MICROPROCESSORS AND INTERFACING LAB

B.E. V-Semester

LAB MANUAL

[MASTER MANUAL]

Prepared by
Dr.S.K.ChayaDevi

DEPARTMENT OF INFORMATION TECHNOLOGY

VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)

INDEX

INSTITUTE VISION AND MISSION

Vision

Striving for a symbiosis of technological excellence and human values.

Mission

To arm young brains with competitive technology and nurture holistic development of the individuals for a better tomorrow.

Our Quality Policy

Education without quality is like a flower without fragrance. It is our earnest resolve to strive towards imparting high standards of teaching, training and developing human resources.

DEPARTMENT VISION AND MISSION

Vision

To be a centre of excellence in core Information Technology and multidisciplinary learning and research, where students get trained in latest technologies for professional and societal growth.

Mission

To enable the students acquire skills related to latest technologies in IT through practice- oriented teaching and training.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's)

The educational objectives of UG program in Information Technology are:

PEO1: With theoretical and practical knowledge to obtain employment or pursue higher studies and solve problems in Information Technology.

PEO2: With effective written and oral communication skills that will help them to work in diversified and dynamic working environments.

PEO3: With competence to succeed in their professional lives with ethical values.

PROGRAM SPECIFIC OUTCOMES (PSO's)

The students of B.E in Information Technology will demonstrate:

PSO1: Competency in programming using different programming languages to implement algorithms.

PSO2: Competency in the analysis and design of a software solution using different modelling tools.

PSO3: Competency in Electronic Design and Embedded System Design using different simulation tools.

PROGRAM OUTCOMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)
DEPARTMENT OF INFORMATION TECHNOLOGY

Microprocessors and Interfacing Lab
(Syllabus for B.E V- SEMESTER)

S.No	Course Outcomes	BloomsTaxonomy Level
1	Do basic assembly language programming using 8085 microprocessors	3
2	Do basic assembly language programming using 8086 microprocessors	3
3	Interface various peripherals to 8086 microprocessor	4

MAPPING LEVELS:

Correlation levels 1, 2 or 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

CO/PO/PSO MAPPING:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	0	1	0	1	0	0	0	0	1	0	0
CO2	3	0	1	0	1	0	0	0	0	1	0	0
CO3	3	0	1	0	1	0	0	0	0	1	0	0

	PSO1	PSO2	PSO3
CO1	0	0	3
CO2	0	0	3
CO3	0	0	3

CO/PO/PSO JUSTIFICATION

MAPPING	CORRELATION LEVELS	JUSTIFICATION
CO1-PO1	**3**	By understanding different processors architectures and its instruction set, students can apply their knowledge to write solutions for engineering problems
CO1-PO3	**1**	Students able design solutions for complex engineering problems and design system components or processes under considerations.
CO1-PO5	**1**	Create, select, and apply appropriate ITtools for complex engineering activities.
CO1-PO10	**1**	Communicate effectively on complex engineering activitieswith the engineering communityand design documentation, make effective presentations
CO2-PO1	**3**	By understanding different processors architectures and its instruction set, students can apply their knowledge to write solutions for engineering problems
CO2-PO3	**1**	Students able design solutions for complex engineering problems and design system components or processes under considerations
CO2-PO5	**1**	Create, select, and apply appropriate IT tools for complex engineering activities.
CO2-PO10	**1**	Communicate effectively on complex engineering activitieswith the engineering community and design documentation, make effective presentations
CO3-PO1	**3**	By understanding different processors architectures and its instruction set, students can apply their knowledge to write solutions for engineering problems
CO3-PO3	**1**	Students able design solutions for complex engineering problems and design system components or processes under considerations.
CO3-PO5	**1**	Create, select, and apply appropriate IT tools for complex engineering activities.
CO3-PO10	**1**	Communicate effectively on complex engineering activitieswith the engineering community and design documentation, make effective presentations

CO-PSO mapping:

CO1-PSO3	**3**	Competency in Electronic Design and Embedded System Design using different simulation tools.
CO2-PSO3	**3**	Competency in Electronic Design and Embedded System Design using different simulation tools.
CO3-PSO3	**3**	Competency in Electronic Design and Embedded System Design using different simulation tools.

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031
DEPARTMENT OF INFORMATION TECHNOLOGY

MICROPROCESSORS AND INTERFACING LAB
SYLLABUS FOR B.E. V SEMESTER

L:T:P(Hrs./week): 0:0:2	SEE Marks :50	Course Code: **PC521IT**
Credits : 1	CIE Marks :30	Duration of SEE : 3 Hours

COURSE OBJECTIVES	**COURSE OUTCOMES** *On completion of the course, students will be able to*
The course will enable the students to write assembly language programs using 8085 and 8086 microprocessors.	1. Do basic assembly language programming using 8085 microprocessor 2. Do basic assembly language programming using 8086 microprocessor. 3. Interface various peripherals to 8086 microprocessor.

1. Assembly Language programming with 8085, 8086 .
2. Interfacing and programming of 8255.
3. Interfacing and programming of 8253/8254.
4. Interfacing and programming of 8279.
5. A/D and D/A converter interface.
6. Stepper motor interface.
7. Display interface

Note: Adequate number of programs covering all the instructions of 8085 & 8086 instruction set. Experiments should be done on the 8085, 8086 microprocessor trainer kits and Assembler

S. No.	Name of the Experiment	CO mapping	PO/PSO mapping
1.Assembly Language programming with 8085, 8086		CO1, CO2	PO1,3,5,10, PSO3
1	Addition and Subtraction of 8 bit data	CO1, CO2	PO1,3,5,10, PSO3
2	Transferring block of data	CO1, CO2	PO1,3,5,10, PSO3
3	Adding an array of bytes	CO1, CO2	PO1,3,5,10, PSO3
4	Count of even and odd numbers in the series	CO1, CO2	PO1,3,5,10, PSO3
5	Count of positive and negative in the series	CO1, CO2	PO1,3,5,10, PSO3
6	Largest number from the given series of bytes	CO1, CO2	PO1,3,5,10, PSO3
7	Smallest number from the given series of bytes	CO1, CO2	PO1,3,5,10, PSO3
8	Length of the given series	CO1, CO2	PO1,3,5,10, PSO3
9	Hexadecimal to BCD conversion	CO1, CO2	PO1,3,5,10, PSO3
10	BCD to Hexadecimal conversion	CO1, CO2	PO1,3,5,10, PSO3
11	Hexadecimal to ASCII conversion	CO1, CO2	PO1,3,5,10, PSO3
12	ASCII to Hexadecimal conversion	CO1, CO2	PO1,3,5,10, PSO3
13	Adding two 16 bit numbers	CO1, CO2	PO1,3,5,10, PSO3
14	Subtracting two 16 bit numbers	CO1, CO2	PO1,3,5,10, PSO3
15	2's complement of the given number	CO1, CO2	PO1,3,5,10, PSO3
16	Comparing 2 strings	CO1, CO2	PO1,3,5,10, PSO3
17	Search a substring from main string	CO1, CO2	PO1,3,5,10, PSO3
18	Adding 5 BCD numbers	CO1, CO2	PO1,3,5,10, PSO3
19	Sorting given array of bytes in ascending order	CO1, CO2	PO1,3,5,10, PSO3
20	Sorting given array of bytes in descending order	CO1, CO2	PO1,3,5,10, PSO3
Programs using MASM			
21	Introduction to MASM	CO1, CO2	PO1,3,5,10, PSO3
22	Multi byte addition	CO1, CO2	PO1,3,5,10, PSO3
23	Multi byte subtraction	CO1, CO2	PO1,3,5,10, PSO3
24	Multi byte multiplication	CO1, CO2	PO1,3,5,10, PSO3
25	Multi byte division	CO1, CO2	PO1,3,5,10, PSO3
26	Searching substring from main string	CO1, CO2	PO1,3,5,10, PSO3
27	Insert substring in to main string	CO1, CO2	PO1,3,5,10, PSO3
28	Delete a substring from main string	CO1, CO2	PO1,3,5,10, PSO3
29	Sorting array of numbers	CO1, CO2	PO1,3,5,10, PSO3

S. No.	Name of the Experiment	CO mapping	PO/PSO mapping
2. Interfacing and programming of 8255 with 8085 and 8086		CO3	PO1,3,5,10, PSO3
3. Interfacing and programming of 8253/8254		CO3	PO1,3,5,10, PSO3
4. Interfacing and programming of 8279		CO3	PO1,3,5,10, PSO3
5. A/D and D/A converter interface		CO3	PO1,3,5,10, PSO3
5.1	Interface DAC to 8085 and 8086 processor	CO3	PO1,3,5,10, PSO3
5.2	Interface ADC to 8086 processor	CO3	PO1,3,5,10, PSO3
6. Interface stepper motor to 8085 and 8086 processor		CO3	PO1,3,5,10, PSO3
7. Seven segment display interfacing with 8086 and 8085 processor		CO3	PO1,3,5,10, PSO3

Note: Adequate number of programs covering all the instructions of 8085 & 8086 instruction set should be done on the 8085 and 8086 microprocessor trainer kits.

HARDWARE REQUIREMENT

1. 8085 microprocessor kit
2. 8086 microprocessor kit,
3. 8255 programmable peripheral interface
4. stepper motor , stepper motor interface
5. Digital to Analog Converter interface
6. Analog to Digital converter interface
7. seven segment display interface
8. 8254 programmable timer interface
9. 8279 key board and display interface
10. traffic light interface
11. Function generator
12. Cathode Ray Oscilloscope
13. Power supply units
14. Systems with i3 processor

SOFTWARE REQUIREMENT

MASM Software

EX.NO:1 Assembly language programming with8085, 8086

Introduction to 8085 processor Kit:

Objectives:

i. To explain the programming model of 8085A
ii. To define the various addressing modes in 8085A instruction set
iii. To understand the hardware specifications of 8085A
iv. Hex codes

Introduction:

INTEL 8085A is a general purpose 8-bit microprocessor capable of addressing 64kB of memory. It is an enhanced version of its predecessor. The 8085A and its instruction set is upward compatible with that of 8080A.

The microprocessor has a set of instructions designed internally to manipulate data and communicate with peripherals. It can be programmed to perform functions on given data by selecting necessary instructions from its set.

The internal architecture of the 8085A determines how and what operations can be performed with the data. To perform any operation, the microprocessor requires registers, an arithmetic logic unit (ALU) and control logic and internal busses. Certain registers of the processor can be written into and hence are available for manipulation or processing of input data. Use of certain write instructions makes this feasible.

Accumulator:

The accumulator is the primary source and destination for one operand and two operand instructions. For example, all data transfers between the CPU and I/O devices are performed through the accumulator. In addition, many memory reference instructions move data between the accumulator and the memory than between any other registers and memory.

All arithmetic and Boolean instructions take one of the operands from the accumulator and return the result to the accumulator. So, the accumulator should be loaded before any arithmetic or Boolean operation.

Registers:

Apart from the accumulator, we got other registers such as B, C, D, E, H and L. These are called as secondary registers. Data stored in any of these 6 registers may be accessed with equal ease. Such data can be moved to any other registers or can be used as the second operand in the 2 operand instructions. These 6 registers can be used to hold 8-bit data when used individually or 16-bit data when used in pairs as BC or DE or HL. Registers H & L comprise the primary data pointer for 8085A. Normally, these 2 registers will be used to hold 16-bit memory address or data being accessed. It is usually called memory pointer. We can transfer data between any specified register and memory location addressed by H & L. It is good to address data memory via registers H & L, whenever possible because it can make your program move efficient and easy to relocate.

Flags:

The ALU consists of 5 flip flops that are set or reset according to data conditions in the accumulator and other registers. The microprocessor uses them to test for data conditions. The 5 flip flops referred to as flags, are the carry (C) flag, zero (Z) flag, sign (S) flag, parity (P) flag and auxiliary carry (AC) flag.

Program counter (PC):

This register deals with sequencing the execution of instructions. The function of PC is to point to the memory location from which the next byte is to be fetched for execution.

Stack pointer (SP);

The stack pointer register is a 16-bit and is used as a memory pointer. It points to a memory location in the RAM called stack. The beginning of the stack is defined by the user.

8085A instruction classification:

The function of a microprocessor system is implemented by a sequence of data transfers between memory, processor and I/O devices and data transformations that occur in the registers within the microprocessor, manipulating a register under program control, addressing it and using it for data transfer and transformation requires a set of binary codes which comprise the INSTRUCTION SET OF MICROPROCESSOR.

The 8085 instruction can be classified into the following 5 functional categories

i. Data transfer (copy) operations
ii. Arithmetic operations
iii. Logical operations

iv. Branching operations

v. Machine-control operations

Since data and instructions may reside anywhere in the internal registers, external registers or memory, locating them requires a particular addressing. The instructions in these 5 functional groups can be categorized according to their method of addressing the hardware registers and memory. This method is called ADDRESSING MODE and 6 modes are available with 8085A which are

i. Implied addressing
ii. Register addressing
iii. Immediate addressing
iv. Direct addressing
v. Register indirect addressing
vi. Combined addressing

i. **Implied addressing:**The instructions using this mode have no explicit operands.

E.g.: STC (set carry flag)

DAA (decimal adjust accumulator)

ii. **Register addressing:** This mode specifies the register or register pair that contains data. Both the source and the destination operands are registers.

E.g.: MOV B, C

-moves the content of the register 'C' to register 'B'.

iii. **Immediate addressing:** For an 8-bit data, this mode uses 2 bytes, with the first byte as the OP code, followed by 1 byte of data. On the other hand, for a 16-bit data, this instruction contains 3 bytes, with the first byte as the OP code followed by 2 bytes of data.

E.g.: MVI B, 05

-loads register 'B' with the value 5

LXI H, 2050

-loads 'H' with 20 and 'L' with 50

iv. **Direct addressing:** instructions using this mode specify the effective address as a part of instruction. These instructions contain 3 bytes, with the first byte as the OP code followed by 2 bytes of address of data (the lower order byte of address in byte 2 and the higher order byte of address in byte 3).

E.g.: LDA 2035

-loads accumulator with the contents of memory location 2035

This mode of addressing is also called absolute addressing.

v. **Register indirect addressing:** this mode contains a register pair which stores the address of the data (the higher order byte of the address in the first register of the pair and the lower order byte in the second register of the pair).

E.g.: LDAX B

-loads the accumulator with the contents of the memory location addressed by 'B' and 'C' register pair

vi. **Combined addressing**: some instructions use a combination of addressing modes. 'CALL' instruction, for example, combines direct and register indirect addressing. The direct address in a CALL instruction specifies the address of the desired subroutine; the register indirect address is that of stack pointer. The CALL instruction pushes the current contents of the PC into the memory location specified by stack pointer. The address that follows CALL instruction is copied to PC and hence execution starts at the address of the subroutine.

Specifications:

1. Hardware specifications:

i. P**rocessor clock frequency:**

INTEL 8085A at 6.144MHz clock

ii. **Memory:**

Monitor EPROM: 0000-1FFF

EPROM~ Expansion: 2000-3FFF and C000-FFFF

System RAM: 4000-5FFF

Monitor data area: 4000-40FF (reserved)

User RAM area: 4100-5FFF

RAM~ Expansion: 6000-BFFF

Note: The RAM area from 4000-40FF should not be accessed by the user, since it is used as scratch pad by the monitor program.

iii. **Input/ Output**:

Parallel: 24/22 I/O lines using 8255 and 8155 respectively.

iv. **Display:** 6 digit, 0.3'', 7 segment red LED display with filter. 4 digits for address display and 2 for data display.

v. **Keyboard:** 20 keys soft keyboard including command keys and hexadecimal keys, also provides with micro switch for interrupt functions.

vi. **Battery backup:** Onboard battery backup facility is provided for the available RAM

vii. **System power consumption:**

+5V@Amp

+12V@200mA

-12V@100mA

viii. **Power supply specifications:**

Input: 230V AC @50Hz

Output: +5V@3A +5V@600mA

+12V@250mA

-12V@250mA

+30V@250mA

-unregulated

ix. **Physical characteristics:**

Micro -85 LC PCB: 6.8"x6.8"

Weight: ½ kg

x. **Bus expansion:** A VXT bus has been incorporated in Micro-85L which facilitates to patch-up any extra hardware to micro-85LC. All address, data and control signals are brought out to this bus. Using VXT-bus, all VBMB boards can be interfaced with micro-85LC.

xi. **Test points:** Test points provided for MR*, MW*, INTA*, IO/M*, IOR*, IOW*, SO, SI, INTA. This enables the user to study the hardware timing easily.

2. Software specifications:

Micro-85LC contains a high performance 8kB monitor program. It is designed to respond to user inputs, RS232C serial communications, tape interface, etc. Out of keys in the keyboard, 16 are hexadecimal commands and register keys and remaining keys are stand-alone keys.

RES -This RES key allows you to terminate any present activity and to return your micro-85LC to an initialized state. When pressed, the micro-85LC sign-on message appears in the display for a few seconds and the monitor will display command prompt"--" in the left most digit.

DEC -Decrement the address by 1 and displays its contents

Or

Display the previous register content

EXE -Executes a particular program after selecting the address through GO command.

NEXT -Increment address by 1 and display its content

Or

Display the next register

The 16 hexadecimal keys have either a dual role or a triple role to play.

i. It functions as hex key entry when an address or a data entry is required.

ii. It functions as the register key entry during register commands.

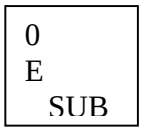
iii. It functions as command key when pressed directly after command prompt.

Note: the hex key function summary is in the below mentioned order.

i. Hex key

ii. Command key

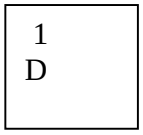
iii. Register key



i. Hex key entry "0"

ii. This key is for substituting memory contents. When NEXT key is pressed immediately after this, it takes the user to the start address for entering user programs, 4100 hex (user RAM).

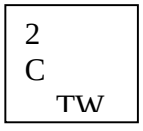
iii. Register key "E".



i. Hex key entry "1"

ii. Examine the 8085A registers and modify the same.

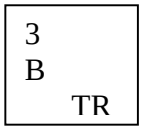
iii. Register key "D".



i. Hex key entry "2"

ii. Writes data from memory onto audio tape.

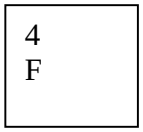
iii. Register key "C"



i. Hex key entry "3"

ii. Retrieve data from audio to memory.

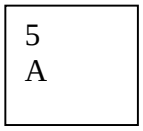
iii. Register key "B".



i. Hex key entry "4"

ii. Block search for a byte.

iii. Register key "F".



i. Hex key entry "5"

ii. Fill a block of RAM memory with desired data.

iii. Register key "A".

6 L	i.	Hex key entry "6"
	ii.	Transfer/register data to/from the serial port. The TW/TR keys are used for sending/receiving data respectively.
	iii.	Register key "L".
7 H F2	i.	Hex key entry "7"
	ii.	Register key "H".
8 I GO	i.	Hex key entry "8"
	ii.	Start running a particular program.
	iii.	Register key "I".
9 PCL SNG	i.	Hex key entry "9"
	ii.	Single step of a program instruction by instruction.
	iii.	Register key "PCL".
A PCH F3	i.	Hex key entry "A"
	ii.	Function key F3 F3 [0] = Input a byte from a port. F3 [1] = Output a byte to a port.
	iii.	Register key "PCH".
	iv.	Used with SNG key for hardware single stepping.
B SPL BC	i.	Hex key entry "B"
	ii.	Check a particular block for blank.
	iii.	Register key "SPL".
C SPH	i.	Hex key entry "C"
	ii.	Move block of memory to another block.
	iii.	Register key "SPH".
D	i.	Hex key entry "D"
	ii.	Compares 2 memory blocks.
E INS	i.	Hex key entry "E"
	ii.	Insert bytes into memory (RAM).
F	i.	Hex key entry "F"
	ii.	Delete bytes from memory (RAM).

Hardware details:

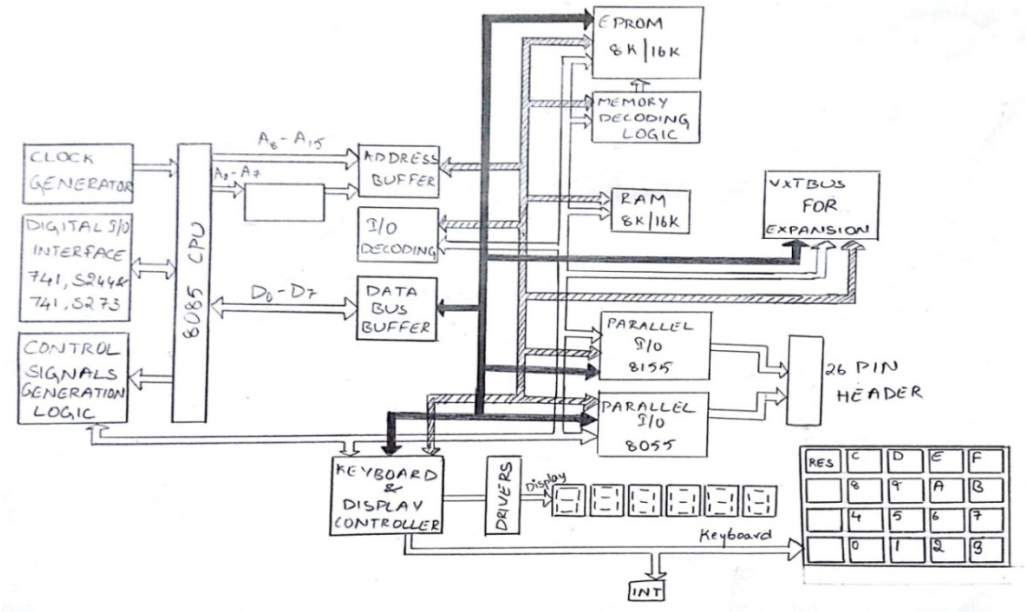
Component layout and functional block diagram:

The functional block diagram shown in the figure provides the complete system design in blocks. It shows the component layout that gives the exact layout and location of the connectors, IC's and other components.

Description of functional block diagram:

Referring to the figure, the following explanation outlines the working of the 8085A processor trainer kit as a system.

Micro-85LC basic functional block diagram:



i. **Inputs to and outputs from the CPU:** The CPU gets the clock from the clock generator which is a crystal at 6.144MHz. The reset*, interrupt lines and data lines are also inputs to the CPU. The CPU outputs comprise the clock reset, address lines, data lines and control lines.

ii. **Address and data bus:** the lower order address lines are latched using ALE and thus de-multiplexed from the data lines. The higher order address lines are taken directly from 8085. These 2 sets of lines make the 16-bit address bus. The 8 data lines are taken directly from the 8085.

iii. **Control bus:** the control signals required for proper operation of the system are IOR* (I/O Read), IOW*(I/O Write), MR*(Memory Read) and MW*(Memory Write). The peripherals on the trainer are all I/O mapped and hence to input from or output to a peripheral IOR* and IOW* are utilized. The memory read and memory write signals are used to enable an EPROM and RAM and write into RAM respectively. These signals are generated from the IO/M* and WR*, RD* signals.

iv. **Chip select logic**: the selection of any peripherals or memory requires a CS* to enable that particular device. This requires address decoding, both memory and I/O. All the above signals (address, data and control) and chip select are routed to all the peripherals and memory devices in the trainer.

v. **Keyboard and display:** the block diagram shows a 21 keys keyboard and 8-digit display. This is the unit with which the user communicated with the system. The keypad and display interfaced with the CPU with the help of an 8279, I/O mapped with the CPU. The display is driven by display drivers driven by the KDC (8279). The keyboard lines are encoded and sent to the KDC.

vi. **Memory**: the block diagram shows an 8kB EPROM or 32kB EPROM and an 8kB RAM or 32kB RAM.

Memory configuration:

This explains the memory mapping facilities available on the micro-85LC. The figure explains the memory configuration of micro-85LC which gives the clear idea of how memory allocation can be done.

EPROMexpansion (FFFF) (C000)
User expansion RAM (BFFF) (6000)
User RAM area (5FFF) (4100)
Monitor program data area (40FF) (4000)
EPROM expansion (3FFF) (2000)
Monitor EPROM area (1FFF) (0000)

Allocation of EPROM:

The micro-85LC has a standard EPROM configuration of 8kB using one 2764(8kB x 8kB EPROM). The address for the basic EPROM is 0000-1FFF.

Start address	End address	Socket number	IC used	Total capacity
0000	1FFF	U9	2764x1	8kB

Allocation of RAM:

The standard micro-85LC comes with 8kB of RAM using one 6264(8k x 8 RAM) whose address is from 4000-5FFF, out of which the first FF locations are used by the system for its data buffers. Hence user RAM area starts from 4100.

Start address	End address	Socket number	IC used	Total capacity
4000	5FFF	U8	6264x1	8kB

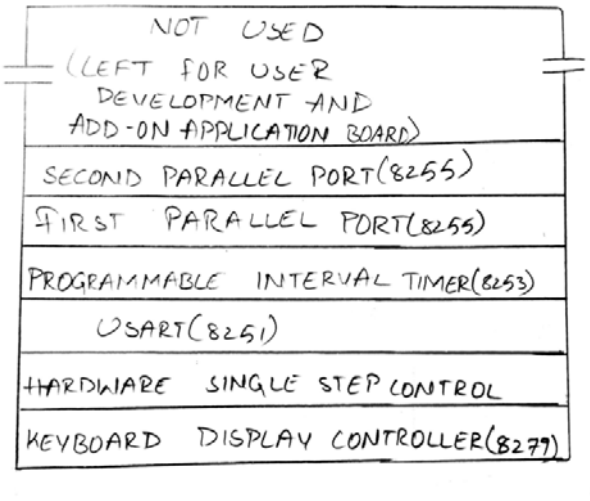
Memory expansion:

Apart from the basic EPROM and RAM at addresses 0000-1FFF and 4000-5FFF respectively, the addresses 2000-3FFF and C000-FFFF is meant for EPROM expansion. The address for the expansion is shown in the figure.

Start address	End address	Socket number	IC used	Total capacity
2000	3FFF			
C000	FFFF	U9	27256	32kB
6000	BFFF	U8	62256	32kB

Battery backup:

Battery backup facility is provided optionally for 32kB RAM for address 4000-5FFF (8kB) in mode 0 or from 4000-BFFF (32kB) in mode 1 at socket U15. A 3.6V NiCad battery is used which provides power to RAM during power-off and gets charged during power-on. The data in RAM can be retained to a maximum of 12 hours if the battery has been charged to its maximum extent.



Allocation of I/O addresses:

The peripherals available on micro-85LC are all I/O mapped. The complete I/O allocation table is shown in the figure. As seen from the table, the on-board peripherals occupy I/O addresses from 00-3F. The add-on application boards occupy I/O addresses from 80-FF. Apart from these reserves I/O addresses, the rest are available to the users for their own development purpose.

Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic
00	NOP	2C	INR L	55	MOV D,L	7D	MOV A,L	A5	ANA L	CE	ACI
01	LXI B	2D	DCR L	56	MOV D,M	7E	MOV A,M	A6	ANA M	CF	RST 1
02	STAX B	2E	MVI L	57	MOV D,A	7F	MOA A,A	A7	ANA A	D0	RNC
03	INX B	2F	CMA	58	MOV E,B	80	ADD B	A8	XRA B	D1	POP D
04	INR B	30	SIM	59	MOV E,C	81	ADD C	A9	XRA C	D2	JNC
05	DCR B	31	LXI SP	5A	MOV E,D	82	ADD D	AA	XRA D	D3	OUT
06	MVI B	32	STA	5B	MOV E,E	83	ADD E	AB	XRA E	D4	CNC
07	RLC	33	INX SP	5C	MOV E,H	84	ADD H	AC	XRA H	D5	PUSH D
09	DAD B	34	INR M	5D	MOV E,L	85	ADD L	AD	XRA L	D6	SUI
0A	LDAX B	35	DCR M	5E	MOV E,M	86	ADD M	AE	XRA M	D7	RST 2
0B	DCX B	36	MVI M	5F	MOV E,A	87	ADD A	AF	XRA A	D8	RC
0C	INR C	37	STC	60	MOV H,B	88	ADC B	B0	ORA B	DA	JC
0D	DCR C	39	DAD SP	61	MOV H,C	89	ADC C	B1	ORA C	DB	IN
0E	MVI C	3A	LDA	62	MOV H,D	8A	ADC D	B2	ORA D	DC	CC
0F	RRC	3B	DCX SP	63	MOV H,E	8B	ADC E	B3	ORA E	DE	SBI
11	LXI D	3C	INR A	64	MOV H,H	8C	ADC H	B4	ORA H	DF	RST 3

Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic
12	STAX D	3D	DCR A	65	MOV H,L	8D	ADC L	B5	ORA L	E0	RPO
13	INX D	3E	MVI A	66	MOV H,M	8E	ADC M	B6	ORA M	E1	POP H
14	INR D	3F	CMC	67	MOV H,A	8F	ADC A	B7	ORA A	E2	JPO
15	DCR D	40	MOV B,B	68	MOV L,B	90	SUB B	B8	CMP B	E3	XTHL
16	MVI D	41	MOV B,C	69	MOV L,C	91	SUB C	B9	CMP C	E4	CPO
17	RAL	42	MOV B,D	6A	MOV L,D	92	SUB D	BA	CMP D	E5	PUSH H
19	DAD D	43	MOV B,E	6B	MOV L,E	93	SUB E	BB	CMP E	E6	ANI
1A	LDAX D	44	MOV B,H	6C	MOV L,H	94	SUB H	BC	CMP H	E7	RST 4
1B	DCX D	45	MOV B,L	6D	MOV L,L	95	SUB L	BD	CMP L	E8	RPE
1C	INR E	46	MOV B,M	6E	MOV L,M	96	SUB M	BE	CMP M	E9	PCHL
1D	DCR E	47	MOV B,A	6F	MOV L,A	97	SUB A	BF	CMP A	EA	JPE
1E	MVI E	48	MOV C,B	70	MOV M,B	98	SBB B	C0	RNZ	EB	XCHG
1F	RAR	49	MOV C,C	71	MOV M,C	99	SBB C	C1	POP B	EC	CPE
20	RIM	4A	MOV C,D	72	MOV M,D	9A	SBB D	C2	JNZ	EE	XRI
21	LXI H	4B	MOV C,E	73	MOV M,E	9B	SBB E	C3	JMP	EF	RST 5
22	SHLD	4C	MOV C,H	74	MOV M,H	9C	SBB H	C4	CNZ	F0	RP
23	INX H	4D	MOV C,L	75	MOV M,L	9D	SBB L	C5	PUSH B	F1	POP PSW
24	INR H	4E	MOV C,M	76	HLT	9E	SBB M	C6	ADI	F2	JP
25	DCR H	4F	MOV C,A	77	MOV M,A	9F	SBB A	C7	RST 0	F3	DI
26	MVI H	50	MOV D,B	78	MOV A,B	A0	ANA B	C8	RZ	F4	CP
27	DAA	51	MOV D,C	79	MOV A,C	A1	ANA C	C9	RET	F5	PUSH PSW
29	DAD H	52	MOV D,D	7A	MOV A,D	A2	ANA D	CA	JZ	F6	ORI
2A	LHLD	53	MOV D,E	7B	MOV A,E	A3	ANA E	CC	CZ	F7	RST 6
2B	DCX H	54	MOV D,H	7C	MOV A,H	A4	ANA H	CD	CALL	F8	RM

Hex	Mnemonic
F9	SPHL
FA	JM
FB	EI
FC	CM
FE	CPI
FF	RST 7

EX.NO:1.1ADDITION AND SUBTRACTION OF 8-BIT DATA

1.1.1Aim:

Write an assembly language code for adding2 signed or unsigned numbers using immediateaddressing mode.

Algorithm:

Step-1: Move operand 1 to accumulator.

Step-2: Add operand 2 with accumulator data and store the result in accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	3E		MVI A,OP1	Moves operand 1 data to accumulator
4101	OP1			
4102	C6		ADI OP2	Adds operand 2 with accumulator content and store result in accumulator
4103	OP2			
4104	32		STA 4200	Stores accumulator content in the memory location 4200
4105	00			
4106	42			
4107	CF		RST1	Stops the program

Result:

Unsigned:

Input1: 23, 33

Output1: 56

Input2: 5D (93), 0D (13)

Output2: 6A (106)

Signed:

Input1: FC (-4), 02 (+2)

Output1: FE (-2)

Input2: FC (-4), FE (-2)

Output2: FA (-6)

Program to add 2 signed or unsigned numbers using immediate addressing mode executed successfully.

1.1.2Aim:

Write an assembly language code for adding 2 signed or unsigned numbers using registeraddressing mode.

Algorithm:

Step-1: Move operand 1 to accumulator.

Step-2: Move operand 2 to register B

Step-3: add contents of register C with accumulator content and store the result in the accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	3E		MVI A,OP1	Moves operand 1 data to accumulator
4101	OP1			
4102	06		MVI B, OP2	Move operand 2 to register B
4103	OP2			
4104	80		ADD B	Add B register content with accumulator content and store the result in accumulator.
4105	32		STA 4200	Stores accumulator content in the memory location 4200
4106	00			
4107	42			
4108	CF		RST1	Stops the program

Result:

Unsigned:

Input1: 23, 33

Output1: 56

Input2: 5D (93), 0D (13)

Output2: 6A (106)

Signed:

Input1: FC (-4), FE (+2)

Output1: FE (-2)

Input2: FC (-4), FE (-2)

Output2: FA (-6)

Program to add 2 signed or unsigned numbers using register addressing mode executed successfully.

1.1.3Aim:

Write an assembly language code for adding2 signed or unsigned numbers using memory addressing mode.

Algorithm:

Step-1: Store 2 operands in 2 consecutive memory locations.

Step-2: Make HL pair point to a memory location that contains operand 1.

Step-3: Move operand 1 to accumulator from the memory location.

Step-4: Increment HL pair so that it points to memory location that contains operand 2.

Step-5: Add contents of memory pointed by HL pair with accumulator content and store the result in the accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200 memory location.
4101	00			
4102	42			
4103	7E		MOV A, M	Move data in memory pointed by HL pair to A
4104	23		INX H	Increment HL pair
4105	86		ADD M	Add memory content pointed by HL pair with accumulator content and store the result in accumulator.
4106	32		STA 4202	Stores accumulator content in the memory location 4202
4107	02			
4108	42			
4109	CF		RST1	Stops the program
4200	OP1			Store operand 1 in 4200 memory location
4201	OP2			Store operand 2 in 4201 memory location

Result:

Unsigned:

Input1: 23, 33

Output1: 56

Input2: 5D (93), 0D (13)

Output2: 6A (106)

Signed:

Input1: FC (-4), FE (+2)

Output1: FE (-2)

Input2: FC (-4), FE (-2)

Output2: FA (-6)

Program to add 2 signed or unsigned numbers using memory addressing mode executed successfully

1.1.4Aim:

To write an assembly language code to subtract 2 signed or unsigned numbers using immediate addressing mode.

Algorithm:

Step-1: Move operand 1 to accumulator.

Step-2: Subtract operand 2 from accumulator data and store the result in accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	3E		MVI A,OP1	Moves operand 1 data to accumulator
4101	OP1			
4102	D6		SUI OP2	Subtracts operand 2from accumulator content and store result in accumulator
4103	OP2			
4104	32		STA 4200	Stores accumulator content in the memory location 4200
4105	00			
4106	42			
4107	CF		RST1	Stops the program

Result:

Unsigned:

Input1: 04, 02

Output1: 02

Input2: 23, 33

Output2: F0 (-10)

Signed:

Input1: FC (-4), FE (-2)

Output1: FE (-2)

Input2: FC (-4), 02

Output2: FA (-6)

Program to subtract 2 signed or unsigned numbers using immediate addressing mode executed successfully.

1.1.5Aim:

Write an assembly language code for subtracting 2 signed or unsigned numbers using register addressing mode.

Algorithm:

Step-1: Move operand 1 to accumulator.

Step-2: Move operand 2 to register B

Step-3: Subtract contents of register C from accumulator content and store the result in the accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	3E		MVI A,OP1	Moves operand 1 data to accumulator
4101	OP1			
4102	06		MVI B, OP2	Move operand 2 to register B
4103	OP2			
4104	90		SUB B	Subtract B register content from accumulator content and store the result in accumulator.
4105	32		STA 4200	Stores accumulator content in the memory location 4200
4106	00			
4107	42			
4108	CF		RST1	Stops the program

Result:

Unsigned:

Input1: 04, 02

Output1: 02

Input2: 23, 33

Output2: F0 (-10)

Signed:

Input1: FC (-4), FE (-2)

Output1: FE (-2)

Input2: FC (-4), 02

Output2: FA (-6)

Program to subtract 2 signed or unsigned numbers using register addressing mode executed successfully.

1.1.6Aim:

Write an assembly language code for subtracting 2 signed or unsigned numbers using memory addressing mode.

Algorithm:

Step-1: Store 2 operands in 2 consecutive memory locations.

Step-2: Make HL pair point to a memory location that contains operand 1.

Step-3: Move operand 1 to accumulator from the memory location.

Step-4: Increment HL pair so that it points to memory location that contains operand 2.

Step-5: Subtract contents of memory pointed by HL pair from accumulator content and store the result in the accumulator.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200 memory location.
4101	00			
4102	42			
4103	7E		MOV A, M	Move data in memory pointed by HL pair to A
4104	23		INX H	Increment HL pair
4105	96		SUB M	Subtract memory content pointed by HL pair from accumulator content and store the result in accumulator.
4106	32		STA 4202	Stores accumulator content in the memory location 4202
4107	02			
4108	42			
4109	CF		RST1	Stops the program
4200	OP1			Store operand 1 in 4200 memory location
4201	OP2			Store operand 2 in 4201 memory location

Result:

Unsigned:

Input1: 04, 02

Output1: 02

Input2: 23, 33

Output2: F0 (-10)

Signed:

Input1: FC (-4), FE (-2)

Output1: FE (-2)

Input2: FC (-4), 02

Output2: FA (-6)

Program to subtract 2 signed or unsigned numbers using memory addressing mode executed successfully

EX.NO:1.2 TRANSFERING BLOCK OF DATA

Aim:

To write an assembly language code that transfers block of data from one memory location to another.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Make DE pair point to destination location.

Step-3: Load no. of data bytes that are to be transferred into register B.

Step-4: Move data in memory pointed by HL pair to accumulator and increment HL pair so that it points to next memory location.

Step-5: Store accumulator content in memory pointed by DE pair and increment DE pair so that it points to next memory location.

Step-6: Decrement B content by 1.

Step- 7: If B content is not zero, then go to step-4.Else, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load 4200 onto HL pair, to make HL pair point to 4200 memory location
4101	00			
4102	42			
4103	11		LXI D, 4300	Load 4300 onto DE pair, to make DE pair point to 4300 memory location
4104	00			
4015	43			
4106	06		MVI B, 06	Move 06 to B
4107	06			
4108	7E	LOOP	MOV A, M	Move accumulator content to memory pointed by HL pair
4109	23		INX H	Increment HL pair
410A	12		STAX D	Store accumulator content in memory pointed by DE pair.
410B	13		INX D	Increment DE pair
410C	05		DCR B	Decrement B content.
410D	C2		JNZ 4108	If zero flag is not set, jump to 4108
410E	08			
410F	41			
4110	CF		RST1	Stop the program
4200	OP1			Load operands into the memory locations specified.
4201	OP2			
4202	OP3			
4203	OP4			
4204	OP5			
4205	OP6			

Result:

input: 4200 00
4201 01
4203 02
4204 03
4204 04
4205 05

Output: 4300 00
4301 01
4303 02
4304 03
4304 04
4305 05

Program that transfers a block of data from one memory location to another executed successfully.

EX.NO:1.3 ADDING AN ARRAY OF BYTES

1.3.1 Aim:

To write an assembly language code that adds an array of unsigned integer bytes.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register B with the array size.

Step-3: Move the contents of memory pointed by HL pair to accumulator.

Step-4: Increment HL pair by 1.

Step-5: Add accumulator content with the content of memory pointed by HL pair and store the result in accumulator.

Step-6: Decrement B register content by 1

Step-7: If B is not zero, then go to step-4. Else, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	06		MVI B, 05	Move 05 to register B
4104	05			
4105	7E		MOV A, M	Move data in memory pointed by HL pair to accumulator
4106	23	LOOP	INX H	Increment HL pair content so that it points to the next memory location
4107	86		ADD M	Add accumulator content with the content of memory pointed by HL pair and store the result in accumulator
4108	05		DCR B	Decrement B register content
4109	C2		JNZ 4106	If zero flag is not set, then go to 4106
410A	06			
410B	41			
410C	32		STA 4500	Store the accumulator content in the memory location 4500
410D	00			
410E	45			
410F	CF		RST1	Stop the program
4200	OP1			Store operands in the memory locations specified
4201	OP2			
4202	OP3			
4203	OP4			
4204	OP5			

Result:

Input: 4200 00

4201 01

4202 02

4203 03

4204 04

Output: 4500 0A

Program that adds the array of unsigned integer bytes executed successfully.

1.3.2 Aim:

To write an assembly language code that adds an array of unsigned integer bytes with carry

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-B with the array size and register-C (to store carry) with 00

Step-3: Move the contents of memory pointed by HL pair to accumulator.

Step-4: Increment HL pair by 1.

Step-5: Add accumulator content with the content of memory pointed by HL pair and store the result in accumulator.

Step-6: If carry flag is set, go to step-7. Otherwise, increment register C content by 1.

Step-7: Decrement B register content by 1

Step-8: If B is not zero, then go to step-4. Else, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load Hl pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	06		MVI B, 05	Move 05 to register B
4104	05			
4105	0E		MVI C, 00	Move 00 to register C
4106	00			
4107	7E		MOV A, M	Move data in memory pointed by HL pair to accumulator
4108	23	LOOP1	INX H	Increment HL pair content so that it points to the next memory location
4109	86		ADD M	Add accumulator content with the content of memory pointed by HL pair and store the result in accumulator
410A	D2		JNC 410E	If carry flag is not set, go to 410E
410B	0E			
410C	81			
410D	0C		INR C	Increment C register content
410E	05	LOOP2	DCR B	Decrement B register content
410F	C2		JNZ 4106	If zero flag is not set, then go to 4108
4110	06			
4111	41			
4112	32		STA 4500	Store the accumulator content in the memory location 4500
4113	00			
4114	45			
4115	CF		RST1	Stop the program

Result:

Input: 4200 01

4201 02

4202 FC

4203 FE

4204 05

4205 06

Output: 4500 08 C 2

Program that adds the array of signed integer bytes executed successfully.

EX.NO:1.4 COUNT OF EVEN AND ODD NUMBERS IN THE GIVEN SERIES

Aim:

To write an assembly language code that counts the no. of even and odd bytes in the given array.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-C with the array size, register-B (to store odd count) with 00 and register-D (to store even count) with 00.

Step-3: Move the contents of memory pointed by HL pair to accumulator and rotate it right by 1 bit.

Step-4: If carry flag sets, increment B content. Otherwise, increment D content.

Step-5: Increment HL pair by 1 so that it points to next memory location.

Step-6: Decrement C content by 1.

Step-7: If C is not zero, then go to step-4. Else, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	0E		MVI C, 05	Move 05 to register C
4104	05			
4105	06		MVI B, 00	Move 00 to register B
4106	00			
4107	16		MVI D, 00	Move 00 to register D
4108	00			
4109	7E	L1	MOV A,M	Move content of memory pointed by HL pair to A
410A	0F		RRC	Rotate accumulator content right by 1
410B	DA		JC 4112	If carry flag sets, go to 4112 (L2)
410C	12			
410D	41			
410E	14		INR D	Increment register D content
410F	C3		JMP 4113	Go to 4113 (L3)
4110	13			
4111	41			
4112	04	L2	INR B	Increment register B content
4113	24	L3	INX H	Increment HL pair content so that it points to next location
4114	0D		DCR C	Decrement register C content
4115	C2		JNZ 4109	If zero flag is not set, go to 4109 (L1)
4116	09			
4117	41			
4118	76		HLT	Stop the program

Result:

Input: 4200 01

4201 02

4202 03

4203 04

4204 05

Output: B 03

D 02

Program that counts the no. of even and odd bytes in the given array, executed successfully.

EX.NO:1.5 COUNT OF POSITIVE AND NEGATIVE NUMBERS IN THE GIVEN ARRAY

Aim:

To write an assembly language code that counts the no. of positive and negative bytes in the given array.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-C with the array size, register-B (to store negative count) with 00 and register-D (to store positive count) with 00.

Step-3: Move the contents of memory pointed by HL pair to accumulator and rotate it left by 1 bit.

Step-4: If carry flag sets, increment B content. Otherwise, increment D content.

Step-5: Increment HL pair by 1 so that it points to next memory location.

Step-6: Decrement C content by 1.

Step-7: If C is not zero, then go to step-4. Else, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	0E		MVI C, 05	Move 05 to register C
4104	05			
4105	06		MVI B, 00	Move 00 to register B
4106	00			
4107	16		MVI D, 00	Move 00 to register D
4108	00			
4109	7E	L1	MOV A,M	Move content of memory pointed by HL pair to A
410A	07		RLC	Rotate accumulator content left by 1
410B	DA		JC 4112	If carry flag sets, go to 4112 (L2)
410C	12			
410D	41			
410E	14		INR D	Increment register D content
410F	C3		JMP 4113	Go to 4113 (L3)
4110	13			
4111	41			
4112	04	L2	INR B	Increment register B content
4113	24	L3	INX H	Increment HL pair content so that it points to next location
4114	0D		DCR C	Decrement register C content
4115	C2		JNZ 4109	If zero flag is not set, go to 4109 (L1)
4116	09			
4117	41			
4118	76		HLT	Stop the program

Result:

Input: 4200 FA

4201 02

4202 F0

4203 01

4204 FE

Output: B 03

D 02

Program that counts the no. of positive and negative numbers in the given array, executed successfully.

EX.NO:1.6 LARGEST NUMBER IN THE GIVEN SERIES OF BYTES

Aim:

To write an assembly language code that finds the largest number in the given series of bytes.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-C with the array size.

Step-3: Move the contents of memory pointed by HL pair to accumulator.

Step-4: Increment HL pair by 1 so that it points to next memory location.

Step-5: Compare data present in the memory location pointed by HL pair with accumulator.

If A<M; C = 1, Z=0

If A=M; C=0, Z=1

If A>M; C=0, Z=0

Step-6: If C is not zero, then go to step-7. Else, move data present in memory location pointed by HL pair to accumulator.

Step-7: Decrement C register content by 1.

Step-8: If zero flag is not set, go to step-4. Otherwise, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	0E		MVI C, 05	Move 05 to register C
4104	05			
4105	7E		MOV A, M	Move data in memory pointed by HL pair to A
4106	23	L2	INX H	Increment HL pair content
4107	BE		CMP M	Compare A with data in memory pointed by HL pair
4108	D2		JNC 410C	If carry flag is not set, go to 410C (L1)
4109	0C			
410A	41			
410B	7E		MOV A, M	Move data in memory pointed by HL pair to A
410C	0D	L1	DCR C	Decrement register C content
410D	C2		JNZ 4106	If zero flag is not set, go to 4106 (L2)
410E	06			
410F	41			
4110	76		HLT	Stop the program
4200	OP1			Store the operands in the memory locations specified.
4201	OP2			
4202	OP3			
4203	OP4			
4204	OP5			

Result:

Input: 4200 43

4201 63

4202 23

4203 72

4204 21

Output: A 72

Program that finds the largest number from the given series of bytes executed successfully.

EX.NO:1.7 SMALLEST NUMBER IN THE GIVEN SERIES OF BYTES

Aim:

To write an assembly language code that finds the smallest number in the given series of bytes.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-C with the array size.

Step-3: Move the contents of memory pointed by HL pair to accumulator.

Step-4: Increment HL pair by 1 so that it points to next memory location.

Step-5: Compare data present in the memory location pointed by HL pair with accumulator.

If A<M; C=1, Z=0

If A=M; C=0, Z=1

If A>M; C=0, Z=0

Step-6: If C is zero, then go to step-7. Else, move data present in memory location pointed by HL pair to accumulator.

Step-7: Decrement C register content by 1.

Step-8: If zero flag is not set, go to step-4. Otherwise, stop the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	0E		MVI C, 05	Move 05 to register C
4104	05			
4105	7E		MOV A, M	Move data in memory pointed by HL pair to A
4106	23	L2	INX H	Increment HL pair content
4107	BE		CMP M	Compare A with data in memory pointed by HL pair
4108	DA		JC 410C	If carry flag is set, go to 410C (L1)
4109	0C			
410A	41			
410B	7E		MOV A, M	Move data in memory pointed by HL pair to A
410C	0D	L1	DCR C	Decrement register C content
410D	C2		JNZ 4106	If zero flag is not set, go to 4106 (L2)
410E	06			
410F	41			
4110	76		HLT	Stop the program
4200	OP1			Store the operands in the memory locations specified.
4201	OP2			
4202	OP3			
4203	OP4			
4204	OP5			

Result:

Input: 4200 43

4201 63

4202 23

4203 72

4204 21

Output: A 21

Program that finds the smallest number from the given series of bytes executed successfully.

EX.NO:1.8 LENGTH OF THE GIVEN SERIES

Aim:

To write an assembly language code that finds the length of the given series of bytes.

Algorithm:

Step-1: Load the array of bytes onto consecutive memory blocks and make HL pair point to the first memory location of the array.

Step-2: Load register-C (keeps the count of bytes in given series) with 00.

Step-3: Move the contents of memory pointed by HL pair to accumulator.

Step-4: Compare accumulator content with FF (FF is given to specify the end of the series).

If A<M; C=1, Z=0

If A=M; C=0, Z=1

If A>M; C=0, Z=0

Step-5: If zero flag is set, go to step-7. Otherwise, increment C-register content by 1.

Step-6: Increment HL pair by 1 so that it points to next memory location and go to step-3.

Step-7: Stop.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Load HL pair with 4200 so that it acts as pointer to 4200 memory location
4101	00			
4102	42			
4103	06		MVI C, 00	Move 00 to register C
4104	00			
4105	7E	L2	MOV A, M	Move data in memory pointed by HL pair to A
4106	FE		CPI FF	Compare accumulator content with FF
4107	FF			
4108	CA		JZ 4110	If zero flag is set, go to 4110 (L1)
4109	10			
410A	41			
410B	0C		INR C	Move data in memory pointed by HL pair to A
410C	23	L1	INX H	Decrement register C content
410D	C3		JMP 4105	If zero flag is not set, go to 4106 (L2)
410E	05			
410F	41			
4110	76		HLT	Stop the program

Result:

Input: 4200 01
4201 02
4202 03
4203 04
4204 05
4205 06
4206 07
4207 08
4208 FF (End of the series)

Output: C 08 (Length of the series)

Program that finds the length of the given series of bytes executed successfully.

EX.NO:1.9 HEXADECIMAL TO BCD CONVERSION

Aim: To write an assembly language code that finds the equivalent BCD value for the given hexadecimal value.

Algorithm:

Step-1: Store the hex value in accumulator.

Step-2: Move 64 to register-B and make HL point to a memory location.

Step-3: Call BINBCD subroutine.

Step-4: Move 0A to register-B and call BINBCD subroutine.

Step-5: Stop

BINBCD subroutine:

Step-1: Move FF to memory pointed by HL pair

Step-2: Increment the content of memory pointed by HL pair.

Step-3: Subtract B register content from A and store the result in A

Step-4: If carry flag is not set, go to step-2

Step-5: Add A and B contents and store result in accumulator

Step-6: Increment HL pair so that it points to next memory location and return to main program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	31		LXI SP, 2000	Start stack at 2000
4101	00			
4102	20			
4103	3E		MVI A, OP	Store the hex value in accumulator
4104	OP			
4105	21		LXI H, 1000	Make HL pair point to 1000 memory location
4106	00			
4107	10			
4108	06		MVI B, 64	Move 64 to register B
4109	64			
410A	CD		CALL 4114	Call BINBCD subroutine at 4114
410B	14			
410C	41			
410D	06		MVI B, 0A	Move 0A to register B
410E	0A			
410F	CD		CALL 4114	Call BINBCD subroutine at 4114
4110	14			
4111	41			
4112	77		MOV M, A	Move contents of A to memory pointed by HL pair
4113	76		HLT	Stop the program
4114	36	BINBCD	MVI M, FF	Move FF to memory pointed by HL pair
4115	FF			
4116	34	L1	INR M	Increment contents of memory pointed by HL pair
4117	90		SUB B	Subtract B from A content and store the result in A
4118	D2		JNC 4116	If carry is not set, go to L1(4116)
4119	16			
411A	41			
411B	80		ADD B	Add b with A contents and store the result in A
411C	23		INX H	Increment Hl pair so that it points to next memory location
411D	C9		RET	Return to main program

Result:

Input: A FF

Output: 1000 02

1001 05

1003 05

Program that finds BCD value for the given hexadecimal value executed successfully.

EX.NO:1.10 BCD TO HEXADECIMAL CONVERSION

Aim:

To write an assembly language code that finds the equivalent hexadecimal value for the given BCD value.

Algorithm:

Step-1: Store the BCD value in A and in B

Step-2: Perform A AND 0F and store the result in register C

Step-3: Copy B content to A and perform A AND F0, rotate the result right by 1 bit 4 times and store the final result in register D.

Step-4: Clear A content and move 0A to E

Step-5: Add E D times and add C to the result thus obtained to get the final hexadecimal value.

. Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	31		LXI SP, 4200	Start stack at 4200 memory location
4101	00			
4102	42			
4103	21		LXI H, 2000	Make HL pair point to 2000 memory location
4104	00			
4105	20			
4106	7E		MOV A, M	Move data in memory pointed by HL pair to A
4107	CD		CALL 410B	Call the BINBCD subroutine
4108	0B			
4109	41			
410A	76		HLT	Stop the program
410B	47	BINBCD	MOV B, A	Move A content to register B
410C	E6		ANI 0F	A AND 0F and result stored in A
410D	0F			
410E	4F		MOV C, A	Move A content to register C
410F	78		MOV A, B	Move B content to accumulator
4110	E6		ANI F0	A AND F0 and result stored in A
4111	F0			
4112	0F		RRC	Rotate accumulator content right by 1 bit
4113	0F		RRC	
4114	0F		RRC	
4115	0F		RRC	
4116	57		MOV D, A	Move A content to register D
4117	AF		XRA A	Clear A content
4118	1E		MVI E, 0A	Move 0A to register E
4119	0A			
411A	83	L1	ADD E	Add A content with E content and store the result in A
411B	15		DCR D	Decrement D content
411C	C2		JNZ 411A	If zero flag is not set, go to L1 (411A)
411D	1A			
411E	41			
411F	81		ADD C	Add A content with C content and store the result in A
4120	C9		RET	Return to the main program

Result:

Input: 2000 72

Output: A 48

Program that finds hexadecimal value for the given BCD value executed successfully.

EX.NO:1.11 HEXADECIMAL TO ASCII CONVERSION

Aim:

To write an assembly language code that finds the equivalent ASCII value for the given hexadecimal value.

Algorithm:

Step-1: Store the hexadecimal value in accumulator.

Step-2: Compare accumulator value with 0A.

If A<0A; C=1 , Z=0

If A=0A; C=0, Z=1

If A>0A; C=0, Z=0

Step-3: Add 07 to accumulator content.

Step-4: If carry flag sets, add 30 to accumulator content.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200 memory location.
4101	00			
4102	42			
4103	7E		MOV A, M	Move data in memory pointed by HL pair to accumulator
4104	FE		CPI 0A	Compare data in accumulator with 0A
4105	0A			
4106	DA		JC 410B	If carry flag sets, go to L1(410B)
4107	0B			
4108	41			
4109	06		ADI 07	Add 07 to accumulator
410A	07			
410B	06	L1	ADI 30	Add 30 to accumulator
410C	30			
410D	76		HLT	Stop the program
4200	OP1			Store operand in 4200

Result:

Input: 4200 0F

Output: A 46

Program that finds equivalent ASCII value for the given hexadecimal value executed successfully.

EX.NO:1.12 ASCII TO HEXADECIMAL CONVERSION

Aim:

To write an assembly language code that finds the equivalent hexadecimal value for the given ASCII value.

Algorithm:

Step-1: Store the hexadecimal value in accumulator.

Step-2: Subtract 30 from the accumulator content.

Step-3: Compare accumulator value with 0A.

If A<0A; C=1, Z=0

If A=0A; C=0, Z=1

If A>0A; C=0, Z=0

Step-4: If carry flag does not set, subtract 07 from accumulator content, else stop program.

. Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200 memory location.
4101	00			
4102	42			
4103	7E		MOV A, M	Move data in memory pointed by HL pair to accumulator
4104	D6		SUI 30	Subtract 30 from A and store result in A
4105	30			
4106	FE		CPI 0A	Compare A content with 0A
4107	0A			
4108	DA		JC 410D	If carry flag sets, go to L1(410D)
4109	0D			
410A	41			
410B	D6		SUI 07	Subtract 07 from A and store the result in A.
410C	07			
410D	76	L1	HLT	Stop the program
4200	OP1			Store the operand in 4200

Result:

Input: 4200 41

Output: A 0A

Program that finds the equivalent hexadecimal value for the given ASCII value executed successfully.

EX.NO:1.13ADDING TWO 16-BIT NUMBERS

Aim:

To write an assembly language code that adds two 16-bit numbers.

Algorithm:

Step-1: Get the lower byte of 1st number into accumulator

Step-2: Add it with lower byte of 2nd number and store the result in memory pointed by HL pair, increment HL pair to point to next memory location

Step-3: Get the higher byte of 1st number into accumulator

Step-4: Add it with higher byte of 2nd number and carry of the lower bit addition

Step-5: Store the results in memory pointed by HL pair and hence result of sum of 2 16-bit numbers is stored in memory pointed by HL pair.

Step-6: Stop program execution.

.Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	AF		XRA A	Clear accumulator
4101	21		LXI H, 2001	Make HL pair point to 2001
4102	01			
4103	20			
4104	11		LXI D, 3001	Make DE pair point to 3001
4105	01			
4106	30			
4107	0E		MVI C, 02	Store 02 in C register
4108	02			
4109	7E	L1	MOV A, M	Move data in memory pointed by HL pair to A
410A	47		MOV B, A	Move data in A to B
410B	1A		LDAX D	Move data in memory pointed by DE pair to A
410C	88		ADC B	Add A and B with carry and store result in A
410D	77		MOV M, A	Move data in A to memory pointed by HL pair
410E	2B		DCX H	Decrement HL pair content
410F	1B0D		DCX D	Decrement DE pair content
4110	0D		DCR C	Decrement C register content
4111	C2		JNZ 4109	If zero flag is not set, go to L1(4109)
4112	09			
4113	41			
4114	76		HLT	Stop the execution of program
2000	OP1			
2001	OP1			
3000	OP2			
3001	OP2			

Result:

Input: 2000 34

2001 12

3000 78

3001 56

Output: 2000 AC

2001 68

Program that adds two 16-bit numbers executed successfully.

EX.NO:1.14 SUBTRACTING TWO 16-BIT NUMBERS

Aim:

To write an assembly language code that subtracts 2 16-bit numbers.

Algorithm:

Step-1: Get the lower byte of 1st number into accumulator

Step-2: Add it with lower byte of 2nd number and store the result in memory pointed by HL pair, increment HL pair to point to next memory location

Step-3: Get the higher byte of 1st number into accumulator

Step-4: Subtract higher byte of 2nd number from accumulator and carry of the lower bit subtraction

Step-5: Store the results in memory pointed by HL pair and hence result of difference of 2 16-bit numbers is stored in memory pointed by HL pair.

Step-6: Stop program execution.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	AF		XRA A	Clear accumulator
4101	21		LXI H, 2001	Make HL pair point to 2001
4102	01			
4103	20			
4104	11		LXI D, 3001	Make DE pair point to 3001
4105	01			
4106	30			
4107	0E		MVI C, 02	Store 02 in C register
4108	02			
4109	7E	L1	MOV A, M	Move data in memory pointed by HL pair to A
410A	47		MOV B, A	Move data in A to B
410B	1A		LDAX D	Move data in memory pointed by DE pair to A
410C	98		SBB B	subtract A and B with carry and store result in A
410D	77		MOV M, A	Move data in A to memory pointed by HL pair
410E	2B		DCX H	Decrement HL pair content
410F	1B0D		DCX D	Decrement DE pair content
4110	0D		DCR C	Decrement C register content
4111	C2		JNZ 4109	If zero flag is not set, go to L1(4109)
4112	09			
4113	41			
4114	76		HLT	Stop the execution of program
2000	OP1			
2001	OP1			
3000	OP2			
3001	OP2			

Result:

Input: 2000 34

2001 12

3000 78

3001 56

Output: 2000 44

2001 44

Program that subtracts two 16-bit numbers executed successfully.

EX.NO:1.15 2'S COMPLEMENT OF THE GIVEN NUMBER

Aim:

To write an assembly language code that finds the 2's complement of the given number

Algorithm:

Step-1: Store the operand in accumulator

Step-2: Complement the accumulator content

Step-3: Add 01 to accumulator content and store the result in accumulator

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200
4101	00			
4102	42			
4103	7E		MOV A, M	Move data in memory pointed by HL pair to A
4104	2F		CMA	Complement accumulator content
4105	C6		ADI 01	Add 01 to accumulator and store the result in accumulator
4106	01			
4107	76		HLT	Stop the program
4200	OP1			Store the operand in memory location specified

Result:

Input: 4200 02

Output: A FE

Program that finds the 2's complement of the given number executed successfully.

EX.NO:1.16 COMPARING 2 STRINGS

Aim:

To write an assembly language code that checks if 2 given strings are equal 9(or) not.

Algorithm:

Step-1: Store FF in register E and length of string in register D

Step-2: Store the strings in contiguous memory locations, make HL pair point to the first memory byte of the first string and make BC pair point to the firs memory byte of second string.

Step-3: Move data in memory pointed by BC pair to accumulator

Step-4: Compare contents of memory pointed by HL pair with accumulator content. If both are equal, then zero flag sets. Otherwise, it does not set.

Step-5: If zero flag does not set, go to step-7 , otherwise increment HL pair and BC pair to point to next memory locations

Step-6:Decrement register D content. If D content is not 0, go to step3.Otherwise move 00 to register E

Step-7: stop the execution

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200
4101	00			
4102	42			
4103	16		MVI D, 04	Move 04 to D
4104	04			
4105	1E		MVI E, FF	Move FF to E
4106	FF			
4107	01		LXI B, 4300	Make BC pair point to 4300
4108	00			
4109	43			
410A	0A	L2	LDAX B	Move context in memory pointed by BC pair to Accumulator
410B	BF		CMP M	Compare accumulator context with context of memory pointed by HL pair
410C	C2		JNZ 4116	If zero flag does not set, go to L3(4116)
410D	16			
410E	41			
410F	23		INX H,INX B	Increment HL pair, Increment BC pair
4110	15		DCR D	Decrement D
4111	C2		JNZ 410A	If zero flag does not set, go to L2(410A)
4112	0A			
4113	41			
4114	1E		MVI E, 00	Move 00 to E
4115	00			
4116	76	L3	HLT	Stop program

Result:

Input1: 4200 01

4201 02

4202 03

4203 04

4300 03

4301 02

4302 03

4303 04

Output1: E FF

Input2: 4200 01

4201 02

4202 03

4203 04

4300 01

4301 02

4302 03

4303 04

Output2: E 00

Program that checks if the given 2 strings are equal executed successfully

---

EX.NO:1.17 SEARCH SUB STRING FROM MAIN STRING

Aim:

To write an assembly language code that searches for a substring 04 in the given main string.

Algorithm:

Step-1: Store 00 in register E and length of string in register C

Step-2: Store the string in contiguous memory location and make HL pair point to the first memory byte of the string.

Step-3: Move data in memory pointed by HL pair to accumulator

Step-4: Compare contents of accumulator with 04. If accumulator content is 04, then zero flag sets. Otherwise, it does not set.

Step-5: If zero flag sets, go to step-7 , otherwise increment HL pair to point to next memory location

Step-6: Decrement register C content. If C content is not 0, go to step-3. Otherwise go to step-8

Step-7: Move FF to register E

Step-8: stop the execution

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200
4101	00			
4102	42			
4103	0E		MVI C, 04	Move 04 to C
4104	04			
4105	1E		MVI E, 00	Move 00 to E
4106	00			
4107	7E	L2	MOV A, M	Move data in memory pointed by HL pair to A
4108	FE		CPI 04	Compare data in A with 04
4109	04			
410A	CA		JZ 4115	If zero flag sets, go to L1(4115)
410B	15			
410C	41			
410D	23		INX H	Increment HL pair so that it points to next memory
410E	0D		DCR C	Decrement C register content
410F	C2		JNZ 4107	If zero flag does not set, go to L2(4107)
4110	07			
4111	41			
4112	C3		JMP 4117	Go to L3(4117)
4113	17			
4114	41			
4115	1E	L1	MVI E, FF	Move FF to register E
4116	FF			
4117	76	L3	HLT	Stop the execution

Result:

nput: 4200 FE
4201 50
4202 04
4203 41
Output: E FF
Input: 4200 75
4201 63
4202 FD
4203 01
Output: E 00

Program that searches the substring 04 in the given main string is executedsuccessfully

EX.NO:1.18 ADDING 5 BCD NUMBERS

Aim:

To write an assembly language code that adds 5 consecutive BCD numbers

Algorithm:

Step-1: Store operands in consecutive memory locations starting at 4200 and make HL pair point to 4200

Step-2: Move 05 to register B

Step-3: Move data in memory pointed by HL pair to accumulator

Step-4: Increment HL pair so that it points to next memory location

Step-5: Add data present in memory pointed by HL pair and accumulator content and store the result in accumulator

Step-6: Decrement register B content. If B content is not 0, go to step-3. Otherwise adjust accumulator content to BCD

Step-7: stop the execution

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21		LXI H, 4200	Make HL pair point to 4200
4101	00			
4102	42			
4103	06		MVI B, 05	Move 05 to B
4104	05			
4105	7E	L1	MOV A, M	Move data in memory pointed by HL pair to A
4106	23		INX H	Increment HL pair content to point to next memory
4107	86		ADD M	Add A and content of memory pointed by HL pair and store result in A
4108	05		DCR B	Decrement B register content
4109	C2		JNZ 4105	If zero flag is not set, go to L1(4105)
410A	05			
410B	41			
410C	27		DAA	Convert accumulator content to BCD
410D	76		HLT	Stop the program

Result:

Input: 4200 05

4201 01

4202 02

4203 03

4204 04

Output: A 15

Program that adds consecutive BCD numbers executed successfully

---

EX.NO:1.19SORTING GIVEN ARRAY OF BYTES

1.19.1 Aim:

To write an assembly language code to sort the given array of bytes in descending order

Algorithm:

Step-1: Initialize HL pair as memory pointer.

Step-2: Get the count at 4200 in to C register.

Step-3: Copy it in D register.

Step-4: Get the first vale in Accumulator.

Step-5: Compare it with the value at next location.

Step-6: If they are out of order, exchange the contents of accumulator and memory.

Step-7: Decrement D register's content by 1.

Step-8: Repeat steps 5 and 7 till the value in D register become zero.

Step-9: Decrement C register's content by 1.

Step-10: Repeat steps 3 to 9 till the value in C register becomes zero.

Step-11: Terminate the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4400	21		LXI H,4200	Load the array size to the HL pair
4401	00			
4402	42			
4403	4E		MOV C,M	Copy the array size to C register
4404	0D		DCR C	Decrement C by 1
4405	51	REPEAT	MOV D,C	Copy content of C to D register
4406	21		LXI H,4201	Load the first data to the HL pair
4407	01			
4408	42			
4409	7E	LOOP	MOV A,M	Copy the data to the accumulator
440A	23		INX H	Increment memory by 1
440B	BE		CMP M	Compare accumulator and memory content
440C	DA		JNC SKIP	Jump on no carry to the label SKIP
440D	14			
440E	44			
440F	46		MOV B,M	Copy memory content to B register
4410	77		MOV M,A	Copy accumulator content to memory
4411	2B		DCX H	Decrement memory by 1
4412	70		MOV M,B	Copy B register's content to memory
4413	23		INX H	Increment memory by 1
4414	15	SKIP	DCR D	Decrement D by 1
4415	C2		JNZ LOOP	Jump on non-zero to the label LOOP
4416	09			
4417	44			
4418	0D		DCR C	Decrement C by 1
4419	C2		JNZ REPEAT	Jump on non-zero to the label REPEAT
441A	05			
441B	44			
441C	76		HLT	Program ends

Result:

Input: 4200-05---------- Array Size

4201-01

4202-02

4203-03

4204-04

4205-05

Output: 4200-05---------- Array Size

4201-05

4202-04

4203-03

4204-02

4205-01

Program that sorts the given array of bytes in descending order executed successfully.

1.19.2 Aim:

To write an assembly language code to sort the given array of bytes in ascending order.

Algorithm:

Step-1: Initialize HL pair as memory pointer.

Step-2: Get the count at 4200 in to C register.

Step-3: Copy it in D register.

Step-4: Get the first vale in Accumulator.

Step-5: Compare it with the value at next location.

Step-6: If they are out of order, exchange the contents of accumulator and memory.

Step-7: Decrement D register's content by 1.

Step-8: Repeat steps 5 and 7 till the value in D register become zero.

Step-9: Decrement C register's content by 1.

Step-10: Repeat steps 3 to 9 till the value in C register becomes zero.

Step-11: Terminate the program.

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4400	21		LXI H,4200	Load the array size to the HL pair
4401	00			
4402	42			
4403	4E		MOV C,M	Copy the array size to C register
4404	0D		DCR C	Decrement C by 1
4405	51	REPEAT	MOV D,C	Copy content of C to D register
4406	21		LXI H,4201	Load the first data to the HL pair
4407	01			
4408	42			
4409	7E	LOOP	MOV A,M	Copy the data to the accumulator
440A	23		INX H	Increment memory by 1
440B	BE		CMP M	Compare accumulator and memory content
440C	DA		JC SKIP	Jump on carry to the label SKIP
440D	14			
440E	44			
440F	46		MOV B,M	Copy memory content to B register
4410	77		MOV M,A	Copy accumulator content to memory
4411	2B		DCX H	Decrement memory by 1
4412	70		MOV M,B	Copy B register's content to memory
4413	23		INX H	Increment memory by 1
4414	15	SKIP	DCR D	Decrement D by 1
4415	C2		JNZ LOOP	Jump on non-zero to the label LOOP
4416	09			
4417	44			
4418	0D		DCR C	Decrement C by 1
4419	C2		JNZ REPEAT	Jump on non-zero to the label REPEAT
441A	05			
441B	44			
441C	76		HLT	Program ends

Result:

Input: 4200-05---------- Array Size

4201-05

4202-04

4203-03

4204-02

4205-01

Output: 4200-05---------- Array Size

4201-01

4202-02

4203-03

4204-04

4205-05

Program that sorts the given array of bytes in ascending order executed successfully.

INTRODUCTION OF 8086/88E KIT:

ESA 86/88E is an economical and powerful general-purpose microcomputer system that can be operated with 8086 or 8088 CPU that may be used as an instructional and learning aid and also as a development tool in R & D labs and industries. 8086 and 8088 are third generation CPUs from INTEL that differ primarily in their external data paths. 8088 uses an 8-bit wide data bus while 8086 uses a 16-bit wide data bus. ESA 86/88E can be operated with either CPU and the only is possible difference would be in the speed of execution (with 8088 CPU, a small speed degradation occurs because of the 8-bit wide data bus). In either case, the CPU is operated in maximum mode.

The basic system can be easily expanded through the system Bus connector. Powerful features like monitor resident Symbolic One-line assembler and Disassemble simplifies the programmer's task of entering Assembly language programs. On-board provision for 8087 Numeric Data Processor makes ESA 86/88E useful for number-crunching applications also. Onboard battery backup provision for RAM is made to retain the user programs in the event of a power failure or when the trainer is powered OFF. The trainer also features onboard Programmable Peripheral Interfaces, Programmable Interval Timers, USART (for serial communication), PC keyboard controller and parallel printer interface. Further, ESA 86/88E firmware also supports ESA EPROM Programmer interface.

ESA 86/88E can be operated on single +5 Volts power supply in stand-alone mode using LCD and optional PC/AT keyboard, or in serial mode with a host computer through its RS-232C interface. User-friendly Widows and DOS Driver packages supplied with ESA 86/88E provide for a powerful and versatile Assembly level programming/debugging environment.

SYSTEM CAPABILITIES:

- Assemble 8086/8088 Instruction Mnemonics using ESA 86/88E Symbolic One-Line Assembler.
- Disassemble HEZ bytes from memory into 8086/88 BPU instructions using monitor resident Disassembler.
- Perform fast numerical computations using the optional 8087 Numeric Data Processor.
- Execute the user program at full speed or debug the program through Single Step and Breakpoint facilities.

- Examine/Modify the contents of memory locations in byte or word format.
- Examine/Modify the content of CPU registers.
- Write or read data to or from I/O ports (byte or word format).
- Operations on blocks of memory such as filling a block of memory with a constant byte or word data, comparing a block of memory with another block, copying a block of data or program within the memory and displaying memory blocks in byte or word format.
- Communicate with a Host PC serially through RS232C interface at a baud rate of up to 19200 and develop/debug applications using the user-friendly Windows or DOS driver packages.
- Supports for downloading user programs into ESA from a host computer system in Intel HEX as well as Intel Extended HEX format.
- Support for uploading user programs to Host Computer system and saving them as HEX files on a system.
- Read, Program, verify and Blank check of popular EPROM s Programmer interface module.
- Use the monitor resident Centronics compatible Parallel printer driver software and obtain hard copies of Serial mode operations.

SPECIFICATIOS:

- Central Processor: 8086 or 8088 CPU operating at 5 MHz in maximum mode.
(Supplied with 8086 CPU)
- Co-Processor: On-board 8087 Numeric Data processor.
- Memory: ESA 86/88E provides a total of 128 Kbytes of onboard memory.
- 64K Bytes of ROM using two 27256 EPROMS.
- 64K Bytes of RAM using two 62256 Static RAM s.

ONBOARD PERIPHERALS AND INTERFACING OPTIONS:

- **8251A**:Universal Synchronous/Asynchronous Receiver/Transmitter supporting standard baud rates from 110 to 19,200. Baud rate is selected through on-board DIPswitch setting.
- **8253-5:**Programmable Interval Timer; Timer 0 is used for Baud clock generation. Timer 1 and Timer 2 are available to the user.
- **8255A:** 3 Programmable Peripheral Interfaces provide up to 72 Programmable I/O lines. One 8255 is used for controlling LCD and reading DIP Switch. Two 8255s are for the user, of which one is populated by default and the other is optional.
- **8288 –** Bus Controller used for generating control signals in Maximum Mode operation.
- **8042/8742 UPI** (Universal Peripheral Interface)
- **Interrupts**

External:

- **NMI:** 8086/8088 Type 2 Interrupt connected to KBINT key on the trainer. The vectoring information for this interrupt is fully user-defined.
- **INTR:** Available to user on system expansion connector J6.

Internal: Interrupt Vectors 1 (Single step Interrupt) and 3 (Breakpoint Interrupt) reserved for monitor.

- **External Interface Signals**

CPU Bus: De-multiplexed and fully buffered, TTL compatible, Address, Data and Control signals are available on two 26-pin ribbon cable connectors.

Parallel I/O: 48 programmable parallel I/O lines (TTL Compatible) through two 26-pin ribbon cable connectors. Note that only one 8255 and its corresponding 26-Pin ribbon cable connector is available as default factory installation, which may additionally be used as a parallel printer interface. Further, ESA 86/88E firmware uses this 8255 for operations with ESA EPRON Programmer interface also.

Serial I/O: RS 232C through on-board 9 pin D-type female connector.

PC Keyboard: PS/2 connector is provided for interfacing PC Keyboard.

20 x 4 LCD: 15-Pin flow-strip for interfacing the 20x4 LCD.

Timer Signals: Timer 1 and Timer 2 signals are brought to a header.

Power Supply: + 5V @ 1A (approx.)

Battery backup: 3.6V Ni-Cd battery as power backup to RAM (optional).

ESA 86/88E MONITOR COMMANDS

COMMAND	FUNCTION	FORMAT/SYNTAX
S	Substitute Memory bytes: Displays/Modifies memory bytes	**S [<address>]<CR>[,/[<new data>],]*<CR>**
SW	Substitute Memory Words: Displays/modifies memory words.	**SW [<address>]<CR>[, /[<new data>],]*<CR>**
D	Display Memory bytes: Displays block of memory in word format	**D<start address>[,<end address>] <CR>**
DW	Display Memory words: Displays block of memory in word format	**DW<start address>[,<end address>] <CR>**
X	Examine/modify registers: Displays modifies 8086/8088CPU registers	**X [<reg><CR>[<new data>/,]]<CR>**
M	Move Memory: Copies a block of memory from one location to the other	**M<start address>,<end address>,<destination address><CR>**
F	Fill Memory (Byte): Fills a block of Memory with constant byte data	**F<start address>,<end address>,<byte value><CR>**
FW	Fill Memory (Words): Fill a block of Memory with constant Word data	**FW<start address>,<end address>,<byte value><CR>**
I	Input byte: Accepts and displays the data byte at the input port	**I<port address><CR>[,]*<CR>**
IW	Input Word: Accepts and displays the data word at the input port	**IW<port address><CR>[,]*<CR>**

O	Output byte: Outputs a data byte to the output port	**O<port address><CR>[<data>/,]*<CR>**
OW	Output byte: Outputs a data word to the output port	**OW<port address><CR>[<data>/,]*<CR>**
C	Compare Memory: Compares a block of memory with another block	**C<start address1>,<end address1>,<start address2><CR>**

N	Single Step: Executes single instruction of the user program	**N<CR>[<start address>,],[,/<new address>]*<CR>**
H	Help Command: Lists Monitor commands with their valid syntax.	**H [<Command mnemonic>]<CR>**
P*	Invoke Programmer Software: Invokes the software for ESA EPROM Programmer Interface	**P<CR>**
A**	Enter Assembler: Invokes EAS 86/88E Symbolic One-line Assembler	**A [address]<CR>**
LL**	List Labels: Lists all labels defined in the symbol table	**LL<CR>**
LC**	Label Clear: Clears all previously defined label from the Symbol Table	**LC<CR>**
Z**	Disassembly Command: Disassemble Hex code into 8086 mnemonics for a specific memory range	**Z [<start address>[, <end address>]]<CR>**

INTRODUCTION TO MASM

1. The first step in the development process is to write an assembly language program, can be written with an ordinary text editor.
2. Assembler translates a source file that was created using the editor, into machine language such as binary or object code
3. The Assembler generates two files on the floppy or hard disk during these two passes.
4. The first file is called as object file. This file contains the binary codes for the instructions and information about the addresses of the instructions.
5. The second file generated by the assembler is called assembler list file. This file contains the assembly language statements, The binary code for each instruction and the offset for each instruction.

C:\MASM\BIN\>MASM myprog.asm;

Where myprog.asm is the name of the .asm file which is to be converted to .obj file

LINKING PROCESS:

A Linker is a program used to join together several object files into one large object file
The command on command prompt for converting .obj file to .EXE file as given below:

C : \ MASM \ BIN \ > LINK myprog.obj

DEBUGGING PROCESS:

A debugger is a program, which allows us to load our object code program into system memory, execute the program, and debug it.

DEBUG COMMANDS:

Command	Command Syntax and Description
Assembler	- A [address] A Command allows you to enter the mnemonic, or human – readable, instructions directly
GO	- G [= address] [addresses] G Command executes the program in the memory.
Quit	- Q Q Command quits the debug.
Register	- R [register] R Command displays the register contents on the screen.
Unassemble	- U [range] U Command translates memory into assembly language mnemonics.

1.20 MULTIBYTE ADDITION AND SUBTRACTION

1.20.1 AIM: To perform addition of 2 signed and unsigned words

ALGORITHM:

1. Get 1st operand from memory that is pointed by source index [SI] register and store it in AX register.
2. Get 2nd operand from memory and store it in BX register.
3. Add contents of AX and BX registers and the result will be in AX register.
4. Store the result in memory location pointed by DI.

PROGRAM:

```
data SEGMENT
src dw 1234H, 3456H
dst dw ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
MOV AX, data
MOV DS,AX
MOV AX,0000H
MOV SI OFFSET src
MOV DI OFFSET dst
MOV AX, [SI]   ;get 1st number in to accumulator
INC SI                          ; Increment SI twice to get next number
INC SI
MOV BX,[SI]                     ;Move second number in to BX
ADD AX,BX                       ; Add two operands
MOV [DI], AX                    ; Store the result in memory
INT 03                          ;it is a software interrupt which tells the
                                 processor to stop execution
```

Result:

Input- src1234H, 3456H
Output - dst 468AH

1.20.2 AIM: To perform subtraction of 2 signed and unsigned words

ALGORITHM:

1. Get 1st operand from memory that is pointed by source index [SI] register and store it in AX register.
2. Get 2nd operand from memory and store it in BX register.
3. subtract contents of AX and BX registers and the result will be in AX register.
4. Store the result in memory location pointed by DI.

PROGRAM:

```
data SEGMENT
src dw 3456H, 1234H
dst dw ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
MOV AX, data
MOV DS, AX
MOV AX, 0000H
MOV SI OFFSET src
MOV DI OFFSET dst
MOV AX, [SI]   ;get 1st number in to accumulator
INC SI                          ; Increment SI twice to get next number
INC SI
MOV BX,[SI]                     ;Move second number in to BX
SUB AX,BX                       ; subtract two operands
MOV [DI], AX                    ; Store the result in memory
INT 03                          ;it is a software interrupt which tells the
                                 processor to stop execution
```

Result:

Input- src1234H, 3456H
Output - dst 2222H

1.21MULTIBYTE MULTIPLICATION

AIM: To perform multiplication of two words of signed and unsigned numbers and observe the result after executions.

ALGORITHM:

1. Get 1st operand from memory that is pointed by source index [SI] register and store it in AX register.
2. Get 2nd operand from memory and store it in BX register.
3. Multiply contents of AX and BX registers and the result will be in AX&DX registers.
4. Store the result in memory location pointed by DI.

PROGRAM:

```
data SEGMENT
src dw 3456H, 1234H
dst dd ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
MOV AX, data
MOV DS, AX
MOV AX, 0000H
MOV SI OFFSET src
MOV DI OFFSET dst
MOV AX, [SI]   ;get 1st number in to accumulator
INC SI                                   ; Increment SI twice to get next number
INC SI
MOV BX,[SI]                              ;Move second number in to BX
MUL BX                                   ;multiply two operands
MOV [DI], AX
ADD DI,0002
MOV [DI],DX                              ; Store the result in memory
INT 03
Code ENDS  ;it is a software interrupt which tells  the
Processor to stop execution
```

NOTE: IMUL instruction used for signed multiplication

Result:

Input- src1234H, 3456H

Output - dst 4AD0E1H

1.22MULTI BYTE DIVISION

AIM: To perform division of word with a byte.

ALGORITHM:

1. Get 1st operand from memory, which is pointed by source index SI register and store it in AX register.
2. Get 2nd operand from memory and store it in BL register.
3. Divide contents of AX with BL register and the results will be in AX register.
4. Store the result in memory location pointed by DI.

PROGRAM:

```
data SEGMENT
dividenddw0044H
divisordb04H
 quotient db ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
MOV AX, data
MOV DS, AX
MOV AX, 0000H
MOV SI OFFSET dividend
MOV DI OFFSET divisor
MOV AX, [SI]  ;get 1st number in to accumulator
MOV BL,[DI]                    ;Move second number in to BL
DIV BL                         ;Division with byte
INC DI
MOV [DI], AL; Store the result in memory
INT 03                         ;it is a software interrupt which tells the
                                Processor to stop execution
```

NOTE: IDIV instruction used for signed division

Result:

Input- Dividend : 44H ; Divisor : 04H
Output - Quotient : 11H

1.23BCD TO HEXA DECIMAL CONVERSION

AIM: To convert BCD number in to Hexadecimal number.

ALGORITHM:

1. Initialize AX with zeros.
2. Make SI as a pointer to the location, where BCD number is stored.
3. Move the BCD number to AL and BL register.
4. Logical AND the contents of BL with 0F the result will be in BL register.
5. Logical AND the contents of AL with F0 and the result will be in AL register.
6. Rotate the contents of AL with CL times.
7. Load BH with 0A and multiply AL contents with BH contents.
8. Add contents AX and BX and result is in AX register.
9. Store the contents of AX in to memory.

PROGRAM:

```
data SEGMENT
bcdno db 72H
hexno db ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
MOV DS, AX
MOV AX, 0000H
MOV SI OFFSET bcdno
MOV DI OFFSET hexno
MOV AL,[SI]
MOV BL, AL
AND BL,0FH
AND AL,FOH
    MOV CL,04H
    ROR AL,CL
    MOV BH,0AH
    MUL BH
   ADD AL,BL
```

```
    MOV [DI,]AL
    INT 03
code ENDS
END start
```

Result:

Input :bcdno=72H

Output: hxeno=48H

1.24CONVERTION OF HEXADECIMAL NUMBER TO DECIMAL NUMBER

AIM: To find the decimal number from the given hexadecimal numbers

ALGORITHM:

1. Initialize AX with zero.
2. Make SI as a pointer to the location, where hex number is stored.
3. Divide the hex number with 64H to find number of hundreds in the hex number.
4. Store the number of hundreds in the memory.
5. Store the remainder in AL. Divide it with 0A, to find number of 10's in the number.
6. Store the number of 10's in to next memory location.
7. Remainder obtained in the units place of the decimal number, store that in to next memory location.

PROGRAM:

```
data SEGMENT
hexnodbFFH
bcdnodb    ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
MOV DS, AX
MOV AX, 0000H
        MOV SI,OFFSET hexno
        MOV DI OFFSET bcdno
         MOV AL,[SI]
MOV BL,64H
DIV BL
MOV [DI], AL
MOV AL,AH
         MOV AH,00H
          MOV BL,0AH
         DIV BL
          INC DI
```

```
        MOV [DI], AL
        INC DI
        MOV [DI],AH
        INT 03
code ENDS
END start
```

Result:

Input: hexno=FFH

Output:bcdno=255

1.25 CONVERSION OF PACKED BCDNUMBER TO UNPACKED BCDNUMBER

AIM:Program to convert packed BCD to unpacked BCD.

ALGORITHM:

1. Initialize SI as a pointer to the memory location where packed BCD number is stored.
2. Initialize DI as a pointer to memory location where unpacked BCD number is to be stored.
3. Get the packed BCD number in to AL and BL register.
4. Load CL register with 04 and rotate BL contents with CL times.
5. By logical AND the AL and BL contents 0F the packed BCD is converted into unpacked BCD.

PROGRAM:

```
data SEGMENT
decnodb25H
bcdnodb    ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
MOV DS, AX
MOV AX, 0000H
        MOV SI,OFFSET decno
        MOV DI OFFSET bcdno
         MOV AL,[SI]
         MOV CL,04
MOV AL,[SI]
MOV BL,AL
ROL BL,CL
AND AL,0FH
AND BL,0FH
MOV [DI],BL
INC DI
MOV [DI],AL
```

INT 03

code **ENDS**

END start

RESULT:

Input: decino=23

Output :unpacked bcd no 02,03

1.26 MOVE BLOCK

AIM:To execute a program to move string of bytes from one location to the other location in the memory.

ALGORITHM:

1. Make SI as a pointer to the source block.
2. Make DI as a pointer to the destination block.
3. Store the number of bytes to transfer from one block to another block in CX.
4. Using MOVSB transfer byte from one location to another location.
5. Repeat step '4', count number of times in CX.

PROGRAM:

```
data SEGMENT
src db 25H,23H,21H,20H
dst db ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
MOV DS, AX
MOV AX, 0000H
        CLD
        MOV SI,OFFSET src
        MOV DI OFFSET dst
        MOV CL ,04
  L1: MOVSB
      LOOP L1
code ENDS
END start
```

RESULT:

Input: src=25H,23H,21H,20H

Output:dst=25H,23H,21H,20H

1.27 REVERSE STRING

AIM: To execute a program to obtain reverse of a given string

ALGORITHM:

1. Make SI as pointer to Data segment i.e., DS:SI and make DI as pointer to extra segment i.e.,ES:DI .
2. Load CX register with the count.
3. Clear the direction flag, get data using SI to AL.
4. Set direction flag and store AL content into a location pointed out by DI.
5. Decrement the count and check for zero condition. If it is not zero go to step 3, else continue.

PROGRAM:

```
data SEGMENT
src db 25H,23H,21H,20H
data ENDS
extra SEGMENT
dst db ?
extra ENDS

code SEGMENT
ASSUME CS: code, DS: data,ES:extra
Start: MOV AX, data
MOV DS, AX
MOV AX, extra
MOV ES, AX
MOV CL, 04H
    MOV SI ,OFFSET src
    MOV DI, OFFSET dst
L1: CLD
    ADD DI,0004H
    LODSB
    STD
    STOSB
   LOOP L1
```

INT 03

code **ENDS**

END start

RESULT:

Input: src=25H,23H,21H,20H

Output: dst=20H.21H,23H,25H

1.28 SORTING

AIM: To perform the ascending order by the assembly language programming on 8086 kit.

ALGORITHM:

1. Load CX and BX registers with a length of array.
2. Get 1st number from memory to AX register.
3. Compare AX register contents with the contents of next memory location.
4. Check for carry condition if there is no carry exchange the contents of AX register and [SI] 02 memory location.
5. If there is a carry increment SI by two times to get next number.
6. Decrement counter
7. If counter is not zero go to step 3, else continue.
8. Decrement the contents of CX, if CX is not zero go to step 2, else continue.

PROGRAM:

```
data SEGMENT
src dw 2525H,2323H,2120H,2023H
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
MOV DS, AX
MOV CL, 04H
 L1:   MOV SI ,OFFSET src
       MOV BX,CX
L2: MOV AX,[SI]
     DEC BX
    CMP AX,[SI]02
     JB L3
   XCHG  AX,[SI]02
   XCHG  AX,[SI]
L3:INC SI
    INC SI
    DEC BX
```

```
    JNE L2
    LOOP L1
    INT 03
code ENDS
END Start
```

RESULT:

Input: src=2525H,2323H,2120H,2023H

Output src=2023H,2120H,2323H,2525H

1.29 LENGTH OF THE STRING

AIM: To execute program for finding the length of the string.

ALGORITHM:

1. Load CX register with 0000 to represent the length of string.
2. Make SI as pointer to data segment.
3. Get content from memory location and compare with 00.
4. If they are not equal repeat step 3.
5. Increment CX contents and get next number again compare with 00.
6. Finally the length is in CX register.

PROGRAM:

```
data SEGMENT
src db 25H,23H,21H,20H,FFH
dst db ?
data ENDS
code SEGMENT
ASSUME CS: code, DS: data
Start: MOV AX, data
       MOV DS, AX
       MOV CL, 00H
       MOV SI ,OFFSET src
       MOV DI, OFFSET dst
   L1:LODSB
      CMP AL, FFH
      JE L2
      INC CL
      JMP L1
   L2: MOV [DI], CL
       INT 03
Code ENDS
END start
```

RESULT:

Input: src = 25H,23H,21H,20H,FFH

Output:dst=04H

1.30 STRING COMPARISION

AIM: To execute a program to compare the string bytes given in two different memory Locations.

ALGORITHM:

1. Make SI as pointer to data segment and DI as pointer to extra segment.
2. Load CL with the count of string and clear direction flag.
3. Compare the two strings in SI and DI if both are equal store 00 in AL register, else store FF.

PROGRAM:

```
data SEGMENT
src db 25H,23H,21H,20H
data ENDS
extra SEGMENT
dst db 25H,23H,21H,20H
extra ENDS
code SEGMENT
ASSUME CS: code, DS: data, ES: extra
Start: MOV AX, data
MOV DS, AX
MOV AX, extra
MOV ES, AX
MOV CL, 04H
    MOV SI, OFFSET src
    MOV DI, OFFSET dst
    CLD
    REP
    CMPSB
    JCXZ  L1
    MOV AL 0FFH
    JMP XX
L1: MOV AL,00H
XX: INT 03
code  ENDS
```

END start

RESULT:

Input:src =25H,23H,21H,20H

dst= 25H,23H,21H,20H

Output: CL= 00H

1.31 SEARCH STRING

AIM: To execute program for searching a byte string.

ALGORITHM:

1. Load CX register with 0005 to represent the length of string.
2. Make DI as pointer to extra segment.
3. Get content from memory location and compare with AL content.
4. If they are not equal repeat step 3.
5. Decrement CX contents and get next number again compare with AL.
6. Finally AL content will represents the presence of AL content in DI location.

PROGRAM:

```
data SEGMENT
src db 25H,23H,,F0H,21H
data ENDS
code SEGMENT
ASSUME CS: code, ES data,
Start: MOV AX, data
MOV ES, AX
MOV DI,OFFSET src
        MOV CL 04H
        MOV AL, 0F0H
  L1: SCASB
      LOONE L1
      JCXZ  L2
      MOV AL,00H
       JMP L3
L2:MOVAL,0FF
     INT 03
code ENDS
END start
```

RESULT:

Input:src =25H,23H,F0H,20H

Output: AL= 00H

1.32 NEAR/FAR PROCEDURE IMPLEMENTATION

AIM;- BCD to Binary conversion program that uses a near procedure to convert BCD number to Binary.

ALGORITHM:

1. Separate nibbles
2. Save lower nibble (don't multiply by 1)
3. Multiply upper nibble by 0A H
4. Add lower nibble to result of multiplication

PROGRAM:

```
data SEGMENT
src db 25H
dst db ?
data ENDS
stack SEGMENT
100 dbdup (0)
LABEL Top stack
stack ENDS
Code SEGMENT
ASSUME CS: code, DS: data ,SP: stack
Start: MOV AX, data
MOV DS, AX
MOV SI,OFFSET src
        MOV SP ,OFFSET Top stack
MOV  AL , [SI]                  ; get input value in to AL register
CALL   CONVE ;Calls  the procedure CONVE
        MOV DI, OFFSET dst              ; Make DI as pointer to  destination
MOV [DI],AL                     ; Loads  AL content into destination  location
INT 03
```

CONVERSION ROUTINE:

```
CONVE: MOV BL, AL; save copy of BCD in BL
       AND BL, 0F H          ; Mask higher nibble
AND  AL,0F0 H                ; Mask lower  nibble
MOV CL,04                    ; Count value for bringing higher nibble in to lower nibble
ROR  AL ,CL
       MOV  BH,0A H          ;Multiply upper BCD digit with 0A
ADD  AL,BL                   ;Add lower BCD digit to result
       RET                   :Binary result  in AL
Code ENDS
```

RESULT:

Input: 2000 72

Output: A 48

Program that finds hexadecimal value for the given BCD value executed successfully.

NOTE: For FAR procedure implementation write main program and procedure in different segments.

Viva Questions:

1. How many number of 16 bit registers in 8085?
2. How many flags defined in 8086?
3. How many 8085 compatible flags in 8086?
4. What is word size in 8086 processor?
5. Mention number of instructions in 8085.

EX.NO:2 INTERFACING AND PROGRAMMING 8255

APPARATUS:

8255, interface card, 8085 processor kit

THEORY:

The 8255 is a 40 pin integrated circuit (IC), designed to perform a variety of interface functions in a computer environment.

Operational modes of 8255:

There are two basic operational modes of 8255

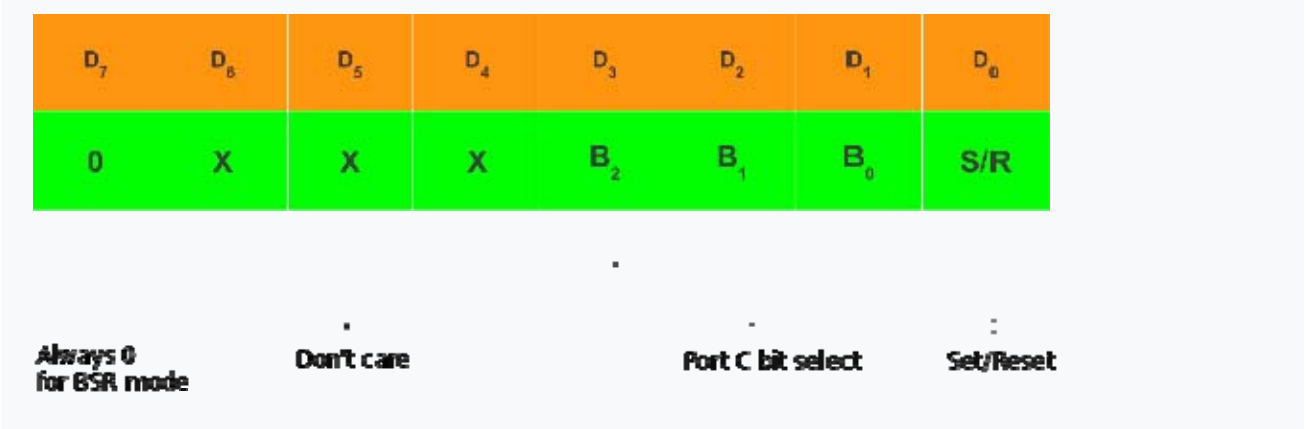
Bit set/reset mode (BSR mode)

Input /Output mode (I/O mode).

The two modes are selected on the basis of the value present at the D_7 bit of the control word register. When $D_7 = 1$, 8255 operates in I/O mode, and when $D_7 = 0$, it operates in the BSR mode.

Bit set/reset (BSR) mode:

The Bit Set/Reset (BSR) mode is applicable to port C only. Each line of port C (PC_0 - PC_7) can be set/reset by suitably loading the control word register. BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode.



8255 BSR mode

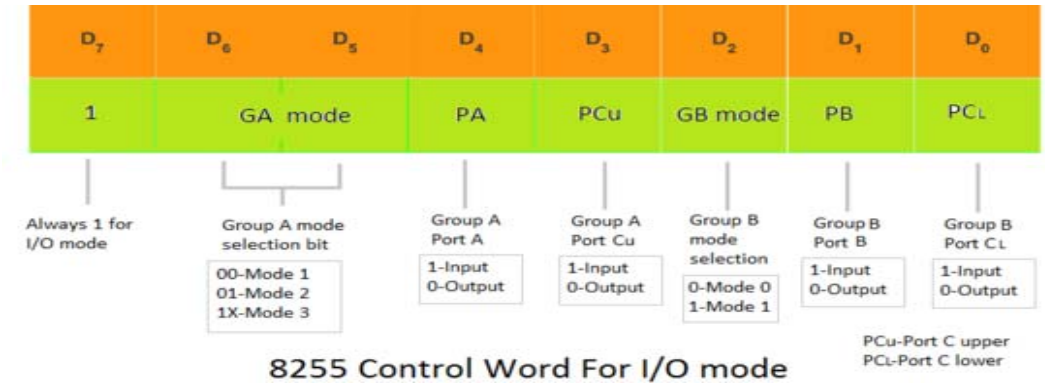
- D_7 bit is always 0 for BSR mode.
- Bits D_6, D_5 and D_4 are don't care bits.
- Bits D_3, D_2 and D_1 are used to select the pin of Port C.
- Bit D_0 is used to set/reset the selected pin of Port C.

Selection of port C pin is determined as follows:

Input /Output mode:

This mode is selected when D_7 bit of the Control Word Register is 1. There are three I/O modes

1. Mode 0 - Simple I/O
2. Mode 1 - Strobed I/O
3. Mode 2 - Strobed Bi-directional I/O



8255 Control Word For I/O mode

Mode 0 - simple I/O

In this mode, the ports can be used for simple I/O operations without handshaking signals. Port A, port B provides simple I/O operation. The two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports. Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.

The input/output features in mode 0 are as follows:

1. Output ports are latched.
2. Input ports are buffered, not latched.
3. Ports do not have handshake or interrupt capability.
4. With 4 ports, 16 different combinations of I/O are possible.

Mode 0 – input mode

- In the input mode, the 8255 gets data from the external peripheral ports and the CPU reads the received data via its data bus.
- The CPU first selects the 8255 chip by making CS low. Then it selects the desired port using A_0 and A_1 lines.
- The CPU then issues an RD signal to read the data from the external peripheral device via the system data bus.

Mode 0 - output mode

- In the output mode, the CPU sends data to 8255 via system data bus and then the external peripheral ports receive this data via 8255 port.
- CPU first selects the 8255 chip by making CS low. It then selects the desired port using A_0 and A_1 lines.
- CPU then issues a WR signal to write data to the selected port via the system data bus. This data is then received by the external peripheral device connected to the selected port.

Mode 1

When we wish to use port A or port B for handshake (strobe) input or output operation, we initialise that port in mode 1 (port A and port B can be initialized to operate in different modes, i.e., for e.g., port A can operate in mode 0 and port B in mode 1). Some of the pins of port C function as handshake lines.

For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

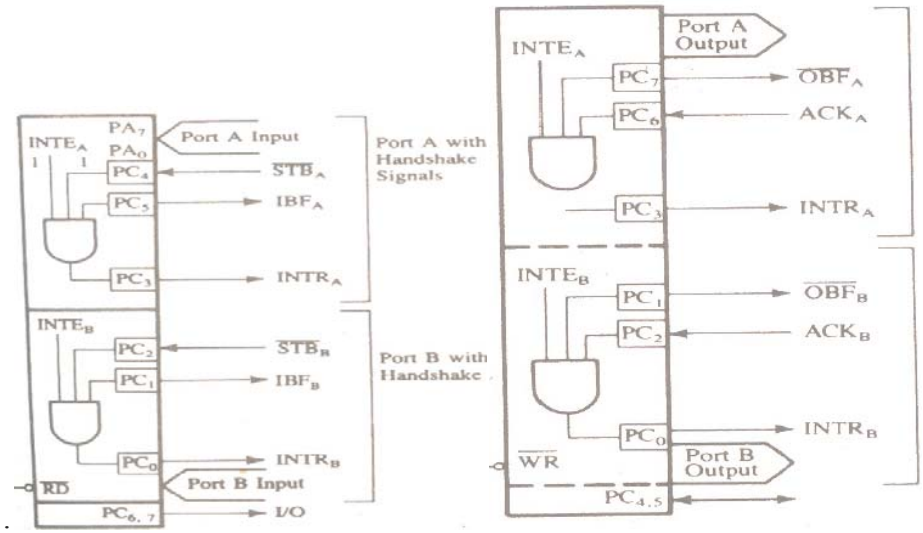
If port A is initialised as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking has following features:

1. Two ports i.e. port A and B can be used as 8-bit I/O ports.
2. Each port uses three lines of port c as handshake signal and remaining two signals can be used as i/o ports.
3. Interrupt logic is supported.
4. Input and Output data are latched.

Input Handshaking signals

1. IBF (Input Buffer Full) - It is an output indicating that the input latch contains information.
2. STB (Strobed Input) - The strobe input loads data into the port latch, which holds the information until it is input to the microprocessor via the IN instruction.
3. INTR (Interrupt request) - It is an output that requests an interrupt. The INTR pin becomes a logic 1 when the STB input returns to a logic 1, and is cleared when the data are input from the port by the microprocessor.
4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed via the port PC4(port A) or PC2(port B) bit position



Output Handshaking signals:

1. OBF (Output Buffer Full) - It is an output that goes low whenever data are output(OUT) to the port A or port B latch. This signal is set to logic 1 whenever the ACK pulse returns from the external device.
2. ACK (Acknowledge)-It causes the OBF pin to return to a logic 1 level. The ACK signal is a response from an external device, indicating that it has received the data from the 82C55A port.

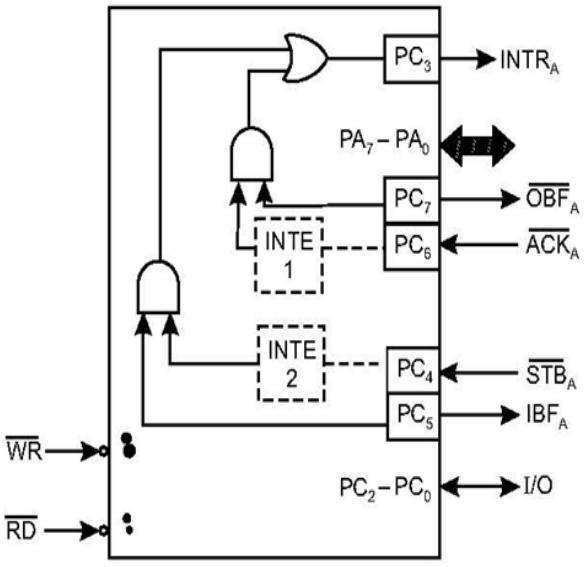
3. INTR (Interrupt request) - It is a signal that often interrupts the microprocessor when the external device receives the data via the signal. This pin is qualified by the internal INTE (interrupt enable) bit.
4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed to enable or disable the INTR pin. The INTE A bit is programmed using the PC6 bit and INTE B is programmed using the PC2 bit.

Mode 2

This functional configuration provides a means for communicating with the peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to mode 1. Interrupt generation and enable/disable function s are also available.

Mode 2 basic functional definitions:

1. Used to group A only
2. 1 8-bit, bidirectional bus port(port A) and a 5-bit control port(port C)
3. Both inputs and outputs are latched
4. The 5-bit control port(port C) is used for control and status for the 8-bit, bidirectional bus port(port A)



2.1AIM: Write and assembly language code for reading switch status of port A and transfer it to port B (to interface 8255 to 8085 in mode 0)

Algorithm:

Step-1: Move control word 90 to accumulator. 90 makes port A as input and port B as output in mode 0

Step-2: Output the accumulator data to control word of 8255 (C6)

Step-3: Input data from port A (C0) to accumulator

Step-4: Output data in accumulator to port B (C2)

Step-5: Go to step-3

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.
9. The status on the switches is shown on the LED's, i.e. , the status of port A (input port) is transferred to port B (output port) and is displayed on LEDs.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4200	3E		MVI A, 90	Move 90 to accumulator
4201	90			
4202	D3		OUT C6	Output accumulator content to C6
4203	C6			
4204	DB	L1	IN C0	Input data from C0 to accumulator
4205	C0			
4206	D3		OUT C2	Output accumulator content to C2
4207	C2			
4208	C3		JMP 4204	Jump to 4204(L1)
4209	04			
420A	42			
420B	CF		HLT	Stop the program

Result:

Input: Port A-10100010

Output: Port B-10100010

Program that transfers the data on the port A switches to LEDs of port B executed successfully.

2.2Aim:

To write and assembly language code to interface 8255 to 8085 in mode 1

Algorithm:

Step-1: Move control word 84 to accumulator

Step-2: Output the accumulator data to control word of 8255 (C6)

Step-3: Move the operand to accumulator

Step-4: Output data in accumulator to port B (C2)

Step-5: Stop the program

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of EXAM REGISTER button and SUBSTITUTE (SUB) button, the content of memory location in which result is stored can be examined.
9. The operand given in the program is shown on the LEDs of port B of 8255

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4200	3E		MVI A, 84	Move 84 to accumulator
4201	90			
4202	D3		OUT C6	Output accumulator content to C6
4203	C6			
4204	3E		MVI A, OP	Move operand to accumulator
4205	OP			
4206	D3		OUT C2	Output accumulator content to C2
4207	C2			
4208	C3		HLT	Stop the program

Result:

Input: A-83

Output: Port B-10000011

Program that interfaces 8255 to 8085 in mode 1 executed successfully.

Viva Questions:

1. Expand PPI
2. How many Ports in 8255?
3. Construct control word for making portA as input port in mode1 and portB as output port in mode1.
4. What are modes of operations in 8255?

EX.NO:3 INTERFACING AND PROGRAMMING 8254/8253

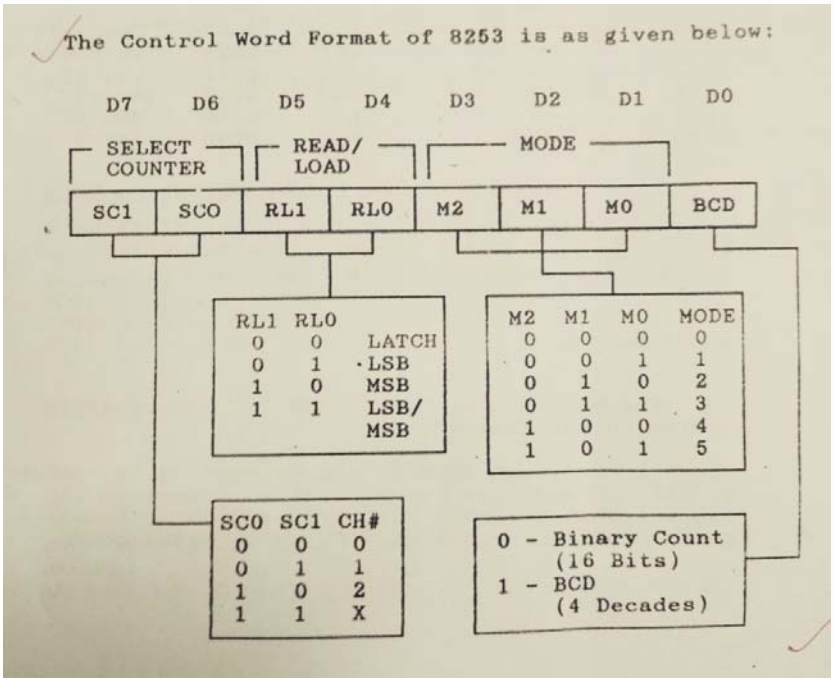
Aim: Program to generate pulse using 8254

Theory: The timer section contains only the chip 8254. The main features of the timer Intel 8254 are as follows:

Three independent 16-bit counters, Input clock from DC to 2 MHz, Programmable counter modes, Count binary or BCD

	A7	A6	A5	A4	A3	A2	A1	A0	HEX
CONTROL REG	1	1	0	0	1	1	1	0	CE
CHANNEL 0	1	1	0	0	1	0	0	0	C8
CHANNEL 1	1	1	0	0	1	0	1	0	CA
CHANNEL 2	1	1	0	0	1	1	0	0	CC

Control Word Format:



Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	3E 30	START:	MVI A ,30	Channel 10
4102	D3 CE		OUT CE	In MODE 0
4104	3E 05		MVI A 05	LSB of Count
4106	D3 C8		OUT 0C8	MSB of count
4108	3E 00		MVI A 00	
410A	D3 C8		OUT C8	
410C	76		HLT	

RESULT: Output square wave is observed in CRO

Viva Questions:

1. How many timer / counters in 8254?
2. Describe mode1 operation of timers.
3. What are registers in 8254?

EX.NO:4 INTERFACING AND PROGRAMMING 8279

4.1 AIM: Program to display message using 8279 interface

APPARATUS: 8085 processor kit, 8279

THEORY:

The Intel 8279 is responsible for de bouncing of the keys, coding of the keyboard matrix and refreshing of the keys, coding of the keypad matrix and refreshing of the display elements in a Microprocessor based development system. Its main features are:

1. Simultaneous Keyboard and Display operation.
2. 3 Input modes such as Scanned Keyboard Mode, Scanned Sensor Mode and Strobed Input Entry Mode.
3. 2 Output modes such as 8 or 16 character multiplexed displays, right entry or left entry display formats.
4. Clock Pre scaler.
5. Programmable Scan Timing.
6. 2 Key lockout or N-Key Roll-over with contact de bounce.
7. Auto Increment facility for easy programming.

I/O Decoding:

I C's 74LS00 (U1) and 74LS138(U2) form the address decoding logic to generate the chip select signal for 8279. Address lines A6 and A7 are NANDed and the NAND gate output is connected to pin 5 of 74LS138 (U2). Similarly, the IOW and IOR are NANDed and the NAND gate output is connected to pin 6 of 74LS138 are connected to address lines A3, A4, A5 respectively.

The 8279 is selected when,

A7	A6	A5	A4	A3	A2	A1	A0
1	1	0	0	0	X	X	X

= C0 (Hex)

Since the address line A1 is connected to A0 of 8279, when,

A7	A6	A5	A4	A3	A2	A1	A0
1	1	0	0	0	X	1	X

= C2 (Hex)

Control / status register is selected, and when,

A7	A6	A5	A4	A3	A2	A1	A0
1	1	0	0	0	X	0	X

= C0 (Hex)

Data register is selected.

Display mode set up:

The command word for keyboard and display mode is,

0	0	0	D	D	K	K	K

DD – DISPLAY MODE:

00 - 8 8-bit Character display – Left Entry

01 - 16 8-bit Character display – Left Entry

10 - 8 8-bit Character display – Right Entry

11 - 16 8-bit Character display – Right Entry

KKK – KEYBOARD MODE:

000 – Encoded Scan Keyboard – 2 key Lockout

001 – Decoded Scan Keyboard – 2 key Lockout

010 – Encoded Scan Keyboard – N key roll-over

011 – Decoded Scan Keyboard – N key roll-over

100 – Encoded Scan Sensor Matrix

101 – Decoded Scan Sensor Matrix

110 – Strobed input, Encoded Display Scan

111 – Strobed input, Decoded Display Scan

CLEAR DISPLAY**:**

The command word format for clear display is,

1	1	0	CD	CD	CD	CF	CA

CD CDCD – The lower two CD bits specify the blanking code to be sent to the segments to turn them off while the 8279 switching from one digit to next.

CD CDCD

0 X - A0-3 B0-3 = 00 = (0000 0000)

0 0 - A0-3 B0-3 = 00 = (0000 0000)

1 0 - A0-3 B0-3 = 20 = (0010 0000)

1 1 - A0-3 B0-3 = FF = (1111 1111)

Enables clear Display when CD=1, the rows of display RAM are cleared by the code set by lower two CD bits. If CD=0 then the contents of RAM will be displayed.

CF – If CF=1, FIFO status is cleared, interrupt output line is reset. Sensor RAM pointer is set to row 0.

CA – Clear All bit has the combined effect of CD and CF. It uses CD clearing code on Display RAM and clears FIFO status. It also resynchronizes the internal timing chain.

WRITE DISPLAY RAM**:**

The write display RAM command word format is,

1	0	0	AI	A	A	A	A

This command is written with A0 (pin21 of 8279) = 1.

All subsequent writes with A0=0 will be to the Display RAM.

AI - Auto Increment Flag. If AI=1, the row address selected will be incremented after each following read or write to the Display RAM.

AAA – Selects one of the 16 rows of the Display RAM.

ACCEPT A KEY AND DISPLAY IT**:**

To accept a key from Hex keyboard and display it in the display.

The initialization of 8279's FIFO RAM and DISPLAY RAM are the same as that given in example 3.

A look up table is provided to obtain the corresponding display codes for the keys pressed.

ALGORITHM:

Step1: Load HL pair with 4500

Step2: Load the count value into register D

Step3: Initialize the character to be displayed from right side

Step4: set the output port to C2

Step5: Move the value CC into register A

Step6: Set the output port to C2

Step7: Move the value of 90 into register A

Step8: Move the data from memory to A

Step9: Set the output port to C0

Step10: Call the delay function

Step11: Increment the HL pair

Step12: Decrement the Register D

Step13: If non zero go to step8

Step14: Go back to Step1

Step15: Move A0 data into register B

Step16: Move FF into register C

Step17: Decrement register C

Step18: If non zero go to step 16.

Step19: Decrement Register B

Step20: If non zero go to step 17

Step21: Return it to the called function.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
4100	21,00,45	START	LXI H,4500	Load HL pair with 4500
4103	16,0F		MVI D,0F	Move 0F into D
4105	3E,10		MVI A,10	Move 10 into A
4107	D3,C2		OUR C2	Out at C2
4109	3E,CC		MVI A,CC	Move CC into A
410B	D3,C2		OUT C2	Out at C2
410D	3E,90		MVI A,90	Move 90 into A
410F	D3,C2		OUT C2	Out at c2

4111	7E	LOOP	MOV A,M	Load data from memory into A
4112	D3,C0		OUT C0	Out at C0
4114	CD,1F,41		CALL DELAY	Call subroutine
4117	23		INX H	Increment HL pair
4118	15		DCR D	Decrement D
4119	C2,11,41		JNZ LOOP	If D not zero go to LOOP
411C	C3,00,41		JMP START	Jump to Start
411F	06,A0	DELAY	MVI B,A0	Move A0 into B
4121	0E,FF	LOOP1	MVI C,FF	Move FF into C
4123	0D	LOOP2	DCR C	Decrement C
4124	C2,23,41		JNZ LOOP2	If C not zero go back to LOOP2
4127	05		DCR B	Decrement B
4128	C2,21,41		JNZ LOOP1	If not zero go back to LOOP1
412B	C9		RET	Return

Seven Segment display code:

Character	HEX value	D	C	B	a	dp	g	f	e
Blank	FF	1	1	1	1	1	1	1	1
H	98	1	0	0	1	1	0	0	0
E	68	0	1	1	0	1	0	0	0
L	7C	0	1	1	1	1	1	0	0
P	C8	1	1	0	0	1	0	0	0
U	1C	0	0	0	1	1	1	0	0
S	29	0	0	1	0	1	0	0	1

RESULT:

Input:

4500: FF FFFFFF

FF FFFFFF

98 68 7C C8

1C 29 FF FF

Output:HELP US

Thus an Assembly Language Program to display a rolling message is successfully completed.

4.2 AIM: Program to display a string of characters using 8086.

APPARATUS: 8086 processor kit, 8279

THEORY:

8279 is a general purpose Programmable Keyboard and Display I/O Interface device designed for use with Intel microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as Hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboards entries are debounced and strobed in an 8- character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16X8 display RAM, which can be organized into dual 16X4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto – increment of the display RAM address.

DESCRIPTION OF THE CIRCUIT:

The 8279-study card provides a keyboard as well as a display section. The display section features six 8 – digit seven segment displays, while the keyboard section comprises a 4 X 4 matrix Hex Keypad and associated circuitry. The option for using shift and control keys during key scanning is also provided. The interface has 3 connectors. P1, J3 and J4. To interface the card with ESA 86/88E trainer, connect the 50 – pin FRC connector P1 to J2 of ESA 86/88E study card adaptor.

For Encode method:

place the jumpers as follows:
JP1 = 1 2
JP2 = 2 3
JP3 = 2 3
JP4 = 1 2
JP5 = 2 3

For Decode method:

place the jumpers as follows:
JP1 = OPEN
JP2 = 1 2
JP3 = 1 2
JP4 = 2 3
JP5 = 1 2

COMMAND WORDS OF 8279:

1	0	0	A_I	A	A	A	A

The CPU seta up the 8279 for a write to display RAM by first writing this command. After writing the command with $A_0 = 1$, all subsequent writes with $A_0 = 0$ will be to the display RAM. The addressing and Auto – increment functions are identical to those for the Read Display RAM.

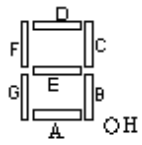
PROGRAM:

PROGRM STARTS AT 2000(ORIGIN) MEMORY LOCATION.

LABEL	MNEMONICS		COMMENTS
	MOV	CX,0006	
	MOV	AL,00	
	MOV	DX,0FFC2	; Routine to clear
	OUT	DX,AL	;all displays
	MOV	AL,90	
	OUT	DX,AL	
RPT:	MOV	AL,00	
	MOV	DX,0FFC0	
	OUT	DX,AL	
	LOOP	RPT	
	MOV	DX,0FFC2	; Eight 8-bit character
	MOV	AL,90	; display left entry
	OUT	DX,AL	;decoded scan keybd
	MOV	SI,2100	
	MOV	CX,0006	
BC0:	MOV	AL,[SI]	; Write to 8279
	MOV	DX,0FFC0	; display RAM
	OUT	DX,AL	;Routine to display
	INC	SI	; a string of characters
	LOOP	BC0	
	INT	03	**;** Termination of the program

The segments of Seven Segment display are connected as shown in below

SEVEN SEGMENT DISPLAY



Viva Questions:

1. Discuss about FIFO RAM.
2. How many display modes in 8279?
3. Describe sensor matrix mode
4. How many display devices can be connected to 8279?

EX.NO:5 A/D AND D/A CONVERTER INTERFACE

D/A Converter interface

APPARATUS:

Oscilloscope, DAC kit, 8085 and 8086processor kits

THEORY:

Digital-to-analog Conversion or simply DAC, is a device that is used to convert a digital (usually binary) code into an analog signal (current, voltage, or electric charge). Digital-to-analog conversion is the primary means by which digital equipment such as computer-based systems are able to translate digital data into real-world signals that are more understandable to or useable by humans, such as music, speech, pictures, video.

The digital-to-analog converters can be broadly classified into three categories, and they are

- Current output
- Voltage output
- Multiplying type

Current output DAC: The current output DAC provides an analog current as output signal.

Voltage output DAC: The analog current signal is internally converted to voltage signal. The voltage output DAC is slower than the current output DAC because of the delay in converting the current signals into voltage signals.

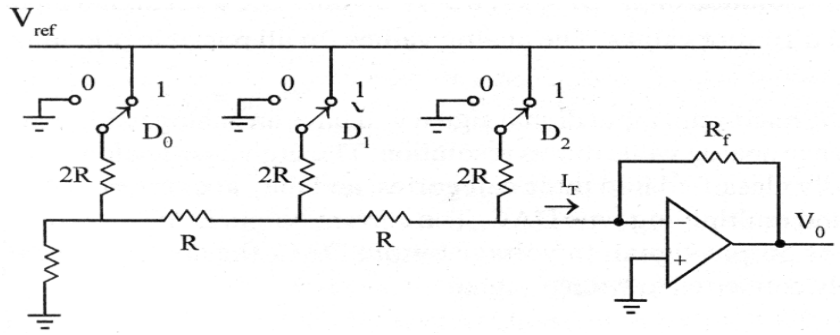
Multiplying type DAC: In multiplying type DAC, the output is given by the product of the input signal and the reference source and the product is linear over a broad range. Basically, there is not much difference between these three types and any DAC can be viewed as multiplying DAC.

Data in binary digital form can be converted to corresponding analog form by using an R-2R ladder network and a summing amplifier.

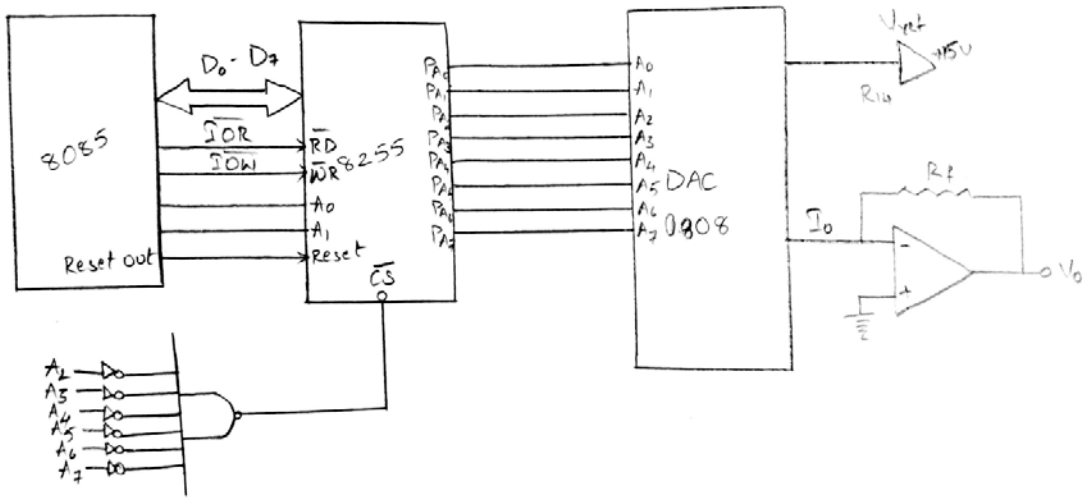
Working of R-2R ladder network DAC:

R-2R weighted resistor ladder network uses only 2 sets of resistors - R and 2R. To build a precise DAC, choose value of resistors that will exactly match the R-2R ratio. V_{ref} is the input binary value reference voltage, i.e., for 1, V_{ref} =5V and for 0, V_{ref} =0V. For example, consider 4-bit input, 0001. Here, only $D_0 = V_{ref}$ and all the other inputs are at 0V and can be treated as ground. The output voltage can be obtained by Thevenin equivalent reduction as

$$V_{out} = -\frac{R_f}{R_i} V_{ref}\left[\frac{D_0}{16} + \frac{D_1}{8} + \frac{D_2}{4} + \frac{D_3}{2}\right]$$



Interfacing DAC to 8085 using 8255:



- Interface 8255 to 8085

- The DAC 0808 is 8-bit digital to analog convertor IC. It converts digital data into equivalent analog current.Therefore I to V converter is used to convert analog output current of DAC to equivalent analog voltage.

- PA0-PA7 pins of Port A are connected to D0-D7 pins of DAC.

- DAC dual power supply of +/- 15V is applied with reference voltage 15V as shown in diagram.

- According to theory of DAC Equivalent analog output is given as: $V_0=V_{ref}$

5.1 AIM:

To write an assembly language code to generate a square wave using 8085

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move FF to accumulator

Step-3: Output it to port A of 8255(40)

Step-4: Create some delay

Step-5: Complement accumulator content

Step-6: If Jump to step-3

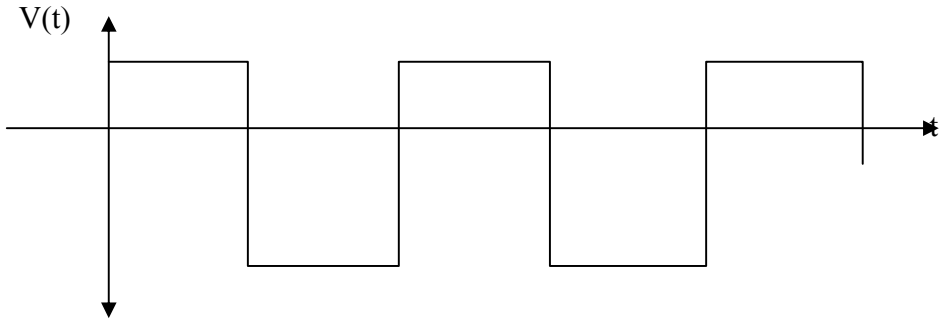
.**PROCEDURE:**

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect the X port to positive, circuitry to negative of oscilloscope probs and connect to channel 1
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
9000	3E		MVI A, 80	Move 80 to accumulator
9001	80			
9002	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
9003	43			
9004	3E		MVI A, FF	Move FF to accumulator
9005	FF			
9006	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
9007	40			
9008	0E		MVI C, FF	Move FF to C
9009	FF			
900A	0D	L1	DCR C	Decrement C
900B	C2		JNZ 900A	If zero flag is not set, go to L1(900A)
900C	0A			
900D	90			
900E	2F		CMA	Complement accumulator content
900F	C3		JMP 9006	Go to L2(9006)
9010	06			
9011	90			

Result:



Amplitude: 2.2V

Time period: 2.4mS

Program that generates a square wave executed successfully.

5.2 AIM:

To write an assembly language code to generate a saw tooth wave for 8085 processor

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 00 to accumulator

Step-3: Output it to port A of 8255(40)

Step-4: Increment accumulator content by 1 and compare accumulator content with FF

Step-5: If zero flag does not set, go to step-3.

Step-6: Jump to step-2

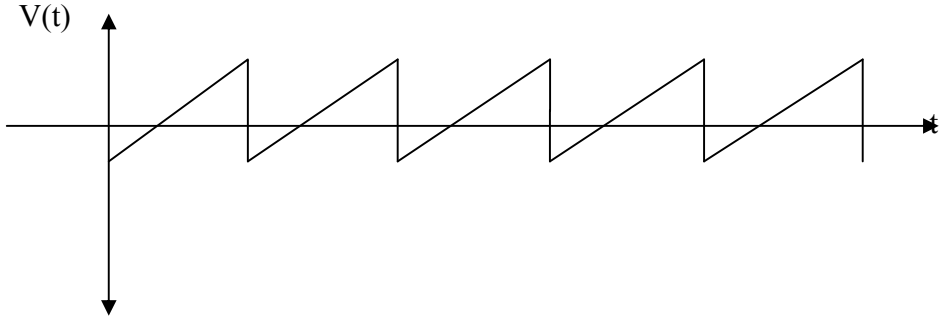
Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect the X port to positive, circuitry to negative of oscilloscope probs and connect to channel 1
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
9000	3E		MVI A, 80	Move 80 to accumulator
9001	80			
9002	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
9003	43			
9004	3E	L1	MVI A, 00	Move 00 to accumulator
9005	00			
9006	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
9007	40			
9008	3C		INR A	Increment accumulator content
9009	FE		CPI FF	Compare accumulator content with FF
900A	FF			
900B	C2		JNZ 9006	If zero flag is not set, go to L2(9006)
900C	06			
900D	90			
900E	C3		JMP 9004	Go to L1(9004)
900F	04			
9010	90			

RESULT:



Amplitude: 2.4V

Time period: 2.6mS

Program that generates a saw tooth wave executed successfully.

5.3AIM：

To write an assembly language code to generate a triangular wave using 8085

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 00 to accumulator

Step-3: Output it to port A of 8255(40)

Step-4: Increment accumulator content by 1 and compare accumulator content with FF

Step-5: If zero flag does not set, go to step-3.

Step-6: Output it to port A of 8255(40)

Step-7: Decrement accumulator content by 1 and compare accumulator content with 00

Step-8: If zero flag does not set, go to step-6.

Step-9: Jump to step-2

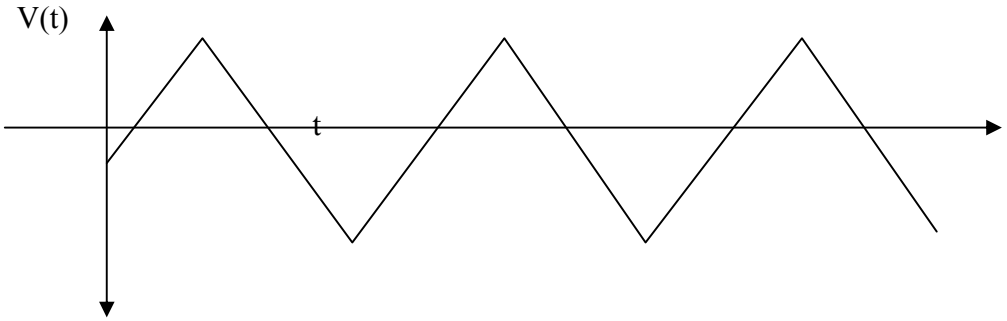
PROCUDRE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect the X port to positive, circuitry to negative of oscilloscope probs and connect to channel 1
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
9000	3E		MVI A, 80	Move 80 to accumulator
9001	80			
9002	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
9003	43			
9004	3E	L1	MVI A, 00	Move 00 to accumulator
9005	00			
9006	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
9007	40			
9008	3C		INR A	Increment accumulator content
9009	FE		CPI FF	Compare accumulator content with FF
900A	FF			
900B	C2		JNZ 9006	If zero flag is not set, go to L2(9006)
900C	06			
900D	90			
900E	D3	L3	OUT 40	Output accumulator content to 40 (port A of 8255)
900F	40			
9010	3D		DCR A	Decrement accumulator content
9011	FE		CPI 00	Compare accumulator content with 00
9012	00			
9013	C2		JNZ 900E	If zero flag is not set, go to L3(900E)
9014	0E			
9015	90			
9016	C3		JMP 9004	Go to L1(9004)
9017	04			
9018	90			

RESULT:



Amplitude: 2.01V

Time period: 4mS

Program that generates a triangular wave executed successfully.

5.4AIM:

To write an assembly language code for generating 2 waves simultaneously (square and triangular wave) with 8085.

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 00 to accumulator

Step-3: Output data in accumulator to port A of 8255(40)

Step-4: Output data in accumulator to port B of 8255(41)

Step-5: Increment accumulator content by 1 and compare accumulator content with FF

Step-6: If zero flag does not set, go to step-4.

Step-7: Output data in accumulator to port A of 8255(40)

Step-8: Output data in accumulator to port B of 8255(41)

Step-9: Decrement accumulator content by 1 and compare accumulator content with 00

Step-10: If zero flag does not set, go to step-8.

Step-11: Jump to step-2

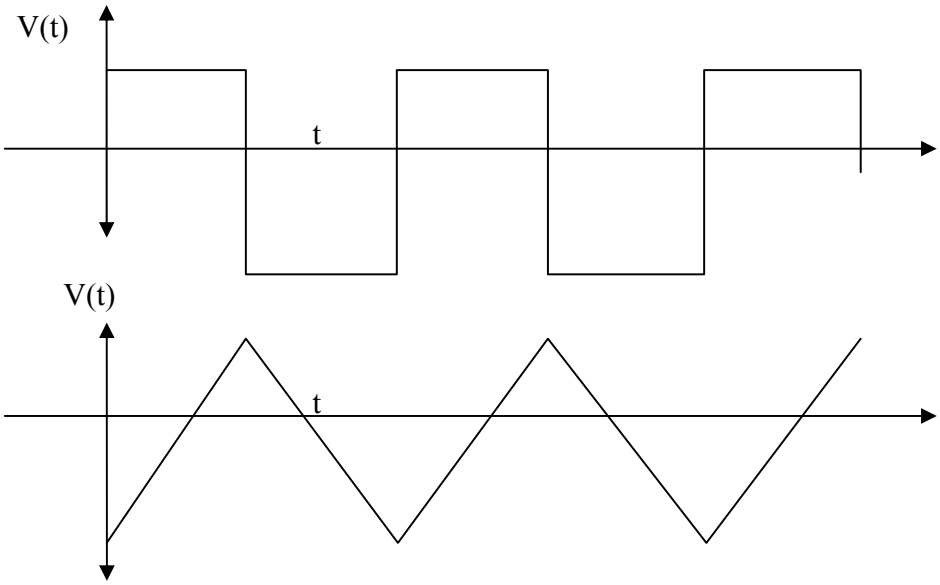
Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect the X port to positive, circuitry to negative of oscilloscope probs and connect to channel 1. Connect Y port to positive, circuitry to negative of oscilloscope probs and connect to channel 2 of oscilloscope.
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
9000	3E		MVI A, 80	Move 80 to accumulator
9001	80			
9002	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
9003	43			
9004	3E		MVI A, 00	Move 00 to accumulator
9005	00			
9006	D3	L1	OUT 40	Output accumulator content to 40 (port A of 8255)
9007	40			
9008	D3	L2	OUT 41	Output accumulator content to 41 (port B of 8255)
9009	41			
900A	3C		INR A	Increment accumulator content
900B	FE		CPI FF	Compare accumulator content with FF
900C	FF			
900D	C2		JNZ 9008	If zero flag is not set, go to L2(9008)
900E	08			
900F	90			
9010	D3		OUT 40	Output accumulator content to 40 (port A of 8255)
9011	40			
9012	D3	L3	OUT 41	Output accumulator content to 41 (port B of 8255)
9013	41			
9014	3D		DCR A	Decrement accumulator content
9015	FE		CPI 00	Compare accumulator content with 00
9016	00			
9017	C2		JNZ 9012	If zero flag is not set, go to L3(9012)
9018	12			
9019	90			
901A	C3		JMP 9006	Go to L1(9006)
901B	06			
901C	90			

RESULT:



Program that generates square and triangular wave simultaneously executed successfully.

5.5 AIM: Program to get sine wave in port-A using DAC using 8086 processor

Generating a sine wave:

To generate a sine wave, we first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees. The values for the sine function varies from –1.0 to +1.0 for 0 to 360 degree angles. Full – scale output of the DAC is achieved when all the data inputs of the DAC are high. So to achieve the full scale, we use the following equation

$$V_{out} = 5V + (5 \text{ X } \sin \theta)$$

Angle vs. voltage Magnitude for Sine Wave.

Angle θ (Degrees)	Sin θ	Vout (voltage Magnitude) 5V + (5V X sin θ)	Values sent to DAC(decimal) (voltage mag. X 25.6)	Equivalent Hex values
0	0.00	5.00	128	80H
30	0.50	7.50	192	C0H
60	0.866	9.33	238	EEH
90	1.00	10.00	255	FFH
120	0.866	9.33	238	EEH
150	0.50	7.50	192	C0H
180	0.00	5.00	128	80H
210	-0.50	2.50	64	40H
240	-0.866	0.669	17	11H
270	-1.00	0.00	0	0H
300	-0.866	0.669	17	11H
330	-0.50	2.50	64	40H
360	0.00	5.0	128	80H

ALGORITHM:

1. Load AL register with Control Word Register contents.
2. Initialize Control Word Register address in to DX and out the contents of AL into Control Word Register.
3. Make SI as pointer to memory to provide the input values and get counter value in CL register.
4. Initialize DX register with port-A and out the AL contents to the port address mentioned in DX.
5. Repeat the loop until all values got outputted through the port A address.
6. Repeat the above procedure to get continuous waveform

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,80	; load AL with Control Word Register contents i.e., 80 Mode 0 operation, port-A as output port.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address

AGAIN:	MOV	SI,2000	; initialize SI to memory to provide the inputs
	MOV	CL,0D	; get counter in CL register
START:	MOV	DX,0FFE0	; initialize DX register with port-A address
	MOV	AL,[SI]	; get the first hex value(equivalent to Angle) to AL from SI
	OUT	DX,AL	; out the contents of AL through port-A address
	INC	SI	; increments the memory of SI to get next value
	LOOP	START	; repeat the loop until all values got outputted through port A address
	JMP	AGAIN	; jumps to the specified label to repeat the steps

5.6 AIM: Program to get triangular wave in port-A using DAC with 8086 processor

ALGORITHM:

1. Load AL register with Control Word Register contents.
2. Initialize Control Word Register address into DX and out the contents of AL through the Control Word Register address.
3. Load initial value i.e., 00 to AL.
4. Initialize DX register with port-A and out the AL contents to the port A address.
5. Increment the AL contents and output through the port A address.
6. Compare AL with FF if it is not equal continue step-5, else continue step – 7.
7. Load AL with FF and output through the port A address.
8. Decrement the AL contents and output through the port A address.
9. Compare AL with 00 and check for equal condition if not equal repeat step-8.
10. Repeat from step-3.

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,80	; load AL with Control Word Register contents ie., 80 Mode 0 operation, port-A as output port.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
START:	MOV	AL,00	; load AL with initial value of triangular wave
BACK:	MOV	DX,0FFE0	; initialize DX register with port-A address
	OUT	DX,AL	; out the contents of AL through port-A address
	INC	AL	; increments AL contents to get next digital value
	CMP	AL, FF	; compare AL with FF
	JNZ	BACK	; check for Zero condition, if no zero jump to the specified label
	MOV	AL,0FF	; load AL with FF
BACK 1:	MOV	DX, 0FFE0	; initialize DX register with port A address
	OUT	DX, AL	; out the contents of AL through the port A address
	DEC	AL	; decrement AL contents
	CMP	AL, 00	; compare AL with 00
	JNZ	BACK 1	; if no zero jump to the specified label
	JMP	START	; jump to the specified label to obtain continuous wave form

5.7AIM: Program to get square wave in port-A using DAC with 8086 processor.

ALGORITHM:

1. Load AL register with Control Word Register contents.
2. Initialize Control Word Register address in to DX and out the contents of AL into Control Word Register.
3. Load initial value i.e., 00 to AL.
4. Initialize DX register with port-A and out the AL contents to the port address mentioned in DX.
5. Provide a delay.
6. Complement the contents of accumulator and repeat the step 4.

PROGRAM:

LABEL	**MNEMONICS**		**COMMENTS**
	MOV	AL,80	; load AL with Control Word Register contents ie., 80 Mode 0 operation, port-A as output port.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
	MOV	AL,00	; load AL with initial value of square wave
START:	MOV	DX,0FFE0	; initialize DX register with port-A address
	OUT	DX,AL	; out the contents of AL through port-A address
	MOV	BX,00FF	; load BX with some count to provide delay
BACK:	DEC	BX	; decrements the contents of BX
	JNZ	BACK	; check for Zero condition
	NOT	AL	; complement contents of AL
	JMP	START	; jumps to the specified label to repeat the steps

5.8 AIM: Program to get saw tooth wave in port-A using DAC with 8086 processor.

ALGORITHM:

1. Load AL register with Control Word Register contents.
2. Initialize Control Word Register address into DX and out the contents of AL through the Control Word Register address.
3. Load initial value i.e., 00 to AL.
4. Initialize DX register with port-A and out the AL contents to the port address.
5. Increment the AL contents and output through the port A address.
6. Unconditionally jump to step 5

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,80	; load AL with Control Word Register contents ie., 80 Mode 0 operation, port-A as output port.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
START:	MOV	AL,00	; load AL with initial value of triangular wave
BACK:	MOV	DX,0FFE0	; initialize DX register with port-A address
	OUT	DX,AL	; out the contents of AL through port-A address
	INC	AL	; increments AL contents to get next digital value
	JMP	BACK	; unconditional jump to the specified label to obtain the continuous wave

ADC INTERFACE :

APPARATUS:

ADC kit, 8085 and 8086 processor kit

THEORY:

Analog to digital converter(ADC) is a quantizing process whereby an analog signal is represented by equivalent binary states. This conversion is basically of 2 types based on the conversion techniques.

One technique involves comparing the given analog signal with the internally generated equivalent signal. This group includes successive approximations, counter and flash type converters. These are used in data loggers and instrumentation applications.

The second technique involves changing an analog signal into time or frequency domain and comparing these new parameters to known values. This group includes integrator converters and voltage to frequency converters. These are used in digital meters, panel meters and monitoring systems where the convertion accuracy is critical.

The generally used ADC is successive approximation type. This includes 3 major elements: the DAC, successive approximation register(SAR) and the comparator.

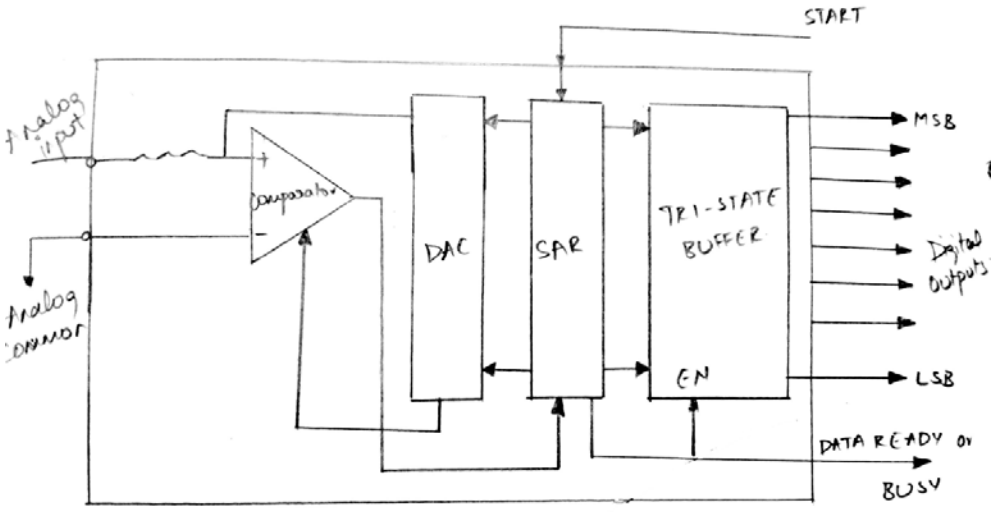
The convertion technique involves comparing the output of the DAC V0 with the analog input signal Vin. The digital input to the DAC is generated using the succesive approximation method. When the DAC output matches the analog signal, the input to the DAC is equivalent digital signal.

The successive approximation method of generating input to the DAC is similar to weighing an unknown material on the chemical balance (less than 1 gram) with a set of such fractional weights as ½ gms, ¼ gms, 1/8 gms etc. The weighing procedure begins with the heaviest weight (1/2 gms) and subsequent weights (in descending order) are added until the balance is tripped. The weight that trips the balance is removed and the process is continued until the smallest weight is used. In case of a 4-bit ADC, bit D3 is turned on first and the output of the DAC is compared with an analog signal. If the comparator changes its state, indicated that the output generated by D3 is larger than the analog signal, bit D3 is turned off in the SAR and bit D2 isturned on. The process continues until the input reaches bit D0. When bit D3 is turned on, the output exceeds the analog signaland therefore, bit D3 is turned off. When the next three successive bits are turned on, the output becomes approximately equal to the analog signal.

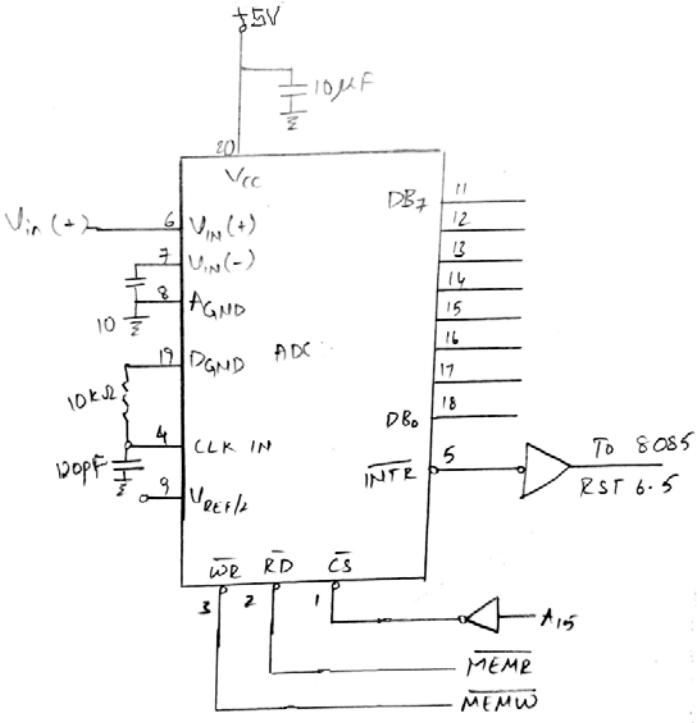
The successive approximation comparision process can be accomplished through either the

software or hardware approach. In the software approach, an ADC is designed using a DAC and the microprocessor plays the role of the counter and SAR. For the hardware approach, a complete ADC, including a tri-state buffer, is now available as an integrated circuit on a chip.

Block diagram of ADC:



Pin diagram of ADC:



5.9 AIM:

To convert given voltage into equivalent digital values with 8085

ALGORITHM:

Step-1: Load CWR of 8255 with 90

Step-2: Load port C of 8255(42) with 00

Step-3: Create some delay

Step-4: Load port C of 8255 (42) with 80

Step-5: Create some delay

Step-6: Input data from port C (42) to accumulator and compare with 80

Step-7: If zero flag does not set, go to step-6. Otherwise, load port C of 8255(42) with 40

Step-8: Input data from port A(40) to accumulator and stop executing the program

PROCEDURE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of exam register button and SUB button, the content of memory in which result is stored can be examined.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8100	3E		MVI A, 90	Move 90 to accumulator
8101	90			
8102	D3		OUT 43	Output accumulator data to port 43
8103	43			
8104	3E		MVI A, 00	Move 00 to accumulator
8105	00			
8106	D3		OUT 42	Output accumulator data to port 42
8107	42			
8108	CD		CALL 8500	Call subroutine at 8500
8109	00			
810A	85			
810B	3E		MVI A, 80	Move 80 to accumulator
810C	80			
810D	D3		OUT 42	Output data in accumulator to port 42
810E	42			
810F	CD		CALL 8500	Call subroutine at 8500
8110	00			
8111	85			
8112	DB	L1	IN 42	Input data from port 42 to accumulator
8113	42			
8114	FE		CPI 80	Compare accumulator data with 80
8115	80			
8116	C2		JNZ 8112	If zero flag is not set, go to L1(8112)
8117	12			
8118	81			
8119	3E		MVI A, 40	Move 40 to accumulator
811A	40			
811B	D3		OUT 42	Output data in accumulator data to port 42

811C	42			
811D	DB		IN 40	Input data from port 40 to accumulator
811E	40			
811F	EF		RST5	Stop the execution of the program
8500	0E		MVI C, FF	Move FF to C
8501	FF			
8502	06	L2	MVI B, FF	Move FF to B
8503	FF			
8504	05	L3	DCR B	Decrement B content
8505	C2		JNZ 8504	If zero flag is not set, go to L3(8504)
8506	04			
8507	85			
8508	0D		DCR C	Decrement C
8509	C2		JNZ 8502	If zero flag is not set, go to L2(8502)
850A	02			
850B	85			
850C	C9		RET	Return to the main program

RESULT:

Voltage given	**Digital value**
1	32
2	64
3	96
4	CA
4.74	FD

Program that converts the given analog value of voltage into equivalent digital value executed successfully.

5.10 AIM: Write a program for converting given analog value in to digital with 8086 processor.

ALGORITHM:

5. Load control word 8B in to control register of 8255
6. Select the channel
7. Send SOC pulse to PA5
8. Check EOC signal
9. If EOC activated read data from port B

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,8B	Load 8B in to control wordregister of 8255.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
	MOV	AL,00	; load AL with channel number 00
	MOV	DX,0FFE0	; initialize DX register with port-A address
	OUT	DX,AL	Selects the channel with port A lines
	MOV	AL,20	generates a pulse at PA5 pin (SOC pulse is given)
	OUT	DX,AL	generates a pulse at PA5 pin (SOC pulse is given)
	XCHG	AX,AX	generates a pulse at PA5 pin (SOC pulse is given)
	XCHG	AX,AX	generates a pulse at PA5 pin (SOC pulse is given)
	XCHG	AX,AX	generates a pulse at PA5 pin (SOC pulse is given)
	MOV	AL,00	generates a pulse at PA5 pin (SOC pulse is given)
	OUT	DX,AL	generates a pulse at PA5 pin (SOC pulse is given)
	MOV	DX,0FFE4	Load port C address in to DX
L1	IN	AL, DX	Read port C value
	AND	AL , 01	Test EOC status with PC0
	JE	L1	If EOC not activated repeat reading port C

	MOV	AL,40	Activate read buffer of ADC with PA6
	MOV	DX,0FFE0	
	OUT	DX,AL	
	MOV	DX,0FFE2	Read digital value through port B
	IN	AL, DX	
	INT	03	End of the program

RESULT:

Voltage given	**Digital value**
0	00
5	FF

Program that converts the given analog value of voltage into equivalent digital value executed successfully

Viva Questions:

1. D/A Converter connected to 8086/8085 as _____. Justify.
2. In A/D conversion what is the role of SOC pin?

EX.NO:6 STEPPER MOTOR INTERFACE

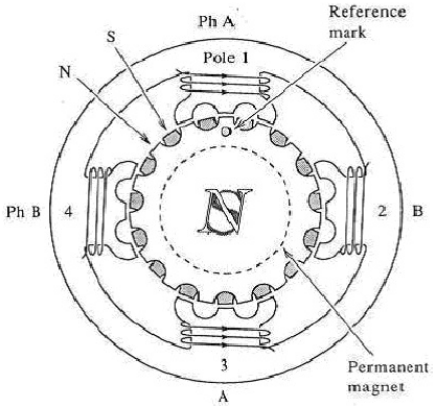
APPARATUS:

Stepper motor interface, 8085 processor kit,8086 processor kit

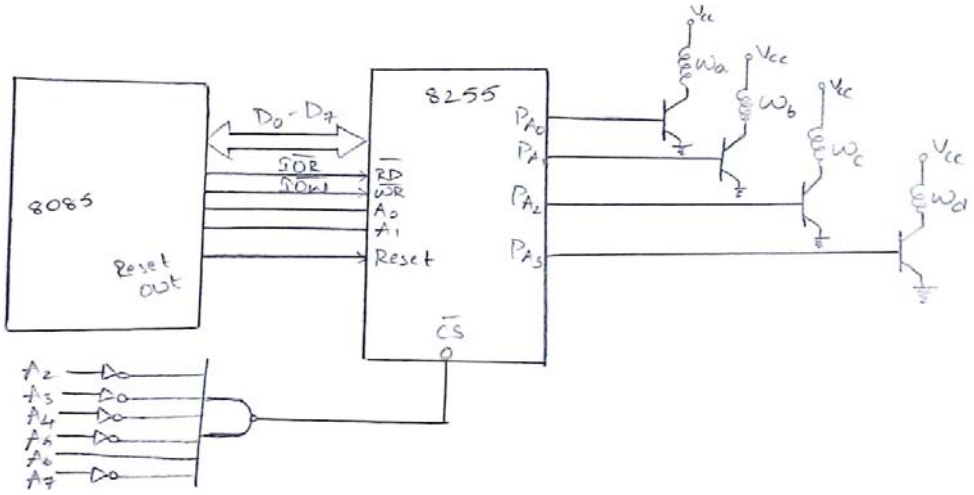
THEORY:

A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation.

Stepper motors operate differently from DC brush motors, which rotate when voltage is applied to their terminals. Stepper motors, on the other hand, effectively have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energized by an external control circuit, such as a microcontroller. To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. So, when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one, and from there the process is repeated. Each of those slight rotations is called a "step," with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle.



Interfacing stepper motor to 8085 using 8255:



6.1AIM:

To write an assembly language code to rotate stepper motor clockwise continuously using 8085

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 88 to accumulator

Step-3: Output accumulator content to port A of 8255(40)

Step-4: Create some delay

Step-5: Rotate accumulator content right

Step-6: If Jump to step-3

PROCEDURE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES 0button.
8. Connect stepper motor kit to 8085 processor kit.
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8300	3E		MVI A, 80	Move 80 to accumulator
8301	80			
8302	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
8303	43			
8304	3E		MVI A, 88	Move 88 to accumulator
8305	88			
8306	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
8307	40			
8308	0E		MVI C, 0F	Move 0F to C
8309	0F			
830A	06	L1	MVI B, FF	Move FF to B
830B	FF			
830C	05	L3	DCR B	Decrement B
830D	C2		JNZ 830C	If zero flag is not set, go to L3(830C)
830E	0C			
830F	83			
8310	0D		DCR C	Decrement C
8311	C2		JNZ 830A	If zero flag is not set, go to L1(830A)
8312	0A			
8313	83			
8314	1F		RAR	Rotate accumulator content right
8315	C3		JMP 8306	Go to L2(8306)
8316	06			
8317	83			

RESULT:

Program that rotates stepper motor clockwise continuously executed successfully.

6.2AIM: Write an assembly language code to rotate stepper motor anti clockwise continuously with 8085 processor

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 88 to accumulator

Step-3: Output accumulator content to port A of 8255(40)

Step-4: Create some delay

Step-5: Rotate accumulator content left

Step-6: If Jump to step-3

PROCEDURE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect stepper motor kit to 8085 processor kit.
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8300	3E		MVI A, 80	Move 80 to accumulator
8301	80			
8302	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
8303	43			
8304	3E		MVI A, 88	Move 88 to accumulator
8305	88			
8306	D3	L2	OUT 40	Output accumulator content to 40 (port A of

				8255)
8307	40			
8308	0E		MVI C, 0F	Move 0F to C
8309	0F			
830A	06	L1	MVI B, FF	Move FF to B
830B	FF			
830C	05	L3	DCR B	Decrement B
830D	C2		JNZ 830C	If zero flag is not set, go to L3(830C)
830E	0C			
830F	83			
8310	0D		DCR C	Decrement C
8311	C2		JNZ 830A	If zero flag is not set, go to L1(830A)
8312	0A			
8313	83			
8314	17		RAL	Rotate accumulator content left
8315	C3		JMP 8306	Go to L2(8306)
8316	06			
8317	83			

RESULT:

Program that rotates stepper motor anti-clockwise continuously executed successfully.

6.3AIM:

write an assembly language code to rotate stepper motor clockwise for $180^o/360^o$using 8085

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 11 to accumulator and 64or C8 to register B

Step-3: Output accumulator content to port A of 8255(40)

Step-4: Create some delay

Step-5: Rotate accumulator content right

Step-6: Decrement B register content

Step-7: if zero flag is not set, go to step-3. Otherwise stop the execution of the program

PROCEDURE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect stepper motor kit to 8085 processor kit.
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8300	3E		MVI A, 80	Move 80 to accumulator
8301	80			
8302	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
8303	43			
8304	06		MVI B, 64 or C8	Move 64 or C8 to B
8305	64 or C8			
8306	3E		MVI A, 11	Move 11 to accumulator
8307	11			
8308	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
8309	40			
830A	0E		MVI C, 0F	Move 0F to C
830B	0F			
830C	06	L1	MVI B, FF	Move FF to B
830D	FF			
830E	05	L3	DCR B	Decrement B
830F	C2		JNZ 830E	If zero flag is not set, go to L3(830E)
8310	0E			
8311	83			
8312	0D		DCR C	Decrement C
8313	C2		JNZ 830C	If zero flag is not set, go to L1(830C)
8314	0C			
8315	83			
8316	1F		RAR	Rotate accumulator content right
8317	05		DCR B	Decrement B
8318	C2		JNZ 8308	If zero flag is not set, go to L2(8308)
8319	08			
831A	83			
831B	76		HLT	Stop the program

RESULT:

Program that rotates stepper motor clockwise for $180^o/360^o$ executed successfully.

6.4AIM:

To write an assembly language code to rotate stepper motor anti clockwise for $180^o/360^o$

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 11 to accumulator and 64 or C8 to register B

Step-3: Output accumulator content to port A of 8255(40)

Step-4: Create some delay

Step-5: Rotate accumulator content left

Step-6: Decrement B register content

Step-7: if zero flag is not set, go to step-3. Otherwise stop the execution of the program

PROCEDURE:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.
5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. Connect stepper motor kit to 8085 processor kit.
9. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button.

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8300	3E		MVI A, 80	Move 80 to accumulator
8301	80			
8302	D3		OUT 43	Output accumulator content to 43 (CWR of 8255)
8303	43			
8304	06		MVI B, 64 or C8	Move 64 or C8 to B
8305	64 or C8			
8306	3E		MVI A, 11	Move 11 to accumulator
8307	11			
8308	D3	L2	OUT 40	Output accumulator content to 40 (port A of 8255)
8309	40			
830A	0E		MVI C, 0F	Move 0F to C
830B	0F			
830C	06	L1	MVI B, FF	Move FF to B
830D	FF			
830E	05	L3	DCR B	Decrement B
830F	C2		JNZ 830E	If zero flag is not set, go to L3(830E)
8310	0E			
8311	83			
8312	0D		DCR C	Decrement C
8313	C2		JNZ 830C	If zero flag is not set, go to L1(830C)
8314	0C			
8315	83			
8316	17		RAL	Rotate accumulator content left
8317	05		DCR B	Decrement B
8318	C2		JNZ 8308	If zero flag is not set, go to L2(8308)
8319	08			
831A	83			
831B	76		HLT	Stop the program

RESULT:

Program that rotates stepper motor anti clockwise for $180^o/360^o$ executed successfully.

6.5 AIM:

write an assembly language code to rotate stepper motor clockwise continuously using 8086

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255(43)

Step-2: Move 88 to accumulator

Step-3: Output accumulator content to port A of 8255(40)

Step-4: Create some delay

Step-5: Rotate accumulator content left

Step-6: If Jump to step-3

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,80	Load 80 in to control wordregister of 8255.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
	MOV	AL,11	; load AL with data for exciting windings
	MOV	DX,0FFE0	; initialize DX register with port-A address
L2	OUT	DX,AL	Selects the channel with port A lines
	MOV	CX,00 FF	Create delay
L1	DEC	CX	
	JNE	L1	
	ROR	AL,1	Generate data for exciting winding
	JMP	L2	

RESULT:stepper motor rotated continuously.

6.6AIM:

Write an assembly language code to rotate stepper motor anti clockwise for $180^o/360^o$ with 8086 processor.

ALGORITHM:

Step-1: Move 80 to accumulator and output it to CWR of 8255

Step-2: Move 11 to accumulator and 64 or C8 to register B

Step-3: Output accumulator content to port A of 8255

Step-4: Create some delay

Step-5: Rotate accumulator content left

Step-6: Decrement B register content

Step-7: if zero flag is not set, go to step-3. Otherwise stop the execution of the program

PROGRAM:

LABEL	MNEMONICS		COMMENTS
	MOV	AL,80	Load 80 in to control wordregister of 8255.
	MOV	DX, 0FFE6	; initialize DX register with Control Word Register address
	OUT	DX,AL	; out contents of AL through Control Word Register address
	MOV	BX, 64H	Count value for 360 rotation
	MOV	AL,11	; load AL with data for exciting windings
	MOV	DX,0FFE0	; initialize DX register with port-A address
L2	OUT	DX,AL	Selects the channel with port A lines
	MOV	CX,00 FF	Create delay
L1	DEC	CX	
	JNE	L1	
	ROR	AL,1	Generate data for exciting winding
	DEC	BX	
	JNZ	L2	Conditional jump instruction for limiting the excitions
	INT	03	End of the program

RESULT: Stepper rotated 360^0 clock wise

Viva Questions:

1. What is the step angle for stepper motor used in lab.
2. If step angle of stepper motor is 1.8^0, to move stepper motor 360^0 clock wise how many excitations required?

EX.NO:7 DISPLAY INTERFACING

7.1 AIM: Write an ALP to display a repeating message using 7 segment interface with 8085 microprocessor and 8255 PPI.

APPARATUS:

Microprocessor trainer kit, Seven segment display interface and power supply unit, 8085 processor

PROCEDURE:

1. Enter the machine code by pressing NEXT.
2. Execute the program by pressing GO.
3. Give the initial address and press EXE.
4. Check the result in the display.

ALGORITHM:

Step1: Move data 80 into register A

Step2: Set the output port to 43

Step3 : Load HL register with 8100

Step4: Move 04 into register C

Step5: Move 04 into register D

Step6: Move 08 into register B

Step7: Move the data from M into A

Step8: Increment the HL register Pair

Step9: Rotate left through carry

Step10: set the output to 41

Step11: Move data from A to E

Step12: Move 01 to register A

Step13: Set output to 42

Step14: Decrement register A

Step15: Set output port to 42

Step16: Move data from E to A

Step17: Decrement register B

Step18: If non zero goto step6

Step19: Decrement Register D

Step20: If non zero go to step 6

Step21:go to step 25

Step22: Decrement register C

Step23: If non zero go to step 5

Step24:If non zero go to step3

Step25: Push PSW

Step26:Push HL pair

Step27: Load HL pair with FFFF

Step28: Decrement HL register pair

Step29:Perform OR operation, if non zero go to step 28

Step30: Pop HL and PSW ,return

PROGRAM:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8000	8E,80		MVI A,80	Move control word 80 to accumulator
8002	D3,43		OUT 43	out to control word register
8004	21,00,81	LOOP4	LXI H,8100	HL pair made as pointer to seven segment code stored from 8100
8007	0E,04		MVI C,04	Initialize line counter move data 4 into reg C
8009	16,04	LOOP3	MVI D,04	Initialize character counter move the data 4 into reg D
800B	06,08	LOOP2	MVI B,08	Initialize bit counter to display character
800D	7E		MOV A,M	Move data from M to A
800E	23		INX H	Increment HL pair
800F	07	LOOP1	RLC	Rotate accumulator content left by 1 bit
8010	D3,41		OUT 41	Output accumulator content to port A
8012	5F		MOV E,A	move data from A to E
8013	3E,01		MVI A,01	move data 01 to reg A
8015	D3,42		OUT 42	Set out at 42
8017	3D		DCR A	decrement reg A
8018	D3,42		OUT 42	Set out at 42
801A	7B		MOV A,E	Move data from E to A
801B	05		DCR B	Decrement Reg B
801C	C2,0E,80		JNZ LOOP1	if nonzero go to Loop1
801F	15		DCR D	decrement reg D
8020	C2,0B,80		JNZ LOOP2	If nonzero go to Loop2
8023	CD,20,80		CALL DELAY	Calls the subroutine delay
8026	0D		DCR C	Decrement reg C

8027	C2,09,80		JNZ LOOP3	If nonzero go to Loop3
802A	C2,04,80		JMP LOOP4	To repeat the process go to Loop4
802D	E5	DELAY	PUSH PSW	Push the value A and flags to stack
802E	E6		PUSH H	Push the value in HL to stack
802F	21,FF,FF		LXI H,FFFF	Load the count value in to HL pair
8032	2B	LI	DCX H	decrement HL pair by one
8033	7D		MOV A,L	Move data from L to A
8034	B4		ORA H	Perform OR operation between HL
8035	C2,32,80		JNZ LI	if nonzero go to label L1
8038	E1		POP H	Retrieve HL content from stack
8039	F1		POP PSW	Retrieve PSW content from stack
803A	C9		RET	Return to main program

7-SEGMENT DISPLAY CODE:

CHARACTER	HEX VALUE	h	g	f	e	d	c	b	a
BLANK	FF	1	1	1	1	1	1	1	1
A	88	1	0	0	0	1	0	0	0
L	C7	1	1	0	0	0	1	1	1
L	C7	1	1	0	0	0	1	1	1
T	87	1	0	0	0	0	1	1	1
H	89	1	0	0	0	1	0	0	1
E	84	1	0	0	0	0	1	0	0
B	83	1	0	0	0	0	0	1	1
E	84	1	0	0	0	0	1	0	0
S	92	1	0	0	1	0	0	1	0
T	97	1	0	0	1	0	1	1	1

RESULT:

Input: 8100 FF, FF , FF, FF

FF, C7, C7, 88

FF, 84, 89, 87

87, 92, 84, 83

Output: ALL THE BEST

Thus an ALP to display 7- segment repeating message using interface with 8085 microprocessor and 8255 PPI is successfully completed.

Viva Questions:

1. What is the difference between common cathode and common anode display.
2. Construct 7 segment code to display message VANI.

ADDITIONAL PROGRAMS:

EX.NO:8TRAFFIC LIGHT CONTROL INTERFACE TO 8085

Aim:To write an ALP to implement Traffic light control interface on 8085 microprocessor

Apparatus required: 8085 microprocessor kit ,traffic light interface

Algorithm:

Step1:Press RESET

Step2: Loading register A with the immediate data

Step3:Load HL register with 16 bit address

Step4:Move the value in address pointed by M to C

Step5:Increment HL register pair

Step6:Move the data contained by M to A

Step7:Make port A as outport

Step8:Increment HL register pair

Step9:Move the value pointed by M to A

Step10:Make port B as outport

Step11:Increment HL register pair

Step12:Move the data pointed by M to A

Step13:Make portC as output port

Step14:Call the subroutine delay

Step15: Decrement register C

Step16:Jump to location if the result does not yield zero

Procedure:

1. Reset the system using RES button.
2. To enter the program into memory location, press NEXT and enter memory location address of the instruction.
3. Enter hex value of the program.
4. Press NEXT to go to the next memory location.

5. Enter next hex value.
6. Enter hex values into memory locations by pressing NEXT button till the end of the program.
7. Press RES button.
8. For the execution of the program, make use of GO command. Give starting address of the program. Press EXEC button. With the help of exam register button and SUB button, the content of memory in which result is stored can be examined.

Program:

MEMORY LOCATION	HEX CODE	LABEL	MNEMONICS	COMMENTS
8000	3E,80		MVI A,80H	Move 80 control word in to reg A
8002	d3,43		OUT 43	Out control word to control word register
8004	21,00,82	BACK:	LXI H,8200	HL pair made as pointer to 8200
8007	4E		MOV C,M	Move the value from memory to reg C
8008	23	LOOP	INX H	Increment HL pair
8009	7E		MOV A,M	Get the value from memory in toA
800A	D3,40		OUT 40	Out to port A
800C	23		INX H	HL pair point to next memory location
800D	73		MOV A,M	Data from memory location moved in to A
800E	D3,41		OUT 41	Out to port B
8010	23		INX H	Make HL point to next location
8011	7E		MOV A,M	Get data from memory in to A
8012	D3,42		OUT 42	Out to port C
8014	CD,00,90		CALL Delay	Delay subroutine is called
8017	0D		DCRC	Count decremented
8018	C2,07,80		JNZ LOOP	Count not zero repeat the loop
801B	C3,03,80		JMP repeat	Loop repeated continusly
801E	EF		RST5	End of the program
Delay program				
9000	F5		PUSH PSW	Push psw in to stack
9001	06,1E		MVI B,1E	Count for delay
9003	11,FF,FF	BACK2	LXID,FFFFH	Count for delay
9006	1B	BACK1	DCX D	Decrement count in reg D
9007	7B		MOV A,E	Move E to A
9008	B2		ORA D	Check whether DE content zero (or) not
9009	C2,06,90		JNZ BACK1	If not go to BACK 1
900C	05		DCR B	Decrement B reg

900D	C2,03,90		JNZ BACK2	If not zero go to BACK2
9010	F1		POP PSW	Retrieve PSW
9011	C9		RET	Return to main program

Result:

8206:84
8205:84
8204:50
8203:5A
8202:80
8201:80
8200:02