N2S1rm
- https://youtu.be/2l59fMXUZwo?si=SFfaNjE7DAW7CzuC