<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The 8085 microprocessor</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 8085 Internal Architecture</td>
<td>1</td>
</tr>
<tr>
<td>1.3 8085 Pin Diagram</td>
<td>6</td>
</tr>
<tr>
<td>1.4 8085 Addressing Modes</td>
<td>11</td>
</tr>
<tr>
<td>2. The 8085 Microprocessor Trainer Kit</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Specifications of STS -85 VLC</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Hardware Specifications</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2 Software Description</td>
<td>15</td>
</tr>
<tr>
<td>3. 8085 Instruction Set Summary</td>
<td>20</td>
</tr>
<tr>
<td>4. Entering a program and Execution procedure</td>
<td>24</td>
</tr>
<tr>
<td>5. A Sample Program</td>
<td>25</td>
</tr>
<tr>
<td>6. 8085 Instructions and Mnemonic codes</td>
<td>26</td>
</tr>
<tr>
<td>7. Lab Exercises</td>
<td>28</td>
</tr>
<tr>
<td>7.1 Subtract two 8 bit numbers</td>
<td>28</td>
</tr>
<tr>
<td>7.2 Divide two 8 bit numbers</td>
<td>29</td>
</tr>
<tr>
<td>7.3 Palindrome checking</td>
<td>30</td>
</tr>
<tr>
<td>7.4 Sort in ascending order</td>
<td>32</td>
</tr>
<tr>
<td>7.5 Sort in descending order</td>
<td>34</td>
</tr>
<tr>
<td>7.6 Add two 16 bit numbers</td>
<td>36</td>
</tr>
<tr>
<td>7.7 Convert a BCD number to a binary number</td>
<td>38</td>
</tr>
<tr>
<td>7.8 Convert a binary number to a BCD number</td>
<td>40</td>
</tr>
<tr>
<td>7.9 Add ten 8 bit numbers</td>
<td>41</td>
</tr>
<tr>
<td>7.10 Multiply two 8 bit numbers</td>
<td>43</td>
</tr>
<tr>
<td>7.11 Largest of ten 8 bit numbers</td>
<td>45</td>
</tr>
<tr>
<td>7.12 Smallest of ten 8 bit numbers</td>
<td>47</td>
</tr>
<tr>
<td>7.13 Scroll a set of data in memory</td>
<td>49</td>
</tr>
</tbody>
</table>
1. THE 8085 MICROPROCESSOR

1.1 Introduction

The 8085 microprocessor was made by Intel in mid 1970s. It was binary compatible with 8080 microprocessor but required less supporting hardware thus leading to less expensive microprocessor systems. It is a general purpose microprocessor capable of addressing 64k of memory. The device has 40 pins, require a +5V power supply and can operate with 3 MHz single phase clock. It has also a separate address space for up to 256 I/O ports. The instruction set is backward compatible with its predecessor 8080 even though they are not pin-compatible.

1.2 8085 Internal Architecture (Fig: 1)
The 8085 has a 16 bit address bus which enables it to address 64 KB of memory, a data bus 8 bit wide and control buses that carry essential signals for various operations. It also has a built in register array which are usually labelled A(Accumulator), B, C, D, E, H, and L. Further special-purpose registers are the 16-bit Program Counter (PC), Stack Pointer (SP), and 8-bit flag register F. The microprocessor has three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one Non-Maskable interrupt (TRAP), and one externally serviced interrupt (INTR). The RST n.5 interrupts refer to actual pins on the processor a feature which permitted simple systems to avoid the cost of a separate interrupt controller chip.

**Control Unit**

Generates signals within microprocessor to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the processor be opened or closed, so that data goes where it is required, and so that ALU operations occur.

**Arithmetic Logic Unit**

The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’, etc. Uses data from memory and from Accumulator to perform arithmetic and always stores the result of operation in the Accumulator.

**Registers**

The 8085 microprocessor includes six registers, one accumulator, and one flag register, as shown in Fig 1. In addition, it has two 16-bit registers: the stack pointer and the program counter. The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in Fig 1. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.
**Accumulator**

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

**Flag Registers**

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

**Program Counter (PC)**

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

**Stack Pointer (SP)**

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

**Instruction Register / Decoder**

This is a temporary store for the current instruction of a program. Latest instruction is sent to here from memory prior to execution. Decoder then takes instruction and ‘decodes’ or interprets the instruction. Decoded instruction is then passed to next stage.
Memory Address Register (MAR)

Holds addresses received from PC for eg: of next program instruction. MAR feeds the address bus with address of the location of the program under execution.

Control Generator

Generates signals within microprocessor to carry out the instruction which has been decoded. In reality it causes certain connections between blocks of the processor to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

8085 System Bus

The microprocessor performs four operations primarily.

- Memory Read
- Memory Write
- I/O Read
- I/O Write

All these operations are part of the communication processes between microprocessor and peripheral devices. The 8085 performs these operations using three sets of communication lines called buses - the address bus, the data bus and the control bus.

Address Bus

The address bus is a group of 16 lines. The address bus is unidirectional: bits flow only in one direction – from the 8085 to the peripheral devices. The microprocessor uses the address bus to perform the first function: identifying a peripheral or memory location. Each peripheral or memory location is identified by a 16 bit address. The 8085 with its 16 lines is capable of addressing 64 K memory locations.
**Data Bus**

The data bus is a group of eight lines used for dataflow. They are bidirectional: data flows in both direction between the 8085 and memory and peripheral devices. The 8 lines enable the microprocessor to manipulate 8-bit data ranging from 00 to FF.

**Control Bus**

The control bus consists of various single lines that carry synchronization signals. These are not groups of lines like address of data bus but individual lines that provide a pulse to indicate an operation. The 8085 generates specific control signal for each operation it performs. These signals are used to identify a device type which the processor intends to communicate.
1.3 8085 Pin Diagram (Fig: 2)

--- X1 | 1
--- X2 | 2

--- RESET OUT | 3
--- SOD | 4
--- S1D | 5

--- TRAP | 6
--- RST 7.5 | 7
--- RST 6.5 | 8
--- RST 5.5 | 9

--- INTR | 10
--- INTA | 11
--- ADO | 12
--- AD1 | 13
--- AD2 | 14
--- AD3 | 15
--- AD4 | 16
--- AD5 | 17
--- AD6 | 18
--- AD7 | 19

--- 8085A | 31
--- WR | 31
--- ALE | 30
--- S0 | 29
--- A15 | 28
--- A14 | 27
--- A13 | 26
--- A12 | 25
--- A11 | 24
--- A10 | 23
--- A9 | 22

--- VCC (5V) | 40
--- HOLD | 39
--- HLDA | 36
--- CLK (OUT) | 37
--- RESET IN | 36
--- READY | 35
--- IO/| 34
--- S1 | 33
--- RD | 32

--- (Gnd) VSS | 20
--- A8 | 21
8085 Pin Description

Properties
- Single + 5V Supply
- 4 Vectored Interrupts (One is Non Maskable)
- Serial In/Serial Out Port
- Decimal, Binary, and Double Precision Arithmetic
- Direct Addressing Capability to 64K bytes of memory

A6-A1 (Output 3 states)
Address Bus; The most significant 8 bits of the memory address or the 8 bits of the I/O address; 3 stated during Hold and Halt modes.

AD0 - 7 (Input/Output 3state)
Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output)
Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3 stated.

SO, S1 (Output)
Data Bus Status: Encoded status of the bus cycle:

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>HALT</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WRITE</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>READ</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>FETCH</td>
</tr>
</tbody>
</table>

S1 can be used as an advanced R/W status.
RD (Output 3state)

READ; indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

WR (Output 3state)

WRITE; indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3 stated during Hold and Halt modes.

READY (Input)

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

HOLD (Input)

HOLD; indicates that another Master is requesting the use of the address and data buses. The CPU, upon receiving the Hold request, will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output)

HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input)

INTERRUPT REQUEST; is used as a general purpose interrupt. It is sampled only using the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt
service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

**INTA (Output)**

INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5
RST 6.5 - (Inputs)
RST 7.5

RESTART INTERRUPTS; These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 ← Highest Priority
RST 6.5
RST 5.5 o Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

**TRAP (Input)**

Trap interrupt is a nonmaskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

**RESET IN (Input)**

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected. The CPU is held in the reset condition as long as Reset is applied.

**RESET OUT (Output)**

Indicates CPIJ is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.
**XI, X2 (Input)**
Crystals or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

**CLK (Output)**
Clock Output for use as a system clock when a crystal or R/C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

**IO/M (Output)**
IO/M indicates whether the Read/Write is to memory or I/O Tristated during Hold and Halt modes.

**SID (Input)**
Serial input data line: The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

**SOD (output)**
Serial output data line: The output SOD is set or reset as specified by the SIM instruction.

**Vcc**
+5 volt supply.

**Vss**
Ground Reference
1.4 8085 Addressing modes

They are mainly classified into four:

- Immediate addressing.
- Register addressing.
- Direct addressing.
- Indirect addressing.

**Immediate addressing**

Data is present in the instruction. Load the immediate data to the destination provided.

Example: MVI R, data

**Register addressing**

Data is provided through the registers.

Example: MOV Rd, Rs

**Direct addressing**

Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.

Example: IN 00H or OUT 01H

**Indirect Addressing**

This means that the Effective Address is calculated by the processor and the contents of the address (and the one following) are used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.
2. The 8085 MICROPROCESSOR TRAINER KIT

2.1 Introduction

From the 4 bit microprocessor brought out by Intel in 1971, advancement in technology have been made and now 8 bit, 16 bit, 32 bit and 64 bit microprocessors are available and 64 bit and 32 bit microprocessors are dominating the market. From the age of vacuum tubes and transistors, we are now in the age of microprocessors. Due to its adoptability and intelligence, they are used extensively. The trainer kit is a low cost 8085 based training tool developed specifically for learning the operation of today's microprocessor based systems.

The purpose of a microprocessor trainer kit is to:

- Facilitate people in learning and using machine code programming.
- Familiarise people of the basic microprocessor hardware.
- Facilitate testing of hand coded programs with the help of break point setting.
- Facilitate easy interface to external I/O devices.
- Facilitate training with its peripherals also.

STS-85 is a low cost microprocessor trainer kit and it is based on the most popular 8 bit microprocessor INTEL 8085. It has 6 digits of seven segment display and 24 keys keyboard. STS – 85 VLC is supplied with minimum basic hardware required to function as a trainer. User can add additional memory on board.

2.2 Specifications of STS –85 VLC

The specifications of STS-85 VLC low cost 8085 microprocessor trainer kit are:

**Hardware**

- CPU : 8085A with 3.579545 MHz/ 4.000MHz/6.144Mhz with no wait states.
- Memory : 8 k EPROM (2764)
  - 8 K RAM (6264)
  - 2 Expansion socket space.
- Key board : 24 keys keyboard
- Display : 6 digit bright seven segment display
- Peripherals Used : 8279, 8255 for user
- Terminations : The data, address and control lines and programmable I/O lines of the peripherals are terminated in different connectors.
Software

(a) Program entry

- Address field is 4 digits wide for the 16 bit address of 8085A. Data field is 2 digits wide for 8 bit data of 8085A.
- Set address and verify the contents of present location.
- Data is set altered in memory immediately for each hex key entry.
- Automatic setting of RAM starts address on pressing INR or DCR key.
- Increment/decrement the address and display /alter the data contents.

(b) Program execution

- Program could be executed from any address using GO key with or without break point. User can use the display routine available in the monitor to display their data.
- Execution with break point with break count facility for easy software debugging.
- Facility to display/edit the registers of user registers saved while break.

(c) Program editing

- Move block of data in the RAM.
- Fill block of memory with constant data in the RAM.
- Facility to alter any location of RAM in data set mode.

(c) Miscellaneous

- Memory test facility for RAM.
- Check sum facility for ROM in the same memory test function.

2.2.1 Hardware Specifications

The main CPU PCB has the facility to accommodate one 8255A (programmable peripheral interface), one 8279 (key board display controller) and 4 numbers 28 pin memory IC sockets. It has facility to terminate data, address and control signals on different connectors. 26 pins FRC connector is provided for using 8255 ports by the user for performing experiments with interface boards. STS-85-VLC require a single 5 volts supply for its operation.
8085A CPU

STS-85-VLC is based on Intel’s 8 bit microprocessor 8085A. It has a crystal connected to it with a frequency as specified under specifications. The READY line of the CPU is pulled up to operate the CPU without any wait states. Display refreshing and keyboard debouncing are performed in hardware by onboard 8279 IC. The lower order address bus is derived from an octal latch using ALE (Address Latch Enable) signal of 8085A. The data, higher order address and control buses are terminated in various connector points.

Memory and I/O Decoding

At power on or reset 8085A address bus has an address of 0000H and hence the memory has been so decoded to have EPROM address 0000H to 1FFF-H (2764). The user RAM address is 8000H to 9EFF-H. STS-85-VLC is using single decoder IC for memory and I/O. STS-85-VLC when supplied has 16K chip selects. The socket next to monitor EPROM can accommodate another RAM. 74LS 138 next to 8085A acts as decoder. Out of 8 of its outputs 4 are used for memory decoding and other 4 are used for I/O decoding. As only 2 I/O chips are used in the system other 2 decoded I/O chip selects are available for the user.

Keyboard and display

The keyboard of STS-85-VLC consists of 24 keys and out of which 23 keys are scanned by 8279. RESET key is connected to CPU reset through standard RC network 16 keys are devoted for hexadecimal data entry and the rest are function keys. The display of STS-85-VLC is 6 digit common anode display. Left most 4 digits of the display represents the address and right most 2 digits represents the data at the displayed address. One 8279 (keyboard display controlled) is used for display refresh and keyboard scanning purposes. 74LS138 decoder drives digit driver transistors on display board to refresh display. The scanning and debounce for a key closure is done by 8279 and it uses another 74LS138 decoder to decode the scan lines. The interrupt output of 8279 is connected to RST 7.5 input of 8085A CPU.

Port lines

One number 8255A (programmable peripheral interface) is provided on board. All the 24 port lines are terminated on 26 pin FRC connector along with VCC and GND. Port line PC7 is connected to TRAP of 8085 A CPU for future use. The system will function without this IC also.

Power Supply

Power supply for the trainer is inbuilt / provided externally as required by user. Only logic supply is provided.
**Enhanced STS-85 VLC features**

In enhanced model of STS-85-VLC battery backup circuit and one 8155 (timer, RAM, I/O) IC is available. The RAM of 8155 is not accessible as the chip select input is given from I/O decoder IC. The ‘timer in’ is from processor ‘clock out’ and ‘timer out’ goes to 8085 RST 7.5 input. The existing track from 8279 ‘intr’ has already been disconnected.

The port address of various peripheral ICs are
a. 8255- 00 to 03
b. 8279- 88H to 89H
c. 8155- 0F0H to 0F5H

**2.2.2 Software description**

STS-85-VLC has monitor software which controls the keyboard and display function. The software facilities available in STS-85-VLC include Address set, Increment, Decrement, Data set, Break set, Break clear, Move, Fill, Go(Execute) with and without break, Memory test and Register display/alter functions.

**System Initialization**

At ‘power on’ or at manual reset 8085A starts executing the program loaded in 0000H which is the starting address of the monitor EPROM. At this start address the monitor software performs initialization of the microprocessor stack pointer, monitor software flags, control/command registers of 8279 for refreshing the display & scanning the keyboard. Then, the display is written to display the power on prompt one character after the other.

After completing initialization, the system waits for a keyboard entry and is ready to receive address or function commands. All the hexadecimal keys are entered into the address field. To enter the data mode, increment (INR) or decrement (DCR) keys should be pressed.

**Address set and alter the contents**

On ‘power on’ or manual reset, STS-85 is displayed. Now to set an address use the hexadecimal keys. Hex key entered is placed in the right most column of the address field after scrolling the existing entry and data at that address is displayed. You will be entering keys in the address filed only, until you press INR or DCR key. Once you press this, then the hex keys entered will change the data field and also update the memory contents immediately.
Increment/Decrement

In data set mode, the function of INR/CDR key is to increment/decrement the address field and update the display with the corresponding data. In address set mode INR/DCR key should be pressed to enter into data set mode. In data set mode, hex keys are entered in the data field of the display and corresponding memory location is updated automatically.

Hex key entered is entered as LSD (least significant digit) in the respective field after scrolling the field left by one digit.

Display/Alter CPU Registers

You can display and alter the user register values by entering REG mode by pressing REG key, you enter REG mode.

Data in the register pair 551FAF. name of the register pair

The dot refers to the register under alteration.

Now, if you want to change F, Enter the new value and the user save area is updated automatically. To go to next register use INR key and the dot shifts over to A and A value can be altered. One more INR will display BC register in the same way. The display format is the same for DE and HL registers. Next register in the sequence is PC and the display format is as shown below.

8000PC

For PC and SP display the dot is having no relevance as they are 16 bit registers and to alter the register value, required hex keys are entered. There is no necessity for using INR key for transferring the displayed data into user save area. Also, by continuously pressing INR/DCR key, you will be looping around in the register mode only. The REG key is of toggle type and by pressing it in REG mode, the system enters the power on prompt.

Execute/RAM test and Checksum

A program entered in RAM could be executed by pressing GO key. When this key is pressed, the display is as shown below

Go Fn

And within short time comes back and displays user PC as, 8000 ch.

This means that the present user PC value is 8000H and this is to be checked or changed. If the user program starting address is different it could be entered. After entering the correct PC value, INR key is depressed. When this is done, the monitor desaves all the user registers and then the program jumps to user program. MTEST is the alternate function of GO key. The purpose of this key is to perform non destructive memory test for RAM and display the result. For ROM, the checksum from start to end
is computed and displayed. The monitor checks the first byte of memory block and finds out whether it is ROM or RAM. In case of ROM, checksum is computed and displayed. In case of RAM, writing and reading is effected to know whether that RAM is okay or not. If the memory block exceeds the RAM area display indicates the same.

To initiate any ALTernate function press ALT key, the system displays,

AL T Fn

Then press the relevant key the second function will be initiated. Here in this case the display shows after pressing MTEST key as,

Mtest

For a while and then prompts,

00 -S start address

The start address is entered using hex keys and INR key is pressed.

0000 -E end address

Now end address is entered and when INR key is pressed. Memory test/check sum is performed and the result is displayed.

Set/Clear Break Point

Break point setting helps to set break point in any program. It is a valuable debugging tool. When Br SET key is pressed, the break address is asked for after displaying the function name as shown below.

Break
8002 bA Break Address

Now, break address should be entered and then INR key is pressed. Then the system asks for the break count.

0010 bC Break Count

When this is also entered and INR key is pressed, break has been set.

BrSet

Break point set in ROM will be ignored while execution. When the break address is reached after the required number of break count, break is cleared, registers are all saved and the system enters the register mode displaying PC Monitor uses RST 5 for performing break. The alternate function of the same key is BrCLR. By pressing this, previously set break is cleared and the display shows,

Br C l r

Block Transfer, Fill

Block transfer/MOVE function is helpful in moving a block of data from one memory area to another. The user has to enter the start address of the source, end address of the source and also the start address of the destination. When the end address entered is less
than the start address the system does not accept the end address when INR key is pressed. The system proceeds further when the end address is greater than start address.

When MOVE is pressed, the system displays

MoveFn

For a while and then displays,

0000  -S start address

The start address is entered using hex keys and INR key is pressed.

0000  -E end address

Now end address is entered and when INR key is pressed, the destination address is displayed

0000  -d destination address

The destination address if required could be changed and when INR key is pressed ‘move’ function is executed. While entering destination start address, the user can verify and alter start and end addresses of source by pressing DCR key without affecting any values. The ‘move’ function takes care of overlapping memories and moves properly.

FILL is the alternate function of MOVE key. This function is for filling an area of RAM with specified data. On pressing this key, prompt for fill is displayed. The starting address, end address and data to be filled are asked for and accepted. When ROM area is tried to be filled it stops there indicating the same.

Fill Fn

For a while and then displays,

8000  -S Start address
8200  -E End address
Data  -3E Fill data

When INR is pressed after entering fill data, the fill function is executed and the system enters the power on mode.

Enhanced STS-85-VLC feature.

In enhanced model of STS-85-VLC one additional software function is provided namely, ‘single step’. This is alternate function of REG key. This function is same as that of ‘GO’ key. Here instead of executing user program continuously only one instruction of user program is executed and the system returns to ‘REG’ mode in monitor displaying next instruction address. User registers are desaved before executing the user program and are saved after executing the user program one instruction pointed by the ‘PC’ register content.

Set the registers including PC as per your requirement. Press ALT key. Then press REG/STEP key. To proceed to next step press REG/STEP key again.

Utilities

Commonly used utilities are given under RST instruction, the single byte call instruction for quick reference and ease of use
RST 0  : Return to monitor
RST 1  : Scans keyboard until a key is received. The key value is returned in Accumulator and also in B register.
RST 2  : One scan of keyboard. If zero flag is set no key is pressed. Otherwise key value is available in accumulator and B register.

RST 3  : Display the data available in the display buffer 9F00 (MSD) to 9F05 (LSD). The data is put on display after converting through segment table. MS bit is placed onto decimal points. Other 7 bits map to segment table. 00 to 0F in display buffer is displayed as 0 to F on display. 10H onwards it continues with G, H, I…and so on.
Power ON location : 9F09/9F0A make this ‘zero’ to restore power on default values including user stack pointer.
User delay location : 9F37 (default value 5): If prompt messages are retaining for long or vanishing fast, set this to default value and press Reset
Important : Registers display will have user program register value only after a break or single step. If you want to view registers set your break point there (Break address should be beginning address of and instruction) and execute from beginning. Program should stop at the set break point to view registers.
### 3. 8085 INSTRUCTION SET SUMMARY

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV r1 r2</td>
<td>Move register to register</td>
<td>4</td>
</tr>
<tr>
<td>MOV M r</td>
<td>Move register to memory</td>
<td>7</td>
</tr>
<tr>
<td>MOV r M</td>
<td>Move memory to register</td>
<td>7</td>
</tr>
<tr>
<td>MVI r</td>
<td>Move immediate register</td>
<td>7</td>
</tr>
<tr>
<td>MVI M</td>
<td>Move immediate memory</td>
<td>10</td>
</tr>
<tr>
<td>LXI B</td>
<td>Load immediate register Pair B &amp; C</td>
<td>10</td>
</tr>
<tr>
<td>LXI D</td>
<td>Load immediate register Pair D &amp; E</td>
<td>10</td>
</tr>
<tr>
<td>LXI H</td>
<td>Load immediate register Pair H &amp; L</td>
<td>10</td>
</tr>
<tr>
<td>LXI SP</td>
<td>Load immediate stack pointer</td>
<td>10</td>
</tr>
<tr>
<td>STAX B</td>
<td>Store A indirect</td>
<td>7</td>
</tr>
<tr>
<td>STAX D</td>
<td>Store A indirect</td>
<td>7</td>
</tr>
<tr>
<td>LDAX B</td>
<td>Load A indirect</td>
<td>7</td>
</tr>
<tr>
<td>LDAX D</td>
<td>Load A indirect</td>
<td>7</td>
</tr>
<tr>
<td>STA</td>
<td>Store A direct</td>
<td>13</td>
</tr>
<tr>
<td>LDA</td>
<td>Load A direct</td>
<td>13</td>
</tr>
<tr>
<td>SHLD</td>
<td>Store H &amp; L direct</td>
<td>16</td>
</tr>
<tr>
<td>LHLD</td>
<td>Load H &amp; L direct</td>
<td>16</td>
</tr>
<tr>
<td>XCHG</td>
<td>Exchange D &amp; E H &amp; L registers</td>
<td>4</td>
</tr>
</tbody>
</table>

- **STACK OPS**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH B</td>
<td>Push register Pair B &amp; C on stack</td>
<td>12</td>
</tr>
<tr>
<td>PUSH D</td>
<td>Push register Pair D &amp; E on stack</td>
<td>12</td>
</tr>
<tr>
<td>PUSH H</td>
<td>Push register Pair H &amp; L on stack</td>
<td>12</td>
</tr>
<tr>
<td>PUSH PSW</td>
<td>Push A and Flags on stack</td>
<td>12</td>
</tr>
<tr>
<td>POP B</td>
<td>Pop register Pair B &amp; C off stack</td>
<td>10</td>
</tr>
<tr>
<td>POP D</td>
<td>Pop register Pair D &amp; E off stack</td>
<td>10</td>
</tr>
<tr>
<td>POP H</td>
<td>Pop register Pair H &amp; L off stack</td>
<td>10</td>
</tr>
<tr>
<td>POP PSW</td>
<td>Pop A and Flags off stack</td>
<td>10</td>
</tr>
<tr>
<td>XTHL</td>
<td>Exchange top of stack H &amp; L</td>
<td>16</td>
</tr>
<tr>
<td>SPHL</td>
<td>H &amp; L to stack pointer</td>
<td>6</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump unconditional</td>
<td>10</td>
</tr>
<tr>
<td>JC</td>
<td>Jump on carry</td>
<td>7/10</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump on no carry</td>
<td>7/10</td>
</tr>
<tr>
<td>JZ</td>
<td>Jump on zero</td>
<td>7/10</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump on no zero</td>
<td>7/10</td>
</tr>
<tr>
<td>JP</td>
<td>Jump on positive</td>
<td>7/10</td>
</tr>
</tbody>
</table>
JM       Jump on minus       7/10
JPE      Jump on parity even 7/10
JPO      Jump on parity odd  7/10
PCHL     H & L to program counter  6
CALL
CALL     Call unconditional   18
CC       Call on carry        9/18
CNC      Call on no carry     9/18
CZ       Call on zero         9/18
CNZ      Call on no zero      9/18
CP       Call on positive     9/18
CM       Call on minus        9/18
CPE      Call on parity even  9/18
CPO      Call on parity odd 9/18

•  RETURN

RET      Return                10
RC       Return on carry      6/12
RNC      Return on no carry   6/12
RZ       Return on zero       6/12
RNZ      Return on no zero    6/12
RP       Return on positive   6/12
RM       Return on minus      6/12
RPE      Return on parity even 6/12
RPO      Return on parity odd 6/12
RESTART
RST      Restart               12

•  INPUT/OUTPUT

IN       Input                  10
OUT      Output                 10

•  INCREMENT AND DECREMENT

INR r    Increment register    4
DCR r    Decrement register    4
INR M    Increment memory      10
DCR M    Decrement memory      10
INX B    Increment B & C registers 6
INX D    Increment D & E registers 6
INX H    Increment H & L registers 6
INX SP   Increment stack pointer 6
DCX B    Decrement B & C      6
DCX D  Decrement D & E                   6
DCX H  Decrement H & L                   6
DCX SP Decrement stack pointer                  6
ADD
ADD r  Add register to A                   4
ADC r  Add register to A with carry                  4
ADD M  Add memory to A                  7
ADC M  Add memory to A with carry                  7
ADI   Add immediate to A                  7
ACI   Add immediate to A with carry                  7
DAD B  Add B & C to H & L                 10
DAD D  Add D & E to H & L                 10
DAD H  Add H & L to H & L                 10
DAD SP Add stack pointer to H & L                10

•  SUBTRACT

SUB r  Subtract register from A                  4
SBB r  Subtract register from A with borrow             4
SUB M  Subtract memory from A                  7
SBB M  Subtract memory from A with borrow             7
SUI   Subtract immediate from A                  7
SBI   Subtract immediate from A with borrow             7

•  LOGICAL

ANA r  Add register with A                  4
XRA r  Exclusive Or register with A                 4
ORA r  Or register with A                  4
CMP r  Compare register with A                  4
ANA M  And memory with A                  7
XRA M  Exclusive Or Memory with A                 7
ORA M  Or memory with A                  7
CMP M  Compare memory with A                  7
ANI   And immediate with A                  7
XRI   Exclusive Or immediate with A                 7
ORI   Or immediate with A                  7
CPI   Compare immediate with A                 7

•  ROTATE

RLC   Rotate A left                        4
RRC   Rotate A right                       4
RAL   Rotate A left through carry             4
RAR   Rotate A right through carry             4
**SPECIALS**

- CMA  Complement A  4
- STC  Set carry  4
- CMC  Complement carry  4
- DAA  Decimal adjust A  4

**CONTROL**

- EI  Enable Interrupts  4
- DI  Disable Interrupts  4
- NOP  No-operation  4
- HLT  Halt (Power down)  5
- RIM  Read Interrupt Mask  4
- SIM  Set Interrupt Mask  4
4. ENTERING A PROGRAM AND EXECUTION PROCEDURE

After connecting it to the power supply, STS-85 prompt appears. Now press INR. Address field shows 8000 and directly you can start entering data corresponding to first address.
Note: In case you select any other starting address, say 8050, then after entering 8050 press INR then only system changes from address field to data field.
- Press INR, the address shifts to 8001.
- Feed 2\textsuperscript{nd} data and then INR and so on.
On completion of the programs, press INR.

Execution procedure:

- Press GO
- System shows some address and ‘Ch’ in data field.
- Now select the starting address of the program which is stored for execution say 8000
- After entering the starting address press INR
- The program will be executed.

Note: In case of addition of two BCD numbers after pressing INR, the initial prompt appears. To refer the output result goes to the address field specified in the program (say 8051). There you can see the added result. For other programs like binding display etc you can see the display blinking directly.
5. A SAMPLE PROGRAM

Aim: To multiply two 8 bit numbers.

Program Analysis: Two 8 bit numbers are stored in memory locations 8100 and 8101. They are multiplied and the results are stored in memory locations 8200 and 8201.

Program:

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Machine code</th>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>AF</td>
<td></td>
<td>XRA A</td>
<td></td>
<td>Clear A</td>
</tr>
<tr>
<td>8001</td>
<td>A8</td>
<td></td>
<td>XRA B</td>
<td></td>
<td>Clear B</td>
</tr>
<tr>
<td>8002</td>
<td>A9</td>
<td></td>
<td>XRA C</td>
<td></td>
<td>Clear C</td>
</tr>
<tr>
<td>8003</td>
<td>21</td>
<td></td>
<td>LXI H</td>
<td>8100</td>
<td>Set HL pair as an index to source memory</td>
</tr>
<tr>
<td>8004</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8005</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8006</td>
<td>46</td>
<td></td>
<td>MOV B, M</td>
<td></td>
<td>Move [M] to B</td>
</tr>
<tr>
<td>8007</td>
<td>23</td>
<td></td>
<td>INX H</td>
<td></td>
<td>Increment HL pair</td>
</tr>
<tr>
<td>8008</td>
<td>86</td>
<td></td>
<td>ADD M</td>
<td></td>
<td>Add [A] to [M]</td>
</tr>
<tr>
<td>8009</td>
<td>D2</td>
<td></td>
<td>JNC</td>
<td>L1</td>
<td>Jump if no carry to L1</td>
</tr>
<tr>
<td>800A</td>
<td>0D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800B</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800C</td>
<td>0C</td>
<td></td>
<td>INR C</td>
<td></td>
<td>Increment [C]</td>
</tr>
<tr>
<td>800D</td>
<td>05</td>
<td></td>
<td>L1</td>
<td>DCR B</td>
<td>Decrement [B]</td>
</tr>
<tr>
<td>800E</td>
<td>C2</td>
<td></td>
<td>JNZ</td>
<td>L2</td>
<td>Jump if nonzero to L2</td>
</tr>
<tr>
<td>800F</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8010</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8011</td>
<td>32</td>
<td></td>
<td>STA</td>
<td>8200</td>
<td>Store [A] in 8200</td>
</tr>
<tr>
<td>8012</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8013</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8014</td>
<td>79</td>
<td></td>
<td>MOV A, C</td>
<td></td>
<td>Move [C] to A</td>
</tr>
<tr>
<td>8015</td>
<td>32</td>
<td></td>
<td>STA</td>
<td>8201</td>
<td>Store [A] in memory location 8201</td>
</tr>
<tr>
<td>8016</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8017</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8018</td>
<td>76</td>
<td></td>
<td>HLT</td>
<td></td>
<td>Stop program</td>
</tr>
</tbody>
</table>

Result: The program is executed and the results are stored in the memory locations 8200 and 8201.

Input:  At 8100 :03
        At 8101 :02
Output: At 8200 : 06
        At 8201 : 00
### 6. 8085 INSTRUCTIONS AND MNEMONIC CODES

<table>
<thead>
<tr>
<th>Hex</th>
<th>mnemonic</th>
<th>Hex</th>
<th>mnemonic</th>
<th>Hex</th>
<th>mnemonic</th>
<th>Hex</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE</td>
<td>ACI 8-Bit</td>
<td>3F</td>
<td>CMC</td>
<td>2B</td>
<td>DCX H</td>
<td>01</td>
<td>LXI B,16-Bit</td>
</tr>
<tr>
<td>8F</td>
<td>ADC A</td>
<td>BF</td>
<td>CMP A</td>
<td>3B</td>
<td>DCX SP</td>
<td>11</td>
<td>LXI D,16-Bit</td>
</tr>
<tr>
<td>88</td>
<td>ADC B</td>
<td>B8</td>
<td>CMP B</td>
<td>F3</td>
<td>DI</td>
<td>21</td>
<td>LXI H,16-Bit</td>
</tr>
<tr>
<td>89</td>
<td>ADC C</td>
<td>B9</td>
<td>CMP C</td>
<td>FB</td>
<td>EI</td>
<td>31</td>
<td>LXI SP,16-Bit</td>
</tr>
<tr>
<td>8A</td>
<td>ADC D</td>
<td>BA</td>
<td>CMP D</td>
<td>76</td>
<td>HLT</td>
<td>7F</td>
<td>MOV A A</td>
</tr>
<tr>
<td>8B</td>
<td>ADC E</td>
<td>BB</td>
<td>CMP E</td>
<td>DB</td>
<td>IN 8-Bit</td>
<td>78</td>
<td>MOV A B</td>
</tr>
<tr>
<td>8C</td>
<td>ADC H</td>
<td>BC</td>
<td>CMP H</td>
<td>3C</td>
<td>INR A</td>
<td>79</td>
<td>MOV A C</td>
</tr>
<tr>
<td>8D</td>
<td>ADC L</td>
<td>BD</td>
<td>CMP</td>
<td>04</td>
<td>INR B</td>
<td>7A</td>
<td>MOV A D</td>
</tr>
<tr>
<td>8E</td>
<td>ADC M</td>
<td>BE</td>
<td>CMP M</td>
<td>0C</td>
<td>INR C</td>
<td>7B</td>
<td>MOV A E</td>
</tr>
<tr>
<td>87</td>
<td>ADD A</td>
<td>D4</td>
<td>CNC 16-Bit</td>
<td>14</td>
<td>INR D</td>
<td>7C</td>
<td>MOV A H</td>
</tr>
<tr>
<td>80</td>
<td>ADD B</td>
<td>C4</td>
<td>CNZ 16-Bit</td>
<td>1C</td>
<td>INR E</td>
<td>7D</td>
<td>MOV A L</td>
</tr>
<tr>
<td>81</td>
<td>ADD C</td>
<td>F4</td>
<td>CP 16-Bit</td>
<td>24</td>
<td>INR H</td>
<td>7E</td>
<td>MOV A M</td>
</tr>
<tr>
<td>82</td>
<td>ADD D</td>
<td>EC</td>
<td>CPE 16-Bit</td>
<td>2C</td>
<td>INR L</td>
<td>47</td>
<td>MOV B A</td>
</tr>
<tr>
<td>83</td>
<td>ADD E</td>
<td>FE</td>
<td>CPI 8-Bit</td>
<td>34</td>
<td>INR M</td>
<td>40</td>
<td>MOV B B</td>
</tr>
<tr>
<td>84</td>
<td>ADD H</td>
<td>E4</td>
<td>CPO 16-Bit</td>
<td>03</td>
<td>INX B</td>
<td>41</td>
<td>MOV B C</td>
</tr>
<tr>
<td>85</td>
<td>ADD L</td>
<td>CC</td>
<td>CZ 16-Bit</td>
<td>13</td>
<td>INX D</td>
<td>42</td>
<td>MOV B D</td>
</tr>
<tr>
<td>86</td>
<td>ADD M</td>
<td>27</td>
<td>DAA</td>
<td>23</td>
<td>INX H</td>
<td>43</td>
<td>MOV B E</td>
</tr>
<tr>
<td>C6</td>
<td>ADI 8-Bit</td>
<td>09</td>
<td>DAD B</td>
<td>33</td>
<td>INX SP</td>
<td>44</td>
<td>MOV B H</td>
</tr>
<tr>
<td>A7</td>
<td>ANA A</td>
<td>19</td>
<td>DAD D</td>
<td>DA</td>
<td>JC 16-Bit</td>
<td>45</td>
<td>MOV B L</td>
</tr>
<tr>
<td>A0</td>
<td>ANA B</td>
<td>29</td>
<td>DAD H</td>
<td>FA</td>
<td>JM 16-Bit</td>
<td>46</td>
<td>MOV B M</td>
</tr>
<tr>
<td>A1</td>
<td>ANA C</td>
<td>39</td>
<td>DAD SP</td>
<td>C3</td>
<td>JMP 16-Bit</td>
<td>4F</td>
<td>MOV C A</td>
</tr>
<tr>
<td>A2</td>
<td>ANA D</td>
<td>3D</td>
<td>DCR A</td>
<td>D2</td>
<td>JNC 16-Bit</td>
<td>48</td>
<td>MOV C B</td>
</tr>
<tr>
<td>A3</td>
<td>ANA E</td>
<td>05</td>
<td>DCR B</td>
<td>C2</td>
<td>JNC 16-Bit</td>
<td>49</td>
<td>MOV C C</td>
</tr>
<tr>
<td>A4</td>
<td>ANA H</td>
<td>0D</td>
<td>DCR C</td>
<td>F2</td>
<td>JP 16-Bit</td>
<td>4A</td>
<td>MOV C D</td>
</tr>
<tr>
<td>A5</td>
<td>ANA L</td>
<td>15</td>
<td>DCR D</td>
<td>EA</td>
<td>JPE 16-Bit</td>
<td>4B</td>
<td>MOV C E</td>
</tr>
<tr>
<td>A6</td>
<td>ANA M</td>
<td>1D</td>
<td>DCR E</td>
<td>E2</td>
<td>JPO 16-Bit</td>
<td>4C</td>
<td>MOV C H</td>
</tr>
<tr>
<td>E6</td>
<td>ANA 8-Bit</td>
<td>25</td>
<td>DCR H</td>
<td>CA</td>
<td>JZ 16-Bit</td>
<td>4D</td>
<td>MOV C L</td>
</tr>
<tr>
<td>CD</td>
<td>CALL 16-Bit</td>
<td>2D</td>
<td>DCR L</td>
<td>3A</td>
<td>LDA 16-Bit</td>
<td>4E</td>
<td>MOV C M</td>
</tr>
<tr>
<td>DC</td>
<td>CC 16-Bit</td>
<td>35</td>
<td>DCR M</td>
<td>0A</td>
<td>LDAX B</td>
<td>57</td>
<td>MOV D A</td>
</tr>
<tr>
<td>FC</td>
<td>CM 16-Bit</td>
<td>0B</td>
<td>DCX B</td>
<td>1A</td>
<td>LDAX D</td>
<td>50</td>
<td>MOV D B</td>
</tr>
<tr>
<td>2F</td>
<td>CMA</td>
<td>1B</td>
<td>DCX D</td>
<td>2A</td>
<td>LHLD 16-Bit</td>
<td>51</td>
<td>MOV D C</td>
</tr>
<tr>
<td>Hex</td>
<td>mnemonic</td>
<td>Hex</td>
<td>mnemonic</td>
<td>Hex</td>
<td>mnemonic</td>
<td>Hex</td>
<td>mnemonic</td>
</tr>
<tr>
<td>-----</td>
<td>-----------</td>
<td>-----</td>
<td>-----------</td>
<td>-----</td>
<td>-----------</td>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td>52</td>
<td>MOV D D</td>
<td>71</td>
<td>MOV M C</td>
<td>E5</td>
<td>PUSH H</td>
<td>9E</td>
<td>SBB M</td>
</tr>
<tr>
<td>53</td>
<td>MOV D E</td>
<td>72</td>
<td>MOV M D</td>
<td>F5</td>
<td>PUSH PSW</td>
<td>DE</td>
<td>SBI 8-Bit</td>
</tr>
<tr>
<td>54</td>
<td>MOV D H</td>
<td>73</td>
<td>MOV M E</td>
<td>17</td>
<td>RAL</td>
<td>22</td>
<td>SHLD 16-Bit</td>
</tr>
<tr>
<td>55</td>
<td>MOV D L</td>
<td>74</td>
<td>MOV M H</td>
<td>1F</td>
<td>RAR</td>
<td>30</td>
<td>SIM</td>
</tr>
<tr>
<td>56</td>
<td>MOV D M</td>
<td>75</td>
<td>MOV M L</td>
<td>D8</td>
<td>RC</td>
<td>F9</td>
<td>SPHL</td>
</tr>
<tr>
<td>5F</td>
<td>MOV E A</td>
<td>3E</td>
<td>MVI A 8-Bit</td>
<td>C9</td>
<td>RET</td>
<td>32</td>
<td>STA 16-Bit</td>
</tr>
<tr>
<td>58</td>
<td>MOV E B</td>
<td>06</td>
<td>MVI B 8-Bit</td>
<td>20</td>
<td>RIM</td>
<td>02</td>
<td>STAX B</td>
</tr>
<tr>
<td>59</td>
<td>MOV E C</td>
<td>OE</td>
<td>MVI C 8-Bit</td>
<td>07</td>
<td>RLC</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>5A</td>
<td>MOV E D</td>
<td>16</td>
<td>MVI D 8-Bit</td>
<td>F8</td>
<td>RM</td>
<td>37</td>
<td>STC</td>
</tr>
<tr>
<td>5B</td>
<td>MOV E E</td>
<td>1E</td>
<td>MOV E 8-Bit</td>
<td>D0</td>
<td>RNC</td>
<td>97</td>
<td>SUB A</td>
</tr>
<tr>
<td>5C</td>
<td>MOV E H</td>
<td>26</td>
<td>MVI H 8-Bit</td>
<td>C0</td>
<td>RNC</td>
<td>90</td>
<td>SUB B</td>
</tr>
<tr>
<td>5D</td>
<td>MOV E L</td>
<td>2E</td>
<td>MVI L 8-Bit</td>
<td>F0</td>
<td>RP</td>
<td>91</td>
<td>SUB C</td>
</tr>
<tr>
<td>5E</td>
<td>MOV E M</td>
<td>36</td>
<td>MVI M 8-Bit</td>
<td>E8</td>
<td>RPE</td>
<td>92</td>
<td>SUB D</td>
</tr>
<tr>
<td>67</td>
<td>MOV H A</td>
<td>00</td>
<td>NOP</td>
<td>E0</td>
<td>RPO</td>
<td>93</td>
<td>SUB E</td>
</tr>
<tr>
<td>60</td>
<td>MOV H B</td>
<td>B7</td>
<td>ORA A</td>
<td>0F</td>
<td>RRC</td>
<td>94</td>
<td>SUB H</td>
</tr>
<tr>
<td>61</td>
<td>MOV H C</td>
<td>B0</td>
<td>ORA B</td>
<td>C7</td>
<td>RST 0</td>
<td>95</td>
<td>SUB L</td>
</tr>
<tr>
<td>62</td>
<td>MOV H D</td>
<td>B1</td>
<td>ORA C</td>
<td>CF</td>
<td>RST 1</td>
<td>96</td>
<td>SUB M</td>
</tr>
<tr>
<td>63</td>
<td>MOV H E</td>
<td>B2</td>
<td>ORA D</td>
<td>D7</td>
<td>RST 2</td>
<td>96</td>
<td>SUB M</td>
</tr>
<tr>
<td>64</td>
<td>MOV H H</td>
<td>B3</td>
<td>ORA E</td>
<td>DF</td>
<td>RST 3</td>
<td>95</td>
<td>SUB L</td>
</tr>
<tr>
<td>65</td>
<td>MOV H L</td>
<td>B4</td>
<td>ORA H</td>
<td>E7</td>
<td>RST 4</td>
<td>AF</td>
<td>XRA A</td>
</tr>
<tr>
<td>66</td>
<td>MOV H M</td>
<td>B5</td>
<td>ORA L</td>
<td>EF</td>
<td>RST 5</td>
<td>A8</td>
<td>XRA B</td>
</tr>
<tr>
<td>6F</td>
<td>MOV L A</td>
<td>B6</td>
<td>ORA M</td>
<td>F7</td>
<td>RST 6</td>
<td>A9</td>
<td>XRA C</td>
</tr>
<tr>
<td>68</td>
<td>MOV L B</td>
<td>F6</td>
<td>ORI 8-Bit</td>
<td>FF</td>
<td>RST 7</td>
<td>AA</td>
<td>XRA D</td>
</tr>
<tr>
<td>69</td>
<td>MOV L C</td>
<td>D3</td>
<td>OUT 8-Bit</td>
<td>C8</td>
<td>RZ</td>
<td>AB</td>
<td>XRA E</td>
</tr>
<tr>
<td>6A</td>
<td>MOV L D</td>
<td>E9</td>
<td>PCHL</td>
<td>9F</td>
<td>SBB A</td>
<td>AC</td>
<td>XRA H</td>
</tr>
<tr>
<td>6B</td>
<td>MOV L E</td>
<td>C1</td>
<td>POP B</td>
<td>98</td>
<td>SBB B</td>
<td>AD</td>
<td>XRA L</td>
</tr>
<tr>
<td>6C</td>
<td>MOV L H</td>
<td>D1</td>
<td>POP D</td>
<td>99</td>
<td>SBB C</td>
<td>AE</td>
<td>XRA M</td>
</tr>
<tr>
<td>6D</td>
<td>MOV L L</td>
<td>E1</td>
<td>POP H</td>
<td>9A</td>
<td>SBB D</td>
<td>EE</td>
<td>XRI 8-Bit</td>
</tr>
<tr>
<td>6E</td>
<td>MOV L M</td>
<td>F1</td>
<td>POP PSW</td>
<td>9B</td>
<td>SBB E</td>
<td>E3</td>
<td>XTHL</td>
</tr>
<tr>
<td>77</td>
<td>MOV M A</td>
<td>C5</td>
<td>PUSH B</td>
<td>9C</td>
<td>SBB H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>MOV M B</td>
<td>D5</td>
<td>PUSH D</td>
<td>9D</td>
<td>SBB L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. LAB EXERCISES

7.1 Aim: To subtract two 8-bit numbers.

Method: The numbers to be subtracted are stored in memory locations. First number is brought to accumulator and the second number in the memory is subtracted from it. If a carry is generated, the result stored in the accumulator is complemented and a 1 is added to it. Finally, the result and carry are stored in memory locations.

Flowchart:
7.2 **Aim:** To divide two 8-bit numbers.

**Method:** The numbers to be divided are stored in memory locations. The dividend is moved to accumulator. The divisor is subtracted from the accumulator content until a carry is generated. The number of times this subtraction is done will give the quotient and the remaining value in the accumulator will give the remainder of division.

**Flowchart:**

1. **Start**
2. Clear register C to keep quotient
3. Fetch the divisor to B and dividend to A
4. \( A = A - B \)
5. **Is carry =1?**
   - **Yes**: \( A = A + B \)
   - **No**: Increment C
6. Store quotient in C in memory
7. Store remainder in A in memory
8. **Stop**
**7.3 Aim:** To check whether the given number is a palindrome or not.

**Method:** The number to be checked is stored in a memory location. It is fetched to a register and the first and last nibbles are separated. The first nibble is rotated left and the carry flag is checked. The last nibble is rotated right and the carry flag is again checked. If the carry flags of these two operations do not yield the same value, 00 is stored in memory location indicating that the number is not a palindrome and the program comes to a halt. But if, they yield the same result the process is repeated 4 times. If it completes 4 iterations successfully i.e. the carry flags for each nibble in an iteration remain the same, 01 is stored in memory location indicating that the number is a palindrome.

**Flowchart:**

```
Start

A = Number to check
E = 4

C = A AND F0
D = A AND 0F

L

Rotate C left and store carry flag in H

Rotate D right and store carry flag in L

Is H=L?

No

Load 00 in memory

Yes

E = E - 1

Is E=0?

No

L

Yes

T
```
Load 01 in memory

C

F

T

Stop
**7.4 Aim:** To sort 10 numbers stored in consecutive memory locations in ascending order.

**Method:** Initialize cycle counter, comparison counter with corresponding values and the address pointer to the location where the data is stored. Move the data pointed by the address pointer to the accumulator. Compare it with next data. If the accumulator content is less than the next data then exchange the data. Decrement comparison counter. Repeat the process with the next data until comparison counter is 0. If the comparison counter is zero then decrement cycle counter and if it is not zero increment the address pointer and repeat the whole process until cycle counter is zero.

**Flowchart:**
Decrement Cycle counter

Is Cycle counter = 0?

Yes → Stop

No → Continue

L
**7.5 Aim:** To sort 10 numbers stored in consecutive memory locations in descending order.

**Method:** Initialize cycle counter, comparison counter with corresponding values and the address pointer to the location where the data is stored. Move the data pointed by the address pointer to the accumulator. Compare it with next data. If the accumulator content is larger than the next data then exchange the data. Decrement comparison counter. Repeat the process with the next data until comparison counter is 0. If the comparison counter is zero then decrement cycle counter and if it is not zero increment the address pointer and repeat the whole process until cycle counter is zero.

**Flowchart:**

```
Start

Initialize Cycle counter, comparison counter and Address pointer.

Bring the data pointed by the address pointer to the accumulator.

Is [A] > next data

Yes

No

Exchange data

Decrement Comparison counter

Move next data to Accumulator

Is comparison counter = 0?

Yes

No

C

L
```
Decrement Cycle counter

Is Cycle counter = 0?

Yes ➔ Stop

No ➔ C

Stop
7.6 Aim: To add two 16 bit numbers.

Method: The numbers to be added are stored in two 16 bit registers. They are added and the resultant sum and carry are stored in memory locations.

Flowchart:
Stop

Store sum and [D] in memory

C
**7.7 Aim:** To convert a BCD number to a binary number.

**Method:** The number is ANDed with F0 to obtain the first 4 bits. Then it is rotated 4 times left through carry and the value is stored in a register (say B). The last 4 bits obtained when the BCD number ANDed with 0F is stored in another register (say C). The value in B is multiplied by 10 and then it is added with the contents of C to obtain the equivalent binary number. The carry, if any is also stored in some registers.

**Flowchart:**

```
Start

A = BCD number

E = 0

B = [A] AND F0

C = [A] and 0F

Rotate [B] 4 times left through carry

A = ([B] * 10) + [C]

Is carry = 1?

Yes

Increment E

No

C
```
Start

Store [E] and [A] in memory
**7.8 Aim:** To convert a binary number to BCD number.

**Method:** The binary number is stored in a register. Count the number of 100s and store it in a register say A. Count the number of 10s in it and store it in a register say B. Subtract all 100s, 10s from the original binary number and the resulting value is stored in another register. These 3 values stored will give the equivalent BCD number.

**Flowchart:**

1. Start
2. A = Binary number
3. B = No. of 100s in [A]
4. C = No. of 10s in [A]
5. A = A – (100*B) – (10*C)
6. Store A, B, C in memory
7. Stop
7.9 **Aim**: To add ten 8 bit numbers.

**Method**: Move first data to accumulator. Initialize count register. Add the next data with data in the accumulator. If there is a carry increments carry register. Decrement the count register. If it is zero store the result. Else fetch the next data and add with value in the accumulator. Repeat until carry register is zero.

**Flowchart:**
Is Counter Zero?

- No: U
- Yes: Display or Save

Stop
7.10 **Aim**: To multiply two 8 bit numbers.

**Method**: Store one of the data in a register (say C register). Move the second data to accumulator. Move the accumulator content to another register (say B register). Set the data in the C register as a counter. Add the data in B register to the content of accumulator. Decrement the value in C register. Repeat the addition until the value in the counter register C is zero. The final value in the accumulator will be the product of the two values.

**Flowchart:**

![Flowchart Image]
Is Counter Zero?

Decrement Counter

Store Result

Stop
7.11 **Aim:** To find out the largest of ten 8 bit numbers.

**Method:** The numbers are stored in consecutive memory locations. The counter register is initialized with 0A and the address pointer points to the first number. The first number is moved to a register say B. The address pointer is incremented and counter register is decremented and the next number is fetched to accumulator. If the content of accumulator is greater than that in B, it is loaded in B. The counter register is decremented and the process is repeated until the counter register reaches to 0. The final value in B will give the largest number in the series.

**Flowchart:**
Is Counter Zero?

No

Store no. pointed by address ptr. in A

Yes

Store [B] in memory

Compare [A] with [B]

Is Carry=1 ?

No

Yes

Move [B] to A

L

Stop
7.12 **Aim**: To find out the smallest of ten 8 bit numbers.

**Method**: The numbers are stored in consecutive memory locations. The counter register is initialized with 0A and the address pointer points to the first number. The first number is moved to a register say B. The address pointer is incremented and counter register is decremented and the next number is fetched to accumulator. If the content of accumulator is less than that in B, it is loaded in B. The counter register is decremented and the process is repeated until the counter register reaches to 0. The final value in B will give the greatest number in the series.

**Flowchart:**
C

Is Counter Zero?

No

Store no. pointed by address ptr. in A

Yes

Store [B] in memory

Compare [A] with [B]

No

Is Carry=1 ?

Yes

Copy [B] to A

Stop

C

L
7.13 Aim: To scroll a set of data stored in memory

Method: The numbers to be scrolled are stored in some memory location (here the scroll data starts at 80F0). When the program is executed, it will put the contents from this location on display and scroll the same. The scroll rate is alterable by altering the delay counter stored in another memory location (here 9F37).

Flow chart:
If non zero?  
Yes → C  
No → Call delay routine  
   ↓  
   Pop B and increment BC pair  
   ↓  
Decrement number of scroll steps  
   ↓  
If non zero?  
   No ← 2  
       ↓  
       Yes → 1