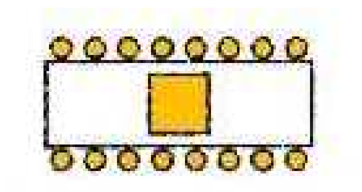
PRACTICAL- MICROPROCESSORS AND ASSEMBLY LANGUAGE PROGRAMMING



Submitted By

Vaibhav Jain
Student
Bachelor of Computer Applications
Swati Jain Institute Of Management Studies.
2001-2004.

Head Office **182, Jaora Compound,** **Indore.** Swati Jain Academy (SJA)	**Institute Campus** **33, Sampat Farms,** **Bicholi Mardana,** **Indore** Swati Jain Institute of Management Studies(SJIMS)

Acknowledgments

No man is born complete and I am no exception. When the times were tensed and it seemed like I should kick the whole bunch of meaningless symbols and code into the recycle bin, pour some water on my keyboard and throw the book away for good, all that could sustain me was the support of teachers, friends and elders. I was lucky enough to be surrounded by such a people who were helpful and supportive. Without their help this Project File would have probably completed on released on my 75th birthday.

In am greatly thankful to Mr. Suaysh Jain, Teacher as well as Friend with out whose dedicated guidance and support this project would have being non existent.

I am also exceedingly thankful to the member faculty at Swati Jain Institute Of Management Studies as well as Swati Jain Madam who were there when their support and that wonderful sense of humor was badly and eagerly needed. Thanks, I can never forget you all.

And, finally a word of gratitude to my Parents and brother Ronak, who were always there with their support and encouragement, even though I'm a little crazy at times.

Vaibhav Jain

vaibhav@genesisconvent.com
57, Shiv Shahkti Nagar,
Kanadia Road,
Indore.
May 2004.

CERTIFICATE

This is to certify that **Vaibhav Jain** an enrollee of Bachelor of Computer Application and a student **Swati Jain Institute of Management Studies** has worked on the project **"Practical in Microprocessors And Assembly Language Programming"**. He has put sincere effort in the project and has performed tasks related to the project in the Computer Lab of **Swati Jain Institute of Management Studies**. This project may be considered as a partial fulfillment for the examinations conducted by **Devi Ahilya Vishva Vidyalaya, Indore**.

Date:

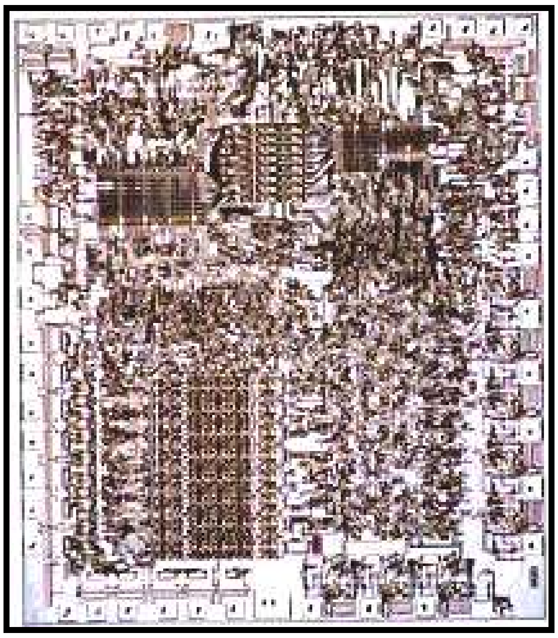
Mr. Suyash Jain

External

Index..

Index.

****



1976: The 8085 Microprocessor

The 8085 Microprocessor

What is 8085-Microprocessor ?

Introduction

The 8085 is an obsolete 8-bit microprocessor which was built by Intel. The famous company which was founded in 1968 by Gordon E. Moore and Robert Noyce.

8085 was a successor of highly successful 8080 microprocessor which powered the legendry Altair systems. The Intel 8080 was an early 8-bit CPU which was released in April 1974 running at 2 MHz, and is generally considered to be the first truly usable microprocessor CPU design. The 8080 formed the basis of many early computers, such as the MITS Altair 8800 and IMSAI 8080, forming the user base for machines running the CP/M operating system.

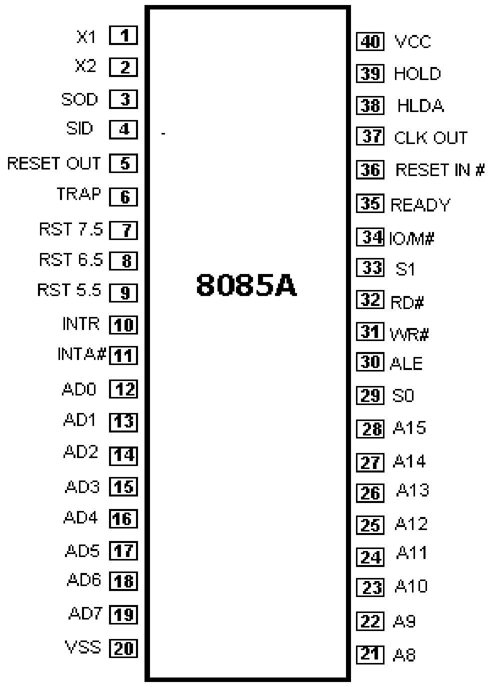
8085 was one step ahead of 8080 as because it required less supporting hardware. It integrated an On Chip Clock Generator and an On Chip System Controller. This led to systems with lesser number if IC`s and supporting hardware . Infact an 8085 based system could be assembled with just 3 IC`s .The five in the name came from the fact that the 8085 required only a 5V power supply rather than the 5V and 12V supplies the Intel 8080. Moreover Its instruction set was completely compatible with Intel-8080.

Below are some of the characteristics of 8085

Intel 8085

Year Introduced	March 1976
Clock Speed	5 MHz
Fabrication Technology	3 – Micron
Transistor count	6,5000
Addressable memory	2^{16} bytes=64 Kilo Bytes
Operating Voltage	+5 volts
Features	♦ 0.8 ms instruction Cycle (VCC = 5V) ♦ On-Chip Clock Generator (with External Crystal) ♦ On-Chip System Controller; Advanced Cycle Status Information Available for Large System Control ♦ Four Vectored interrupt (One is non-maskable) Plus the 8080A-compatible interrupt. ♦ Serial, In/Serial Out Port ♦ Decimal, Binary and Double Precision Arithmetic ♦ Addressing Capability to 64K Bytes of Memory ♦ TTL Compatible ♦ Full Software Compatibility with Intel-8080 ♦ Required Only 5-volt power supply
Applications	♦ Toledo scale. Computed cost from weight and price ♦ Traffic Signal Controllers ♦ Microprocessor based controller devices

Pinout Diagram & Pin Description



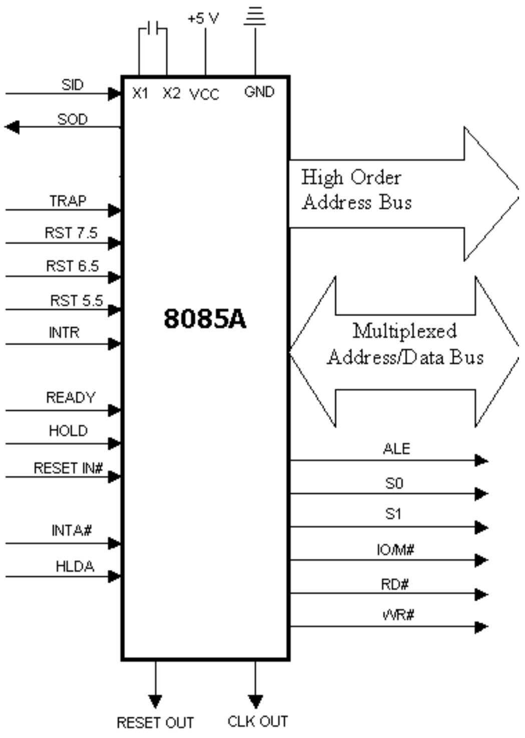


Figure:1 8085- Pinout **Figure:2 Control & Status Signals**

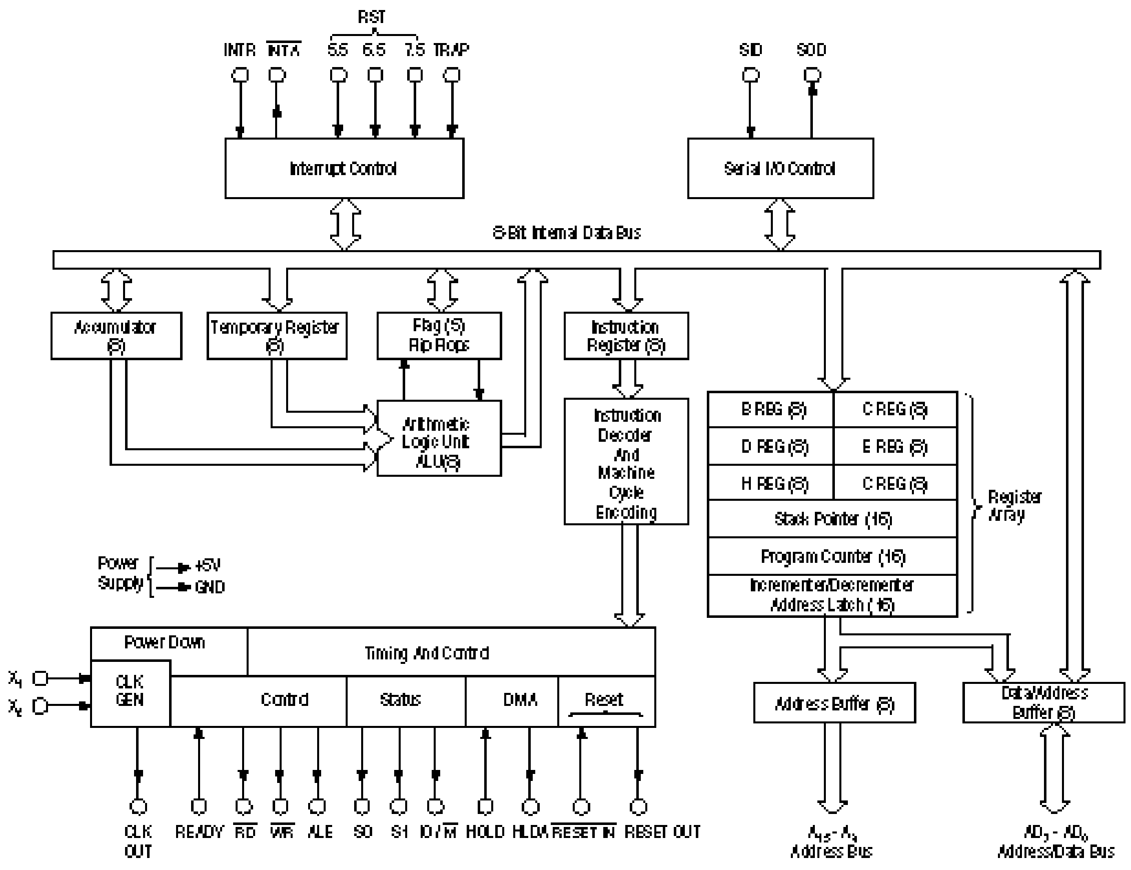
Pin Description

The following Table describes the function of each pin:

Symbol	Function
A6 – A15	Address Bus: The most significant 8-bits of the memory address or the 8-bits of the I/O address, 3-stated during Hold and Halt modes and during RESET.
A0 - A7	Multiplexed Address/Data Bus: Lower 8-bits of the memory address (or I/O address) appear on the bus during the first clock cycle (T state) of a machine cycle. It then becomes the data bus during the second and third clock cycles.
ALE	Address Latch Enable: It occurs during the first clock state of a machine cycle and enables address to get latched into the on-chip latch peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. The falling edge ALE can also be used to strobe the status information ALE is never 3-state.
S0, S1 & (IO/M#)	Machine cycle status: S1 can be used as an advanced R/W status. IO/M#, S0 and S1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.
RD#	READ control: A low level on RD indicates the selected memory or I/O device is to be read that the Data Bus is available for the data transfer, 3-stated during Hold and Halt modes and during RESET.
WR#	WRITE control: A low level on WR indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of WR, 3-stated during Hold and Halt modes and during RESET.

Symbol	Function
READY	If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, the CPU will wait an integral number of clock cycles for READY to go high before completing the read or write cycle READY must conform to specified setup and hold times.
HOLD	HOLD indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3-stated. And status of power down is controlled by HOLD.
HLDA	HOLD ACKNOWLEDGE: Indicates that the cpu has received the HOLD request and that it will relinquish the bus in the next clock cycle. HLDA goes low after the Hold request is removed. The cpu takes the bus one half clock cycle after HLDA goes low.
INTR	INTERRUPT REQUEST: Is used as a general purpose interrupt. It is sampled on during the next to the last clock cycle of an instruction and during Hold and Halt states. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted. Power down mode is reset by INTR.
INTA#	INTERRUPT ACKNOWLEDGE: Is used instead of (and has the same timing as) RD during the instruction cycle after an INTR is accepted.
RST 5.5 RST 6.5 RST 7.5	RESTART INTERRUPTS: These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted. The priority of these interrupts is ordered as shown in on the left . These interrupts have a higher priority than INTR. In addition, they may be individually masked out using the SIM instruction. Power down mode is reset by these interrupts.
TRAP	Trap interrupt is a nonmaskable RESTART interrupt. It is recognized at the same timing as INTR or RST 5.5 - 7.5. It is unaffected by any mask or Interrupt Disable. It has the highest priority of any interrupt. Power down mode is reset by input of TRAP.
RESET IN#	RESET IN: Sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops and release power down mode. The data and address buses and the control lines are 3-stated during RESET and because of the asynchronous nature of RESET IN, the processor's internal registers and flags may be altered by RESET with unpredictable results. RESET IN is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay. The CPU is held in the reset condition as long as RESET IN is applied.
RESET OUT	Indicated CPU is being reset. Can be used as a system reset. The signal is synchronized to the processor clock and lasts an integral number of clock periods.
X1,X2	X1 and X2 are connected to a crystal to drive the internal clock generator. X1 can also be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency.
CLK	Clock Output for use as a system clock. The period of CLK is twice the X1, X2 input period.
SID	Serial input data line. The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
SOD	Serial output data line. The output SOD is set or reset as specified by the SIM instruction.
Vcc	+ 5 Volt supply
GND	Ground Reference.

Functional Description & Internal Architecture



8085 is a complex microprocessor with an equally complex architecture. Like all processors it is also build upon a set of distinct units working and cooperating with each other to perform task assigned to the processor. Internally 8085 consists of a Register Array Accumulator, Control Unit , A Clock Generator , Interrupt Controller and an Instruction Decoder. All these units are connected to each other via a 8 bit internal data bus that carries data between these units, and a control bus that transmits various control signal originating at or meant for the control unit. The 8085 uses a multiplexed Data Bus. The address is spilt between the higher 8-bit Address Bus and the lower 8-bit Address/Data Bus. During the first T state (clock cycle) of a machine cycle the low order address is sent out on the Address/Data Bus. These lower 8-bits may be latched externally by the Address Latch Enable signal (ALE). During the rest of the machine cycle the data bus is used for memory or I/O data. The 8085 provides RD, WR, S0, S1, and IO/M signals for bus control. The external Address Bus is interfaced by an Address and Data Buffer. An Interrupt Acknowledge signal (INTA) is also provided to signal interrupt acceptance to the initiating device. Hold and all Interrupts are synchronized with the processor's internal clock. The 8085 also provides Serial Input Data (SID) and Serial Output Data (SOD) lines for a simple serial interface. In addition to these features, the 8085 has three maskable, vector interrupt pins, one nonmaskable TRAP interrupt and HALT and HOLD.

The Register Array

The 8085 has twelve addressable 8-bit register pairs. Six others can be used interchangeably as 8-bit registers or 16-bit register pairs. The 8085 register set is as follows:

Mnemonic	Register	Contents
ACC or A	Accumulator	8- bit
PC	Program Counter	16- Bit Address
BC, DE,HL	General Purpose Register, Data Pointer (HL)	8 bits x 6 or 16- bits x 3
SP	Stack Pointer	16- bit address
Flags	Flag Register	5 flags (8- bit space)

Register A

This is the Accumulator of 8085. It is the only general purpose register that is connected to the ALU. That is why all arithmetic as well as logical operation is are performed using this register.

Registers B-L

These are the general purpose register provided to store 8-bit or 1-Byte of data or operands. In case 16-bit storage is required than these registers can be paired with the adjacent register to form a 16 bit general purpose register. These register pairs are usually used for performing operations involving two bytes. 8085 allows only following pairs to be made: B-C , H-L , D-E

Stack Pointer

This register forms the basis of stack mechanism of the 8085. It keeps the track of the top of the in memory stack. Whenever data needs to be pushed on stack than it is transferred to the memory location pointed by the SP and after transferring SP is updated to point to the new memory location. In case of Pop instruction first the SP is decremented and the data pointed by it is transferred to the specified register.

Program Counter

This register keeps tracks of the address of next instruction to be fetched and executed. At the end of each machine cycle instruction the PC register is updated with the new address location of instruction and the Opcode from that address. if fetched at the beginning of next instruction cycle.

Flags Register

This is a special purpose register that contains various flip flops to indicate results of arithmetic operations that are executed:

S Flag:	This flag is set when the result of an arithmetic is negative.
Z Flag:	This flag indicates that the result of pervious operation was zero.
Ac Flag:	This flag is used internally to perform binary to BCD conversion.
P Flag:	This is indicates that the parity of the data in accumulator is even
Cy Flag:	This flags indicates overflow or underflow of the result of the operation

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
S	Z	xxx	Ac	xxx	P	xxx	Cy

Flags Register Format

Interrupts & Serial I/O

The 8085 has 5 interrupt inputs: INTR, RST 5.5 RST 6.5, RST 7.5, and TRAP. Each of the three RESTART inputs, 5.5, 6.5, and 7.5, has a programmable mask. TRAP is also a RESTART interrupt but it is nonmaskable.

The three maskable interrupts cause the internal execution of a RST instruction if the interrupts are enabled the interrupt mask is not set. The nonmaskable TRAP causes the internal execution of a RST vector independent of the state of the interrupt enable or masks.

There are two different types of inputs in the restart interrupt. RST 5.5 and RST 6.5 are high level-sensitive like INTR (and INT on the 8080A) and are recognized with the same timing as INTR. RST 7.5 is rising edge-sensitive. For RST 7.5, only a pulse is required to set an internal flip-flop which generates the internal interrupt request. The RST 7.5 request flip-flop remains set until the request is serviced. Then it is reset automatically, This flip-flop may also be reset by using the SIM instruction. The RST 7.5 internal flip-flop will be set by a pulse on the RST 7.5 pin even when the RST 7.5 interrupt is masked out.

Name	Priority	Branch Address	Type Trigger
TRAP	1	24H	Rising edge and High Level until sampled
RST 7.5	2	3CH	Rising Edge
RST 6.5	3	34H	High Level until Sampled
RST 5.5	4	2CH	High Level until Sampled
INTR	5	--	High Level until Sampled

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending, as follows: TRAP-highest priority, RST 7.5, RST 6.5, RST 5.5, INTR-lowest priority. The TRAP interrupt is useful for catastrophic evens such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the highest priority. It is not affected by any flag or mask. The TRAP input is both edge and level sensitive. The TRAP input must go high and remain high until it is acknowledged. It will not be recognized again until it goes low, then high again. This avoids any false triggering due to noise or logic glitches. Note that the servicing of any interrupt (TRAP, RST 7.5, RST 6.5, RST 5.5,INTR) disables all future interrupts (except TRAP) until an EI instruction is executed.

The TRAP interrupt is special in that it disables interrupts, but preserves the previous interrupt enable status. Performing the first RIM instruction following a TRAP interrupt allows you to determine whether interrupts were enabled or disabled prior to the TRAP. All subsequent RIM instructions provide current interrupt enable status. Performing a RIM instruction following INTR or RST 5.5-7.5 will provide current interrupt Enable status, revealing that Interrupts are disabled.

SIM Instruction

The execution of the SIM instruction uses the contents of the accumulator to mask interrupts. The data in the accumulator specifies bit flags for various Interrupt masks as well as for outputting serial data to the devices connected to the SOD Pin. The format of the control word is as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

SIM Instruction Accumulator Data Format

Bit	Description
SOD (Serial Output Data)	This bit specifies the output level of the SOD pin. If set than SOD pin is latched High else it is latched Low
SDE (Serial Data Enable)	This bit is low than the SOD Bit is ignored. On order to output data to the SOD pin this bit needs to be set.
R7.5 (Reset RST 7.5 Flip-flop)	When this bit is set to 1, the edge detecting flip-flop of RST 7.5 interrupt is reset.
MSE (Mask Set Enable)	When this bit is set to 1, the interrupt mask bits are valid.
M7.5 (Mask RST7.5)	When this bit is set to 1 and MSE bit is set to 1, RST7.5 interrupt is masked.
M6.5 (Mask RST6.5)	When this bit is set to 1 and MSE bit is set to 1, RST6.5 interrupt is masked.
M5.5 (Mask RST5.5)	When this bit is set to 1 and MSE bit is set to 1, RST 5.5 interrupt is masked.

RIM Instruction

When the contents of the accumulator are read out after RIM instruction has been executed, 8085 interrupt status can be known.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SID	I7.5	I6.5	I 5.5	IE	M7.5	M6.5	M5.5

RIM Instruction Accumulator Data Format

Bit	Description
SID(Serial Input Data)	his bit reflects the status of SID pin. Is the status at SID is high this bit is set, other this bit is reset.
I7.5 (Pending RST7.5)	When RST7.5 interrupt is pending, "1" is read out.
I6.5 (Pending RST6.5)	When RST6.5 interrupt is pending, "1" is read out.
I5.5 (Pending RST5.5)	When RST5.5 interrupt is pending, "1" is read out.
IE (Interrupt Enable Flag)	When interrupt is Enable, "1" is read out.
M7.5 (Mask RST7.5)	When RST7.5 interrupt is masked, "1" is read out.
M6.5 (Mask RST6.5)	When RST6.5 interrupt is masked, "1" is read out.
M5.5 (Mask RST5.5)	When RST5.5 interrupt is masked ,"1" is read out.

Programming 8085

Data Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Intel 8085 is follows small eadiean format of byte representation i.e. in a byte the LSB lies in higher memory and the MSB lies in lower memory.

Instruction Format

The 8085 supports three type of instruction. One byte instructions & Two bytes instruction and Three Bytes Instructions

One Byte Instruction	**Opcode**		
Two Byte Instruction	**Opcode**	Operand 1	
Three Byte Instruction	**Opcode**	Operand 1	Operand 2

Opcode Format

The Opcode byte of every instruction specifies type of operation and objects on which those operations are to be performed.

Code	Source	Dest

The code bits 7-6 specify the operation type required. The bits 5-3 Specify the source register & bits 2-0 usually identify source of data needed to execute the instruction. Internal Registers are identified as follows:-

Binary Code	Register	Binary Code	Register
000	**B**	110	**M**
001	**C**	111	**A**
010	**D**	00	**B-C**
011	**E**	01	**D-E**
100	**H**	10	**H-L**
101	**L**	11	**SP or PSW**

8085 Instruction Set

DATA TRANSFER INSTRUCTIONS

MOV R1, R2

Move Data From Source Reg. To Destination Reg.

This instruction copies the contents of the source register R2 into the destination register R1.The contents of source register are not altered.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	1	D	D	D	S	S	S		
Example	MOV A,B									

MOV R, M

Move Data From Memory To Destination Reg.

This instruction copies the contents of the memory location pointer to by H-L pair to the register R . The contents of memory is not altered.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	1	S	S	S	1	1	0		
Example	MOV B,M									

MOV M, R2

Move Data From Source Reg. To Memory

This instruction copies the contents of the source register R2 into the destination register R1.The contents of source register are not altered.

Bytes	1	**T States**	7	**Machine Cycles**	F W					
Flags Effected	S		Z		AC		P		CY	
Opcode Format	0	1	1	1	0	S	S	S		
Example	MOV M,B									

MVI R, 8-bit

Move Immediate Operand to Destination Reg.

This instruction copies 8-bit data byte into the destination register R.

Bytes	2	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	S	S	S	1	1	0		
Example	MVI A,74h									

MVI M, 8-bit

Move Immediate Operand to Memory

This instruction copies the 8-bit data byte into the memory location pointed to by H-L pair destination register R.

Bytes	2	**T States**	10	**Machine Cycles**	F R W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	1	0	1	1	0		
Example	MVI M,20h									

LXI RP,16-bit

Load Immediate Word in Destination Reg. Pair

This instruction 16-bit Word into the destination Register Pair. The first data byte is taken as the LSB and the second byte is takes as the MSB.

Bytes	3	**T States**	10	**Machine Cycles**	F R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	D	D	0	0	0	0		
Example	LXI SP, FFFFh									

LDAX RP

Load Accumulator Indirect

This instruction copies the contents of the memory location pointed to by Reg. Pair B-C or Reg. Pair D-E into the accumulator.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	D	D	0	0	1	0		
Example	LDAB B									

LHLD adr

Load H-L Direct

Load the H-L Pair with contents of memory location whose address is specified as immediate operand.

Bytes	3	**T States**	16	**Machine Cycles**	F R R R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	0	0	0	1	0		
Example	LHLD 2000h									

LDA adr

Load Accumulator Direct

Loads the contents of memory location whose address is given as the immediate operand into the Accumulator

Bytes	3	**T States**	13	**Machine Cycles**	F R R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	1	0	0	1	0		
Example	LDA 2000h									

STA adr

Store Accumulator Direct

Stores the contents Accumulator into the memory location whose address is given as the immediate operand.

Bytes	3	**T States**	13	**Machine Cycles**	F R R W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	1	1	1	1	0	1	0		
Example	STA 2000h									

STAX RP

Store Accumulator Indirect

This instruction copies the contents of the accumulator to the memory location pointed to by Reg. Pair B-C or Reg. Pair D-E.

Bytes	1	**T States**	7	**Machine Cycles**	F W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	D	D	1	0	1	0		
Example	STAX D									

SHLD adr

Store H-L Direct

Stores the contents of H-L Pair into the memory location whose address is specified as immediate operand.

Bytes	3	**T States**	16	**Machine Cycles**	F R R W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	0	1	0	1	0		
Example	SHLD 2000h									

XCHG

Exchange D-E and H-L

Exchanges the contents of Reg. D with Reg. H and Reg. E with Reg. L.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	1	0	1	0	1	1		
Example	XCHG									

ARITHMETIC AND LOGICAL INSTRUCTION

ADD R

Add To Accumulator

Add the contents of specified register to the contents of the Accumulator and result is stored in the Accumulator

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	0	S	S	S		
Example	ADD B									

ADD M

Add Memory To Accumulator

This instruction add the byte pointed to by H-L pair to the Accumulator. The result of addition is stored in the Accumulator

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	0	1	1	0		
Example	ADD M									

ADI 8-bit

Add To Accumulator, Immediate

Add the contents of immediate operand to the contents of the Accumulator and result is stored in the Accumulator

Bytes	2	**T States**		4	**Machine Cycles**			F		
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	1	0	0	0	1	1	0		
Example	ADI 20h									

ADC R

Add With Carry

This instruction adds specified Reg. And the carry bit to the Accumulator. The result of addition is stored in the Accumulator.

Bytes	1	**T States**		4	**Machine Cycles**			F		
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	1	S	S	S		
Example	ADC C									

ADC M

Add With Carry

This instruction adds the carry bit and the memory location pointed to by H-L Pair to the Accumulator and store the result in the Accumulator.

Bytes	1	**T States**		7	**Machine Cycles**			F R		
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	1	1	1	0		
Example	ADC M									

ACI 8-bit

Add With Carry Immediate

This instruction adds the carry bit and the immediate operand to the Accumulator and store the result in the Accumulator.

Bytes	2	**T States**		7	**Machine Cycles**			F R		
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	1	0	0	1	1	1	0		
Example	ACI 20h									

SUB R

Subtract with Accumulator

Subtract the contents of specified register from the contents of the Accumulator and result is stored in the Accumulator

Bytes	1	**T States**		4	**Machine Cycles**			F		
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	1	0	S	S	S		
Example	SUB B									

SUB M

Subtract with Memory

This instruction subtract the byte pointed to by H-L pair to the Accumulator. The result of addition is stored in the Accumulator

Bytes	1	**T States**		7	**Machine Cycles**		F R			
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	1	0	1	1	0		
Example	SUB M									

SUI 8-bit

Subtract Immediate with Accumulator

Subtracts the contents of immediate operand to the contents of the Accumulator and result is stored in the Accumulator

Bytes	2	**T States**		7	**Machine Cycles**		F R			
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	1	0	0	0	1	1	0		
Example	SUI B									

SBB R

Subtract With Borrow

This instruction subtracts specified Reg. And the carry bit from the contents of Accumulator. The result of addition is stored in the Accumulator.

Bytes	1	**T States**		4	**Machine Cycles**		F			
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	1	S	S	S		
Example	SBB C									

SBI 8-bit

Subtract Immediate and Borrow

This instruction subtracts the carry bit and immediate operand to the Accumulator and store the result in the Accumulator.

Bytes	2	**T States**		7	**Machine Cycles**		F R			
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	0	0	1	1	1	0		
Example	SBI 20h									

SBB M

Subtract Memory and Borrow

This instruction subtract the carry bit and the memory location pointed to by H-L from the Accumulator and stores result in Accumulator..

Bytes	1	**T States**		7	**Machine Cycles**		F R			
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	1	0	0	1	1	1	0		
Example	SBB M									

INR R

Increment Register

This instruction increments the contents of the specified register by one.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	
Opcode Format	0	0	S	S	S	1	0	0		
Example	INR L									

INR M

Increment Memory

This instruction increments the contents of the memory location pointed by H-L pair by one.

Bytes	1	**T States**	10	**Machine Cycles**	F R W					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	
Opcode Format	0	0	1	1	0	1	0	0		
Example	INR M									

DCR R

Decrement Register

This instruction decrements the contents of the specified register by one.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	
Opcode Format	0	0	S	S	S	1	0	1		
Example	DCR A									

DCR M

Decrement Memory

This instruction decrements the contents of the memory location pointed by H-L pair by one.

Bytes	1	**T States**	10	**Machine Cycles**	F R W					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	
Opcode Format	0	0	1	1	0	1	0	1		
Example	DCR M									

INX RP

Increment Register Pair

This instruction increments the specified register pair with one.

Bytes	1	**T States**	6	**Machine Cycles**	S					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	S	S	0	0	1	1		
Example	INX H									

DCX RP

Decrement Register Pair

This instruction decrements the specified register pair with one.

Bytes	1	**T States**	6	**Machine Cycles**	S					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	S	S	S	1	0	0		
Example	DCX H									

ANA R

And Accumulator with Register

This instruction performs Bitwise AND of Accumulator with specified Reg. And stores result in Accumulator

Bytes	1	**T States**	7	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	1	P	•	CY	0
Opcode Format	1	0	1	0	0	S	S	S		
Example	ANA B									

ANA M

And Accumulator with Memory

Instructs 8085 to perform logical AND operation of Accumulator with that of memory pointer to by H-L pair.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	1	P	•	CY	0
Opcode Format	1	0	1	0	0	1	1	0		
Example	ANA M									

ANI 8-bit

And Accumulator with Immediate

Instructs 8085 to perform logical AND operation of Accumulator with the immediate operand and to store the result back in Accumulator

Bytes	2	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	1	P	•	CY	0
Opcode Format	1	1	1	0	0	1	1	0		
Example	ANI 08h									

XRA R

XOR Accumulator with Register

This instruction performs Bitwise XOR of Accumulator with specified Reg. And stores result in Accumulator

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	0	1	0	1	S	S	S		
Example	XRA B									

XRA M

XOR Accumulator with Memory

Instructs 8085 to perform logical XOR operation of Accumulator with that of memory pointer to by H-L pair.

Bytes	1	**T States**		7	**Machine Cycles**			F R		
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	0	1	0	1	S	S	S		
Example	XRA M									

XRI 8-bit

XOR Accumulator with Immediate

Instructs 8085 to perform logical XOR operation of Accumulator with the immediate operand and to store the result back in Accumulator

Bytes	2	**T States**		7	**Machine Cycles**			F		
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	1	1	0	1	1	1	0		
Example	XRI 90h									

ORA R

OR Accumulator with Register

This instruction performs Bitwise OR of Accumulator with specified Reg. And stores result in Accumulator

Bytes	1	**T States**		7	**Machine Cycles**			F R		
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	0	1	1	0	S	S	S		
Example	ORA A									

ORA M

OR Accumulator with Memory

Instructs 8085 to perform logical OR operation of Accumulator with that of memory pointer to by H-L pair.

Bytes	1	**T States**		7	**Machine Cycles**			F R		
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	0	1	1	0	S	S	S		
Example	ORA M									

ORI 8-bit

OR Accumulator with Immediate

Instructs 8085 to perform logical OR operation of Accumulator with the immediate operand and to store the result back in Accumulator

Bytes	2	**T States**		4	**Machine Cycles**			F		
Flags Effected	S	•	Z	•	Ac	0	P	•	CY	0
Opcode Format	1	1	1	0	1	1	1	0		
Example	ORI 03h									

CMP R

Compare Accumulator with Register

Subtract the contents of destination register from the Accumulator but the result is discarded and only flags are changed.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	1	1	1	S	S	S		
Example	CMP B									

CMP M

Compare Accumulator with Memory

Compares the contents of accumulator with contents of memory location pointed to by H-L Pair.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	0	1	1	1	1	1	0		
Example	CMP M									

CPI 8-bit

Compare Accumulator with Immediate

Instructs 8085 to compare the contents of Accumulator with the immediate operand.

Bytes	1	**T States**	7	**Machine Cycles**	F R					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	1	1	1	1	1	1	1	0		
Example	CPI 08h									

DAD RP

Double Add To H-L Pair

Add the specified Reg. Pair to the H-L Pair and the result is stored in the H-L Pair.

Bytes	1	**T States**	10	**Machine Cycles**	F B B					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	S	S	1	0	0	1		
Example	DAD H									

RAL

Rotate Accumulator Left Through Carry

Transfers each bit of Accumulator to its adjacent higher bit. The LSB is filled with contents of Carry Flag and MSB transferred to Carry flag.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	0	0	0	1	0	1	1	1		
Example	RAL									

RAR

Rotate Accumulator Right Through Carry

Transfers each bit of Accumulator to its adjacent lower bit. The MSB is filled with contents of Carry Flag and LSB transferred to Carry flag.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	•
Opcode Format	0	0	0	1	1	1	1	1		
Example	RAR									

RLC

Rotate Accumulator Left with Carry

Transfers each bit of Accumulator to its adjacent higher bit. The LSB is filled with contents of MSB and MSB copied to Carry flag.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	•
Opcode Format	0	0	0	0	0	1	1	1		
Example	RLC									

RRC

Rotate Accumulator Right with Carry

Transfers each bit of Accumulator to its adjacent lower bit. The MSB is filled with contents of LSB and LSB copied to Carry flag.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	•
Opcode Format	0	0	0	0	1	1	1	1		
Example	RRC									

BRANCH CONTROL INSTRUCTIONS

JMP adr

Jump to Address

Transfers the contents of immediate operand to the PC register and the next instruction is fetched from that memory location.

Bytes	3	**T States**	12	**Machine Cycles**	S R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	0	0	0	0	1	1		
Example	JMP 2000h									

J[con] adr.

Conditional Jump to Immediate Address

This instruction tests the flag specified as opcode suffix and if the specified condition is met than the jump is performed.

Bytes	3	**T States**	12	**Machine Cycles**	S R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	C	C	C	0	1	0		
Example	JPO 200h									

CALL adr

Call Subroutine

Pushes the contents of PC register into the stack and transfers the immediate operand to the PC register.

Bytes	3	**T States**	18	**Machine Cycles**	S R R W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	0	0	1	1	0	1		
Example	CALL 2000h									

C[con] adr.

Conditional Call to Subroutine

This instruction tests the flag specified as opcode suffix and if the specified condition is met than the call is performed.

Bytes	3	**T States**	18	**Machine Cycles**	S R R W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	C	C	C	1	0	0		
Example	CPO 200h									

RET

Return from Subroutine

Pops the stack in the PC register into the stack and the next operand is fetched from the new location.

Bytes	1	**T States**	12	**Machine Cycles**	S R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	0	0	1	0	0	1		
Example	RET									

R[con]

Conditional Call to Subroutine

This instruction tests the flag specified as opcode suffix and if the specified condition is met than the return operation is performed.

Bytes	1	**T States**	12	**Machine Cycles**	S R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	C	C	C	0	0	0		
Example	RM									

PCHL

Transfer To H-L

Performs an Unconditional Jump to the address pointed to by H-L Pair.

Bytes	1	**T States**	6	**Machine Cycles**	S					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	1	0	1	0	0	1		
Example	PCHL									

I/O AND MACHINE CONTROL INSTRUCTIONS

PUSH RP — Push to Stack

Transfers the contents specified pair to the top of the stack and the SP is decremented by two. SP cannot be specified as the operand

Bytes	1	**T States**	10	**Machine Cycles**	F W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	1	0	1		
Example	PUSH B									

POP RP — Pop from Stack

Transfers the contents memory location pointed to by SP to the specified pair and SP is incremented by two. SP cannot be specified as the operand

Bytes	1	**T States**	10	**Machine Cycles**	F R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	POP C									

PUSH PSW — Push Program Status to Stack

Pushes contents Accumulator and Flag Reg.`s into the stack.

Bytes	1	**T States**	10	**Machine Cycles**	F W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	1	1	0	1	0	1		
Example	PUSH PSW									

POP PSW — Pop from Stack

Pops the contents of the stack into the Accumulator and the Flag registers.

Bytes	1	**T States**	10	**Machine Cycles**	F R R					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	POP PSW									

XTHL — Exchange Stack Top with H-L

Exchanges the contents of the top of the stack with the contents of the H-L pair.

Bytes	1	**T States**	16	**Machine Cycles**	F R R W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	XTHL									

SPHL

Transfer H-L To SP

Transfers the contents of Reg. Pair H-L to SP.

Bytes	1	**T States**	6	**Machine Cycles**	S					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	1	1	1	0	0	1		
Example	SPHL									

OUT 8-bit

Output To Port

Transfers the contents of Accumulator to the port specified as immediate operand.

Bytes	2	**T States**	10	**Machine Cycles**	F R O					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	OUT 20h									

IN 8-bit

Input From Port

Receives a data byte from specified port and copies it to the Accumulator.

Bytes	2	**T States**	10	**Machine Cycles**	F R I					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	0	1	1	0	1	1		
Example	IN 40h									

DI

Disable Interrupts

Disables all the interrupts except the TRAP interrupt.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	DI									

EI

Enable Interrupt

Re-enables the interrupts and reinforces the previous interrupt mask specified.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	S	S	0	0	0	1		
Example	EI									

NOP

No Operation

Inserts a Wait State in current Machine Cycle.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	0	0	0	0	0	0		
Example	NOP									

HLT

HALT

Instructs 8085 to enter into halted state where it waits until it is interrupted externally.

Bytes	1	**T States**	5	**Machine Cycles**	F B					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	1	1	1	0	1	1	0		
Example	HLT									

RIM

Read Interrupt Mask

Reads the current interrupt mask and stores it into Accumulator.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	0	0	0	0	0		
Example	RIM									

SIM

Set Interrupt Mask

Sets the interrupt mask to that specified by the Accumulator.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	1	0	0	0	0		
Example	SIM									

RST [n]

Restart At N

Performs a software Reset such that the control is transferred to location determined as 8*n.

Bytes	1	**T States**	12	**Machine Cycles**	S W W					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	1	1	N	N	N	1	1	1		
Example	RST 2									

DAA

Decimal Adjust Accumulator

Changes the contents of the Accumulator to two 4-bit BCD digits. Sets Auxiliary Flag is there is a carry from lower digit to higher.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S	•	Z	•	Ac	•	P	•	CY	•
Opcode Format	0	0	1	0	0	1	1	1		
Example	DAA									

CMA

Complement Accumulator

Calculates One` s Complement of the contents of the Accumulator and stores it in the Accumulator.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	
Opcode Format	0	0	1	0	1	1	1	1		
Example	CMA									

STC

Set Carry Flag

Sets the Carry Flag in the Flags Register.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	1
Opcode Format	0	0	1	1	0	1	1	1		
Example	STC									

CMC

Complement Carry Flag

Inverts the status of Carry Flag in the Flags Register.

Bytes	1	**T States**	4	**Machine Cycles**	F					
Flags Effected	S		Z		Ac		P		CY	•
Opcode Format	0	0	1	1	1	1	1	1		
Example	CMC									

****

Machine Cycles

F	Fetch with 4 T-States
S	Fetch with 6 T-States
R	Read With 3 T-States
W	Write with 3 T-States
I	I/O Read with 3 T-States
O	I/O Write with 3 T-States
B	Bus Ideal With 3 T-States

Codes (S S S) or (D D D)
DDD: Destination Register
SSS: Source Register

B	000	A	111
C	001	BC	00
D	010	DE	01
E	011	HL	10
H	100	SP	11
L	101		

Conditional Codes (C C C)

NZ	Zero =0	000
Z	Zero =1	001
NC	Carry =0	010
C	Carry =1	011
PO	Parity =0	100
PE	Parity =1	101
P	Sign =0	110
M	Sign =1	111

Honey We Made It…

Programming With 8085

16-Bit Register Left Rotation

Objective

To implement routines to perform 16-bit rotation to the left and right of H-L Register. Takes input as a 16-bit number in Register Pair H-L and number of times the rotation is to be performed in register C.

Statistics

Code Size	20 Bytes
T-State Count	108
Execution Time	54 x 10^{-6} sec @ 2.0 MHz

Code [16bitrotation.asm]

```
0000    210100 lxi h,1           ;load data in HL
0003    0E22    mvi c,22h        ;rotate to left 34
                                 ;times
0005    0C      inr c
                loop:
0006    0D      dcr c
0007    CA1900          jz exit   ; exit if count is zero
000A    7D      mov a,l ; copy l to acc
000B    37      1stc    ;
000C    3F      cmc     ;set carry flag to zero
000D    17      ral     ;rotate acc left 1-bit
000E    6F      mov l,a ;copy back to l
000F    7C      mov a,h;now work on contents of h
0010    17      ral     ;rotate through cary
0011    67      mov h,a;copy contents back to h
0012    3E00            mvi a,0
0014    8D      adc l   ;add l with cary in acc
0015    6F      mov l,a ;mov this to l
0016    C30600          jmp loop;cary on the loop
0019    76      exit: hlt
```

Input

A		B		D		H	**12**
F		C	**01**	E		L	**34**

S		Z		A		P		C	

Output

A	**68**	B		D		H	**24**
F	**44**	C	**00**	E		L	**68**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

16-Bit Register Right Rotation

Objective

To implement routines to perform 16-bit rotation to the right of H-L Register. Takes input as a 16-bit number in Register Pair H-L and number of times the rotation is to be performed in register C.

Statistics

Code Size	27
T-State Count	112
Execution Time	56 x 10^{-6}sec @ 2.0 MHz

Program [16bitrotationr.asm]

```
0000 213412    lxi h,1234h      ;load data in HL
0003 0E01      mvi c,1h;rotate to right 34 times
;*******************************************
;********************************************
0005 0C        inr c
               loop:
0006 0D        dcr c
0007 CA1A00    jz exit   ; exit if count is zero
000A 7C        mov a,h; copy l to acc
000B 37        stc;
000C 3F        cmc     ;set carry flag to zero
000D 1F        rar              ;rotate acc left 1-bit
000E 67        mov h,a;copy back to l
000F 7D        mov a,l ;now work on contents of h
0010 1F        rar              ;rotate through cary
0011 6F        mov l,a ;copy contents back to h
0012 3E00      mvi a,0
0014 1F        rar     ;rotate acc and send cary bit
                                  ;to the MSB
0015 B4        ora h            ;or l with cary in acc
0016 67        mov h,a          ;mov this to l
0017 C30600    jmp loop         ;cary on the loop

001A 76        exit: hlt
;*******************************************
;********************************************
```

Input

A		B		D		H	**12**
F		C	**01**	E		L	**34**

S		Z		A		P		S	

Output

A	**09**	B		D		H	**09**
F	**44**	C	00	E		L	**1A**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Simulation of Integer Addition

Objective

To implement a program to simulate Integer Addition on 8085 using only the logical bitwise and shift innstructions. This program was writtern to implmented this algorithm. The algorithm utilizes recursion to perform the task. So a recursice proccecure in also implemented.

Statistics

Code Size	21 Bytes
T-State Count	127
Execution Time	63 x 10^{-6}sec @ 2.0 MHz

Program [vaiadd.asm]

```
;Addtion Without Add Instruction
;******************************
;implements a new Addition Algorithm
;discovered by Vaibhav Jain
;(c) Vaibhav Jain
;**************************
 0000   0610    mvi b,10h         ;add operand1
 0002   0EEE    mvi c,eeh         ;add operand2

 0004   CD0800 call AddBC ;addtion result is in B
 0007   76         hlt        ; result is B=8B

                AddBC:
 0008   79      mov a,c;copy c to acc
 0009   B7      ora a     ;is c=zero
 000A   C8      rz        ;is zero than return

 000B   A0      ana b     ;bit wise And to b
 000C   07      rlc                 ;rotate it to left
 000D    57     mov d,a;save result in d
 000E   79      mov a,c;copy C to a
 000F   A8      xra b     ;bitwise Xor it to B
 0010   47      mov b,a;save result in B
 0011   4A      mov c,d;copy And+Rot result to c
 0012   CD0800call AddBC        ;call it recursively
 0015   C9      ret
```

Input

A		B	**10**	D		H	
F		C	**EE**	E		L	

S		Z		A		P		S	

Output

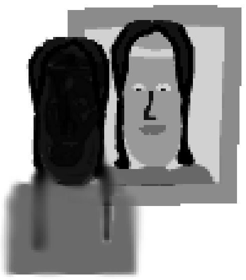
A	**00**	B	**FE**	D	**00**	H	**00**
F	**44**	C	**00**	E	**00**	L	**00**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Reversal of an Array

Objective

To implement a program to reverse a given array. Input is taken as length of array in Register C and address of array in H-L pair.



Statistics

Code Size	17 Bytes
T-State Count	636
Execution Time	318 x 10^{-6}sec @ 2.0 MHz

Program [arrreverse.asm]

```
;program to reverse a given array
;input h-l address of array
;c: count of no. of elements
;*******************************
0000 21D007    lxi h,2000
0003 0E08      mvi c,8

0005 54        mov d,h
0006 5D        mov e,l
0007 41        mov b,c
0008 7E        loop1: mov a,m
0009 F5               push psw
000A 23               inx h
000B 0D               dcr c
000C C20800           jnz loop1


000F F1        loop2: pop psw
0010 12               stax d
0011 13               inx d
0012 05               dcr b
0013 C20F00           jnz loop2
0016 76        hlt
;******************************************
;******************************************
```

Input

A		B		D		H	20
F		C	16	E		L	00

S		Z		A		P		S	

Memory

2000	01	02	03	04	05
2005	06	07	08	09	0A
200A	0B	0C	0D	0E	0F
200F	10	11	12	13	14
2014	15	16	17	18	19
2019	1A	1B	1C	1D	1E
201E	00	00	00	00	00

Output

A	00	B	0	D	20	H	20
F	44	C	00	E	16	L	16

S	0	Z	1	A	0	P	1	S	0

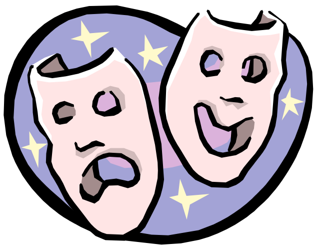
Memory

2000	15	14	13	12	11
2005	10	0F	0E	0D	0C
200A	0B	0A	09	08	07
200F	06	05	04	03	02
2014	01	16	17	18	19
2019	1A	1B	1C	1D	1E
201E	00	00	00	00	00

Maximum & Minimum of an Array

Objective

To implement a program to reverse a find the maximum and the minimum elements of a given array. Input is taken as length of array in Register C and address of array in H-L pair. The monimum element found is stored in register B and minimum element in register D.



Statistics

Code Size	33 Bytes
T-State Count	1149
Execution Time	574 x 10^{-6}sec @ 2.0 MHz

Program [arrmaxmin.asm]

```
;get max & min elements of the given array

0000 21D007     lxi h,2000          ; address of array
0003 0E0F       mvi c,F             ; no. of elements
0005 AF         xra a
0006 B1         ora c
0007 CA2100     jz exit
000A 7E         mov a,m
000B 46         mov b,m
000C 23         loop:    inx h
000D 0D                             dcr c
000E CA2100                         jz exit
0011 BE                             cmp m
0012 D21600                         jnc jmp2
0015 7E                             mov a,m

0016 57         jmp2:               mov d,a
0017 78                             mov a,b
0018 BE                             cmp m
0019 DA1D00                         jc loopout
001C 46                             mov b,m

001D 7A         loopout:mov a,d
001E C30C00                         jmp loop
0021 76         exit:               hlt
;*******************************************
;*******************************************
```

Input

A		B		D		H	**20**
F		C	**0F**	E		L	**00**

S		Z		A		P		S	

Memory

2000	0F	0E	0D	0C	0B
2005	0A	09	08	07	06
200A	05	04	03	02	01
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**0F**	B	**01**	D	**0F**	H	**20**
F	**44**	C	**00**	E	**00**	L	**0F**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

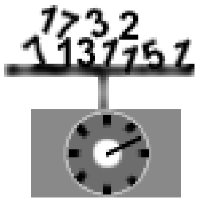
Memory

2000	0F	0E	0D	0C	0B
2005	0A	09	08	07	06
200A	05	04	03	02	01
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Addition of Two Arrays

Objective

To implement a program to add two bounded arrays element by element into a third bounded array. Input is given as address of first source array in H-L, second source array in D-E and the destination bounded array in B-C.



Statistics

Code Size	47 Bytes
T-State Count	551
Execution Time	275x 10^{-6}sec @ 2.0 MHz

Program [addarrays.asm]

```
;adding two arrays element by element into an
;another array
0000 21D007    lxi h,2000;source bounded array1
0003 11B80B    lxi d,3000;source bounded array2
0006 01A00F    lxi b,4000;destination bounded array
0009 E5        push h  ;save address of arr1
000A 62        mov h,d;
000B 6B        mov l,e  ;load address of arr2 in h-l
000C 50        mov d,b;
000D 59        mov e,c;load add of dest. arr in d-e
000E 7E        mov a,m           ;load count of arr2
000F E3        xthl              ;m->arr1
0010 46        mov b,m           ;load count of arr1
0011 BE        cmp m             ;
0012 DA1600    jc condition1
0015 78        mov a,b
0016 4F condition1:      mov c,a;load the counter
0017 12        stax d   ;store count in dest.arr
0018 13        inx d    ;increment to storage area
0019 23        inx h    ;increment to storage area
001A E3        xthl     ;m->arr2
001B 23        inx h    ;increment to storage area
001C 0C        inr      ; incr c for zero safety
001D 0D        loop:    dcr c
001E CA2C00    jz exit
0021 7E        mov a,m           ; m points to arr2
0022 23        inx h             ; incr pointer to arr2
0023 E3        xthl     ; m now points to arr1
0024 86        add m; add its element to arr2
0025 23        inx h    ; ince pointer to arr1
0026 12        stax d   ;store the sum in d.b array
0027 13        inx d    ;set new p->dest.array
0028 E3        xthl     ;m->arr2
0029 C31D00    jmp loop
002C E1        exit:    pop h    clean the stack
002D 76        hlt
;*****************************************
;******************************************
```

Input

A		B	**20**	D	**20**	H	**20**
F		C	**0B**	E	**06**	L	**00**

S		Z		A		P		S	

Memory

2000	09	09	08	07	06
2005	05	04	03	02	01
3000	04	01	02	03	04
3005	00	00	00	00	00
4000	04	0A	0A	0A	0A
4005	00	00	00	00	00
400A	00	00	00	00	00

Output

A	**0A**	B	**09**	D	**40**	H	**20**
F	**44**	C	**00**	E	**05**	L	**05**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	09	09	08	07	06
2005	05	04	03	02	01
3000	04	01	02	03	04
3005	00	00	00	00	00
4000	04	0A	0A	0A	0A
4005	00	00	00	00	00
400A	00	00	00	00	00

Hexadecimal to Binary Conversion

Objective

To implement a program to convert a hex-decimal number to an equivalent Binary ASCII Character string. Input is taken in the form of hex-decimal number in register A and the address of storage in H-L where the string is to be placed.

Statistics

Code Size	23 Bytes
T-State Count	464
Execution Time	232 x 10^{-6}sec @ 2.0 MHz

Program [hextobin.asm]

```
;hexa decimal to binary conversion
;converts a given byte in an ascii-z
;binary character string

0000 3E34      mvi a,34h;convert it into binary
0002 210020    lxi h,2000h;store it at address
;********************

0005 47        mov b,a; save param in B
0006 0E08      mvi c,8h         ;load counter in C
0008 78        loop: mov a,b    ;get rotated number
0009 07        rlc              ;rotate Acc left
000A 47        mov b,a          ;save num in B
000B 3E00      mvi a,0  ;set Acc to zero
000D CE30      aci 30h  ; Ascii '0'=30h & '1'=31h
000F 77        mov m,a;save the Bit in memory
0010 23        inx h    ;To next memory location
0011 0D        dcr c            ;decrement count
0012 C20800    jnz loop ;continue loop
0015 3600    exit: mvi m,0;end string with NULL char
0017 76        hlt
;*******************************************
;
;*******************************************
;
```

Input

A	**34**	B		D		H	**20**
F		C		E		L	**00**

S		Z		A		P		S	

Memory

2000	00	00	00	00	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**30**	B	**34**	D	**00**	H	**20**
F	**44**	C	**00**	E	**00**	L	**08**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	‘0’	‘0’	‘1’	‘1’	‘0’
2005	‘0’	‘0’	‘0’	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

16-Bit Register Left Shifting

Objective

To implement routines to perform 16-bit shift to the left of H-L Register. Takes input as a 16-bit number in Register Pair H-L and number of times the rotation is to be performed in register C.

Statistics

Code Size	18 Bytes
T-State Count	111
Execution Time	55 x 10^{-6}sec @ 2.0 MHz

Program [16bitshift.asm]

```
;16 bit arithmetic shifting to left
;shifts register h-l left by 1 bit a time
;(c) Vaibhav Jain
;date:9/4/04

0000 213412     lxi h,1234H; load data in H-L pair
0003 0E02       mvi c,2h; load count in c

0005 37         loop: stc ; set the carry flag
0006 3F         cmc ; this sets carry it to zero
0007 7D         mov a,l; copy lower 8 bits of data
0008 17         ral ; rotate accumulator left trough
                ;carry
0009 6F         mov l,a; copy back the contents to l
000A 7C          mov a,h;copy upper 8 bits
000B 17         ral ; rotate it through carry
000C 67         mov h,a; copy back the contents
;****Shift is complete perform loop
000D 0D         dcr c
000E C20500     jnz loop
0011 76 hlt ;
*** result is 18h in H-L pair
;*******************************************
;********************************************
```

Input

A		B		D		H	**12**
F		C	**02**	E		L	**34**

S		Z		A		P		S	

Output

A	**48**	B		D		H	**48**
F	**44**	C	**00**	E		L	**D0**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

16-Bit Register Right Shifting

Objective

To implement routines to perform 16-bit shift to the Right of H-L Register. Takes input as a 16-bit number in Register Pair H-L and number of times the rotation is to be performed in register C.

Statistics

Code Size	18 Bytes
T-State Count	111
Execution Time	55 x 10^{-6}sec @ 2.0 MHz

Program [16bitshiftr.asm]

```
;16 bit arithmetic shifting to right
;shifts register h-l right by 1 bit a time
;(c) Vaibhav Jain ;date:9/4/04s

0000 213412    lxi h,1234h; load data in H-L pair
0003 0E02      mvi c,2h; load count in c
0005 37 l      oop: stc ; set the carry flag
0006 3F        cmc ; this sets carry it to zero
0007 7C        mov a,h; copy lower 8 bits of data
0008 1F     rar ; rotate accumulator left trough carry
0009 67        mov h,a; copy back the contents to l
000A 7D        mov a,l;copy upper 8 bits
000B 1F        rar ; rotate it through carry
000C 6F        mov l,a; copy back the contents
;****Shift is complete perform loop
000D 0D        dcr c
000E C20500    jnz loop
0011 76        hlt
 ;*** result is 18h in H-L pair
;******************************************
;*******************************************
```

Input

A		B		D		H	**12**
F		C	**02**	E		L	**34**

S		Z		A		P		S	

Output

A	**8D**	B		D		H	**04**
F	**44**	C	**00**	E		L	**8D**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Uppercase to Lowercase & vice-versa

Objective

To perform uppercase to lowercase conversion of an ASCII-Z string by implementing two different procedures for it. Input to these functions is given in the form of address of string in H-L Register pair.

Statistics

Code Size	66 Bytes
T-State Count	801
Execution Time	400 x 10^{-6}sec @ 2.0 MHz

Program [u2lcase.asm]

```
Uppercase To Lowercase and vice-versa
; of an ascii-z string ;(c) Vaibhav Jain
;date: 15/4/04
;*************************
0000    210020  lxi h,2000h
0003    CD2800 call L2Ucase ;convert to lowercase
0006    217020  lxi h,200Ah; address second string
0009    CD0D00 call U2Lcase ;convert to lowercase
000C    76          hlt

;* ******* ;Uppercase to Lowecase ;
Input : H-L= Address of ASCII-Z string
;**********************
U2Lcase:
000D E5          push h   ;save h-l pair
000E F5          push psw ;save acc and flags
000F 7E          U2Lcase1: mov a,m ;get the char

0010 B7          ora a ;check if zero
0011 CA2500   jz U2Lcase2 ;exit if zero
0014 FE41      cpi 41h ;if >='A'(65)
0016 DA2100   jc U2Lcase3

0019 FE5B      cpi 5Bh ;if >='['(91)
001B D22100   jnc U2Lcase3

001E C620      adi 20h ;convert to lcase

0020    77          mov m,a ;save char to memory
                    U2Lcase3:
0021    23          inx h ;move to next pos
0022    C30F00           jmp U2Lcase1;continue loop
                    U2Lcase2:
0025    F1          pop psw ;restore acc and flags
0026    E1          pop h ;restore h-l pair
0027    C9          ret
;code continues
```

Input

A		B		D		H	**20**
F		C		E		L	**00**

S		Z		A		P		S	

Memory

2000	‘H’	‘e’	‘l’	‘l’	‘O’
2005	00	00	00	00	00
200A	‘W’	‘O’	‘R’	‘l’	‘D’
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A		B		D		H	**20**
F		C		E		L	**70**

S	**0**	Z	**0**	A	**0**	P	**0**	S	**0**

Memory

2000	‘H’	‘E’	‘L’	‘L’	‘O’
2005	00	00	00	00	00
200A	‘W’	‘O’	‘R’	‘L’	‘D’
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Uppercase to Lowercase & vice-versa

Program [u2lcase.asm]

```
;*********************** ;;***********************
;Lowecase to Uppercase
;Input : H-L= Address of ASCII-Z string
 ;**********************
L2Ucase:
0028    E5      push h          ;save h-l pair

0029    F5      push psw        ;save acc and flags
                L2Ucase1:
002A    7E      mov a,m         ;get the char

002B    B7      ora a           ;check if zero
002C    CA4000          jz L2Ucase2 ;exit if zero
002F    FE61    cpi 61h         ;if >='a'(97)

0031    DA3C00          jc L2Ucase3
0034    FE7B    cpi 7Bh         ;if >='{'(123)
0036    D23C00          jnc L2Ucase3

0039    D620    sui 20h         ;convert to Ucase
003B    77      mov m,a         ;save char to memory
003C    23      L2Ucase3:       inx h ;move to next pos
003D    C32A00          jmp L2Ucase1;continue loop
0040    F1      L2Ucase2: pop psw       ;restore acc and flags

0041    E1      pop h           ;restore h-l pair

0042    C9      ret

;***************************************
;*****************************************
;******************************************
```

Length of an ASCII-Z String

Objective

To determine the length of an ASCII-Z string by implementing two different procedures for it. Input to this functions is given in the form of address of string in H-L Register pair. The length is returned in the Register C.

Statistics

Code Size	23 Bytes
T-State Count	837
Execution Time	418 x 10^{-6}sec @ 2.0 MHz

Program [strlen.asm]

```
;8085 implementation of strlen() function
;of C standard library
;date: 15/4/04
;**************
0000    210020 lxi h,2000h ;load address of string
0003    CD0700 call Strlen ;
0006    76        hlt ;

; ;************************ ;
Strlen:
Calculates length of an Ascii-Z string
;Input: H-L: Adress of string
;Modifies: C: length of array, Flags
;************************
0007    F5        Strlen: push psw ;save acc and flags
0008    E5        push h ;save h-l pair

0009    0E00     mvi c,0h ;set c to zero
000B    7E        Strlenloop: mov a,m ;get the char

000C    B7        ora a ; check if zero
000D    CA1500           jz Strlenout; exit if zero

0010    0C        inr c ;incr counter
0011    23        inx h ;to next char pos

0012    C30B00           jmp Strlenloop; do the loop

0015    E1        Strlenout: pop h ;restore H-L pair
0016    F1        pop psw ;restore Acc and Flags
0017    C9        ret ;Job done Bye-Bye
;*******************************************
;********************************************
```

Input

A		B		D		H	**20**
F		C		E		L	**00**

S		Z		A		P		S	

Memory

2000	'A'	'B'	'C'	'D'	'E'
2005	'E'	'F'	'G'	'H'	'I'
200A	'J'	'K'	'L'	'M'	'N'
200F	'O'	'P'	'Q'	'R'	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B		D		H	**20**
F	**00**	C	**15**	E		L	**00**

S	**0**	Z	**0**	A	**0**	P	**0**	S	**0**

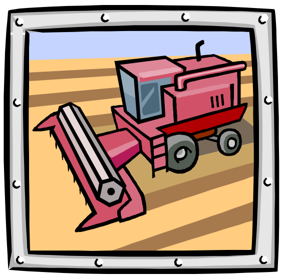
Memory

2000	'A'	'B'	'C'	'D'	'E'
2005	'E'	'F'	'G'	'H'	'I'
200A	'J'	'K'	'L'	'M'	'N'
200F	'O'	'P'	'Q'	'R'	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Implementing Strcpy() Function

Objective

Implementation of function strcpy in 8085 assembly language. Assuming all strings to be already allocated, and that the strings do not overlap. Address of source array is stored in H-L Pair and Destination is D-E Pair.



Statistics

Code Size	15 Bytes
T-State Count	422
Execution Time	211 x 10^{-6}sec @ 2.0 MHz

Program [strcpy.asm]

```
;program to copy ascii-z strings from
;source to destination
;(c) Vaibhav Jain
;Date: 9/4/2004

0000 21D007    lxi h,2000 ;source ascii-z string

0003 11B80B    lxi d,3000 ;destination ascii-z string
;*****************
loop:

0006    7E       mov a,m ; read src char
0007    12       stax d ; store it in destnation

0008    13       inx d ;incr to next dest pos

0009    23       inx h ;incr to next src pos

000A    B7       ora a ;check if A was zero

000B    C20600           jnz loop ;continue if not zero
000E    76       hlt ;program exit
;*******************************************
;********************************************
```

Input

A		B		D	**30**	H	**20**
F		C	**4**	E	**00**	L	**00**

S		Z		A		P		S	

Memory

2000	'A'	'B'	'C'	'D'	'E'
2005	'E'	'F'	'G'	'H'	00
3000	00	00	00	00	00
3005	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B		D	**30**	H	**20**
F	**44**	C		E	**08**	L	**08**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

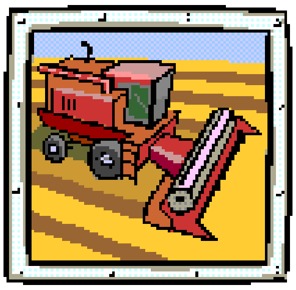
Memory

2000	'A'	'B'	'C'	'D'	'E'
2005	'E'	'F'	'G'	'H'	'I'
3000	'A'	'B'	'C'	'D'	'E'
3005	'E'	'F'	'G'	'H'	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Implementing Strncpy() Function

Objective

Implementation of function strcpy in 8085 assembly language. Assuming all strings to be already allocated, and that the strings do not overlap. Address of source array is stored in H-L Pair and Destination is D-E Pair and character count in Reg-C .



Statistics

Code Size	19 Bytes
T-State Count	248
Execution Time	124 x 10^{-6}sec @ 2.0 MHz

Program [strncpy.asm]

```
;Implement the following variant of the strncpy
;function:
;void strncpy(char * src, char * dest, unsigned char n)
;Assume all strings to be already allocated, and that
;the
;strings do not overlap. In strncpy not more than
;n bytes are copied.If there is no null (0) byte among
;the first n bytes of src, the result will not be null-
;terminated.
;Further, in the case where the length of src is less
than n, the ;remainder of dest will be padded with
;nulls
;(c) Vaibhav Jain
;Date: 9/4/2004

0000    21D007  lxi h,2000 ;source ascii-z string
0003    11B80B  lxi d,3000 ;destination ascii-z string
;*****************
0006    0C      inr c ;protection from zero count
0007    0D      loop: dcr c
0008    CA1300          jz exit
000B    7E      mov a,m ; read src char
000C    12      stax d ; store it in destnation
000D    13      inx d ;incr to next dest pos
000E    23      inx h ;incr to next src pos
000F    B7      ora a ;check if A was zero
0010    C20700          jnz loop ;continue if not zero
0013    76      exit: hlt ;program exit
;*******************************************
;********************************************
```

Input

A		B		D	**30**	H	**20**
F		C	**4**	E	**00**	L	**00**

S		Z		A		P		S	

Memory

2000	‘A’	‘B’	‘C’	‘D’	‘E’
2005	‘E’	‘F’	‘G’	‘H’	‘I’
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**69**	B		D	**30**	H	**20**
F	**44**	C		E	**04**	L	**04**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	‘A’	‘B’	‘C’	‘D’	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Generation of Fibonacci Series

Objective

To implement a program to generate fibonacci series. The program takes input as the count of number of terms to be generated in Register C and the address of storage in H-L.

Statistics

Code Size	18 Bytes
T-State Count	718
Execution Time	359 x 10^{-6}sec @ 2.0 MHz

Program [fibo.asm]

```
          ;Fibonacci Series program
;(c) Vaibhav Jain 2004
;*********Input*********
 0000   21D007 lxi h,2000
 0003   0E10    mvi c,10h
;**************Program******
 0005   AF      xra a
 0006   0601    mvi b,1
 0008   77      loop:    mov m,a
 0009   80      add b
 000A   57      mov d,a
 000B   78      mov a,b
 000C   42      mov b,d
 000D   23      inx h
 000E   0D      dcr c
 000F   C20800 jnz loop
 0012   76      hlt
;******************************************
;*******************************************
SYMBOL TABLE :

 LOOP           0008
```

Input

A		B		D		H	**20**
F		C	**10**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	00	00	00	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**C2**	B	**2F**	D	**2F**	H	**20**
F	**45**	C	**00**	E	**00**	L	**15**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

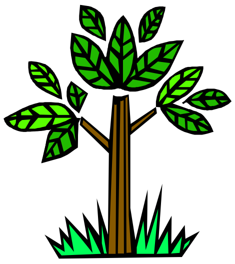
Memory

2000	00	01	01	02	03
2005	05	08	0D	15	22
200A	37	59	90	E9	79
200F	62	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Generation of Multiplication Tables

Objective

To implement a program to fill memory with multiples of given number. The program takes input as the number whose table is to be generated in Register B , Count of number of multiples to be generated in Register C and the address of storage in H-L.



Statistics

Code Size	19 Bytes
T-State Count	856
Execution Time	428 x 10^{-6}sec @ 2.0 MHz

Program [table.asm]

```
;program to create a table of given number
;from 1 to the given index on a selected
;memory location
;(c) Vaibhav Jain 2004
;date: 9/4/2004

 0000   0603    mvi b,3h          ;table of 3
 0002   0E15    mvi c,15h         ;from 0 to 21
 0004   21D007  lxi h,2000;address to store table

;**********************************

 0007   AF      xra a    ;set a to zero
 0008   0C      inr c    ;safety from zero count

                loop:
 0009   77      mov m,a
 000A   0D      dcr c
 000B   CA1300           jz exit
 000E   80      add b    ;add b to accumulator
 000F   23      inx h
 0010   C30900  jmp loop

 0013   76      exit: hlt
;****************************************
;*****************************************

SYMBOL TABLE :
LOOP   0009
EXIT   0013
```

Input

A		B		D		H	**20**
F		C	**15**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	00	00	00	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**3F**	B	**03**	D	**00**	H	**20**
F	**44**	C	**00**	E	**00**	L	**15**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

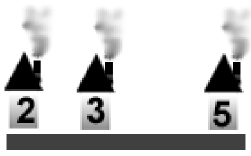
Memory

2000	00	03	06	09	0C
2005	0F	12	15	18	1B
200A	1E	21	24	27	2A
200F	2D	30	33	36	39
2014	3C	3E	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Insertion Sorting

Objective

To create a routine for simulating Insertion Sort Algorithm on 8085. Takes input as H-L pair containing address of the array to sort and number array elements in C register.



Statistics

Code Size	29 Bytes
T-State Count	3260
Execution Time	1630 x 10^{-6}sec @ 2.0 MHz

Program [insertion.asm]

```
SOURCE FILE NAME : INSERT~1.ASM

          ;Insertion sort routine
          ;vaibhav jain 05/04/2004
 0000 21D007  lxi h,2000        ;load adress of array
 0003 0E0A    mvi c,Ah          ;load no. of elements

 0005   AF    xra a    ; set acc to 0
 0006   0C    inr c
 0007   3C    outerloop: inr a  ;generate next
                                ;counter
 0008   B9    cmp c    ; compare with max counter
 0009   D22D00         jnc exit  ; exit if equal
 000C   C5    push b   ;save max counter info
 000D   F5    push psw; save current counter
 000E   E5    push h   ;save current base
 000F   4F    mov c,a;copy new counter to c
 0010   54    mov d,h;
 0011   5D    mov e,l  ;copy h-l to d-e
 0012   7E    mov a,m           ;get cmp element
 0013   2B    dcx h    ;set h-l cmp base

 0014   0D    innerloop:        dcr c     ; dcr counter
 0015   CA2500         jz outerout; exit loop if zero
 0018   BE    cmp m    ; cmp cur element to acc
 0019   D22500jnc outerout; exit loop if less than
 001C   47    mov b,a;save acc to b
 001D   7E    mov a,m           ;copy cur ele to acc
 001E   12    stax d   ; store it at prebase
 001F   78    mov a,b;copy cmpele back to acc
 0020   1B    dcx d    ;dcr d-l
 0021   2B    dcx h    ;dcr h-l
 0022   C31400jmp innerloop; jmp back to loop

 0025   12 outerout: stax d; store acc to curbase
;***********code continues**************
```

Input

A		B		D		H	**10**
F		C	**16**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	01	02	03	0A
2005	09	08	05	06	07
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**17**	B	**00**	D	**20**	H	**20**
F	**44**	C	**17**	E	**08**	L	**16**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	00	01	02	03	05
2005	06	07	08	09	0A
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Insertion Sorting

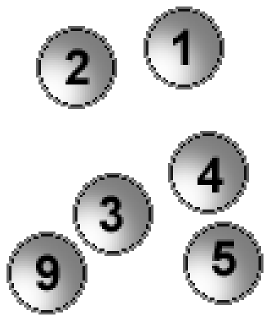
Program [insertion.asm]

```
0026    E1      pop h   ;retrive saved base
0027    F1      pop psw         ;retrive pre counter
0028    C1      pop b   ;retrive max counter
0029    23              inx h     ; inr h-l to next base
002A    C30700jmp outerloop; jmp to next iteration
002D    76      exit:     hlt

;***********code concludes**************

 SYMBOL TABLE :

OUTERLOOP   0007
INNERLOOP   0014
OUTEROUT    0025
EXIT    002D

;*****************************************
;******************************************
```

Bubble Sorting-I

Objective

To create a routine for simulating Bubble Sort Algorithm on 8085 through simulation on 16-bit subtraction and without using stack space. Takes input as H-L pair containing address of the array to sort and number array elements in C register.



Statistics

Code Size	34 Bytes
T-State Count	3708
Execution Time	1854 x 10^{-6}sec @ 2.0 MHz

Program [bubblesrt.asm]

```
SOURCE FILE NAME : BUBBLE~1.ASM

          ;Bubble sort routine Part 2
          ;********************
 0000     21D007 lxi h,2000
 0003     0E0A   mvi c,0Ah
 0005     AF     xra a
 0006     B1     ora c
 0007     CA2200        jz exit
 000A     0D     dcr c
 000B     C5     outerloop:      push b
 000C     E5     push h
 000D     7E     innerloop:      mov a,m
 000E     23     inx h
 000F     BE     cmp m
 0010     DA1800        jc innerout
 0013     46     mov b,m
 0014     77     mov m,a
 0015     2B     dcx h
 0016     70     mov m,b
 0017     23     inx h
 0018     0D     innerout:       dcr c
 0019     C20D00        jnz innerloop
 001C     E1     pop h
 001D     C1            pop b
 001E     0D     dcr c
 001F     C20B00        jnz outerloop
exit::
 0022     76     hlt

SYMBOL TABLE :
OUTERLOOP    000B
INNERLOOP    000D
INNEROUT     0018
EXIT         0022
;*****************************************
;******************************************
```

Input

A		B		D		H	**20**
F		C	**0A**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	09	08	07	06
2005	05	04	03	02	01
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**1**	B	**01**	D	**20**	H	**20**
F	**44**	C	**01**	E	**00**	L	**00**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

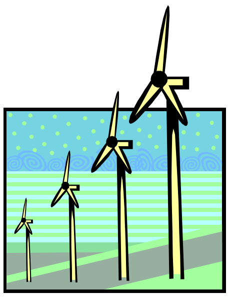
Memory

2000	00	01	02	03	04
2005	05	06	07	08	09
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Bubble Sorting-II

Objective

To create a faster routine for simulating Bubble Sort Algorithm on 8085 by using stack space. Takes input as H-L pair containing address of the array to sort and number array elements in C register.



Statistics

Code Size	43 Bytes
T-State Count	3593
Execution Time	1796 x 10^{-6}sec @ 2.0 MHz

Program [bubblesrt2.asm]

```
;bubble sort Program
;without using stack
;(c) Vaibhav Jain 2003
          ;*****Input************
 0000   21D007  lxi h,2000        ;address of array
 0003   0E16    mvi c,16h         ; no. of elements
          ;********Program********
 0005    54     mov d,h
 0006    5D     mov e,l
outerlop:
 0007   46      mov b,m
 0008   0D      dcr c
 0009   78      loop:     mov a,b
 000A   23      inx h
 000B   46      mov b,m
 000C   B8      cmp b
 000D   DA1500            jc loop2
 0010   2B      dcx h
 0011   70      mov m,b
 0012   23      inx h
 0013   77      mov m,a
 0014   47      mov b,a
 0015   0D      loop2:dcr c
 0016   C20900  jnz loop
 0019   7C      mov a,h
 001A   2F      cma
 001B   67      mov h,a
 001C   7D      mov a,l

 001D   2F      cma
 001E    6F     mov l,a

 001F   19      dad d
 0020   7D      mov a,l

;***********code continues**************
```

Input

A		B		D		H	**20**
F		C	**0A**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	09	08	07	06
2005	05	04	03	02	01
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B	**00**	D	**00**	H	**20**
F	**45**	C	**00**	E	**00**	L	**00**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	00	01	02	03	04
2005	05	06	07	08	09
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Bubble Sort-II

Program [bubblesrt2.asm]

```
 0021    2F       cma
 0022   4F        mov c,a
 0023   62        mov h,d
 0024   6B        mov l,e
 0025   FE01      cpi 1
 0027   C20700          jnz outerlop
 002A   76        hlt

;***********code concludes**************
SYMBOL TABLE :

OUTERLOP     0007
LOOP         0009
LOOP2        0015
;*****************************************
;******************************************
```

Selection Sorting

Objective

To create a routine for simulating Selection Sort Algorithm on 8085. Takes input as H-L pair containing address of the array to sort and number array elements in C register.

Statistics

Code Size	38 Bytes
T-State Count	3439
Execution Time	1719 x 10^{-6}sec @ 2.0 MHz

Program [selsort2.asm]

```
0000    21D007 lxi h,2000
0003    0E0A    mvi c,0Ah
0005    AF      xra a
0006    B1      ora c
0007    CA2600          jz exit
000A    54      outerloop:      mov d,h
000B    5D      mov e,l
000C    7E      mov a,m
000D    C5      push b
000E    E5      push h
000F    BE      innerloop:      cmp m
0010    DA1600          jc innerout
0013    54              mov d,h
0014    5D      mov e,l
0015    7E      mov a,m
0016    23      innerout:       inx h
0017    0D      dcr c
0018    C20F00jnz innerloop
001B    E1      pop h
001C    C1      pop b
001D    46      mov b,m
001E    77              mov m,a
001F    78      mov a,b
0020    12      stax d
0021    23      inx h
0022    0D      dcr c
0023    C20A00          jnz outerloop

0026    76      exit:   hlt

SYMBOL TABLE :

OUTERLOOP   000A
INNERLOOP   000F
INNEROUT    0016
EXIT    0026
```

Input

A		B		D		H	**20**
F		C	**0A**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	09	08	07	06
2005	05	04	03	02	01
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**09**	B	**09**	D	**20**	H	**20**
F	**44**	C	**00**	E	**09**	L	**0A**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	00	01	02	03	04
2005	05	06	07	08	09
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

8-Bit Division Algorithm

Objective

To implement a program to simulating 8-bit division on 8085. Takes input as dividend in Register B and Divisor in Register C. After division Quotient is stored in B and Remainder in C.

Statistics

Code Size	55 Bytes
T-State Count	290
Execution Time	145×10^{-6}sec @ 2.0 MHz

Program [div8bit.asm]

```
SOURCE FILE NAME : DIV8BIT.ASM
;Division of 8085
;Divides Value of Reg.B with Value of Reg.C
;stores Quotient in B and Remainder in C
 0000   06FE    mvi b,feh
 0002   0E10    mvi c,10h
;**************************
 0004   79      mov a,c
 0005   0E08    mvi c,8h
loop:
 0007   07      rlc        ;
 0008   DA1400 jc loopout        ;if end reached exit
 000B   0D      dcr c    ;decrement counter
 000C   C20700jnz loop ;do the shift loop
  ;****there is an overflow******
 000F   D1      pop d    ;Restore D-E
 0010   37      stc      ;set CY to show error
 0011   06FF    mvi b,ffh        ;set highest quiotient
 0013   76      hlt
loopout:
 0014    0F     rrc      ;move 1bit back to MSB
 0015   3F      cmc      ;set CY to zero
 0016   1E00    mvi e,0 ;set Quotient to zero
 0018   57      mov d,a;save Divisor in D
 0019   3E08    mvi a,8h         ;load a with 8h
 001B   91      sub c    ;Minus remaing count
 001C   3C      inr a    ;incr count by 1
 001D   4F      mov c,a;save this count in c
 001E   78      mov a,b;restore dividend

        ;NFE==>No Flags Effected
loopdiv:
 001F   BA      cmp d   ;cmp Divident with
        ;divisor
 0020   47      mov b,a;NFE:save dividend ->B
 0021   7B      mov a,e;NFE:load quotient
;***********code continues**************
```

Input

A		B	**FE**	D		H	
F		C	**10**	E		L	

S		Z		A		P		S	

Output

A	**0E**	B	**0F**	D	**08**	H	
F	**44**	C	**0E**	E	**0F**	L	

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

8-Bit Division Algorithm

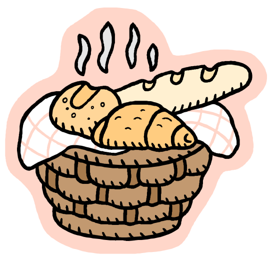
Program [div8bit.asm]

```
 0022   3F       cmc      ;complement CY
        ;quotient<<1 if Divisor<divident
        ;<<0 if Divisor>divident
 0023   17       ral      ;insert Cary bit in
 0024   0F       rrc      ;restore state of CY
 0025   07       rlc      ;
 0026   5F       mov e,a;save quitoent in E
 0027   D22D00            jnc skipdivsub ;Divisor>Divident=>Skip
 002A   78                          mov a,b;load the dividend
 002B   92       sub d    ;minus the divisor
 002C   47       mov b,a;save new divident
skipdivsub:
 002D   7A       mov a,d;restore divisor<== D
 002E   0F       rrc      ;rotate divisor right
 002F   57       mov d,a;save divisor in D
 0030   78       mov a,b;load A with dividend
 0031   0D       dcr c
 0032   C21F00jnz loopdiv
 0035   48       mov c,b;save remainder in C
 0036   43       mov b,e;save quotient in B
 0037   76       hlt
;***********code concludes**************


SYMBOL TABLE :
LOOP             0007
LOOPOUT           0014
LOOPDIV          001F
SKIPDIVSUB       002D
;*******************************************
;*******************************************
```

16-Bit Multiplication Algorithm

Objective

To implement a program to find a 16-bit product of two 8-bit numbers. Takes input as first operand in Register H and second operand in Register L. After multiplication the 16-bit product is stored in H-L pair with MSB in Register H and MSB in L.



Statistics

Code Size	44 Bytes
T-State Count	1115
Execution Time	557 x 10^{-6}sec @ 2.0 MHz

Program [mult.asm]

```
SOURCE FILE NAME : MULT.ASM

  ;Binary Multiplication Algorithm for 8085
;takes the operands in H & L registers
;stores 16-bit multiplication result
;in H-L pair
;(c) Vaibhav Jain
;date 09/4/2004

 0000   26EF    mvi h,efh;          operand1 in H
 0002   2EF0    mvi l,f0h;          operand2 in L

        ;*******Initialization*******
 0004    7C     mov a,h; copy operand 1 to acc
 0005   5D      mov e,l ; copy operand 2 to e
 0006   1600    mvi d,0 ; set register d to 0
 0008   210000 lxi h,0  ; set multiplication result to
                        ;zero
 loop:
 000B   B7      ora a   ;check if a if now zero
 000C   CA2000          jz exit   ;exit loop if true
 000F   37      stc     ;set carry flag
 0010   3F      cmc     ;complement cary flag
 0011   1F      rar     ;rotate acc right through
                        ;carry
 0012   D21600 jnc cond1;skip nxt step if no carry
 0015   19      dad d   ;double add d-e to h-l
cond1:
 0016   EB      xchg    ;xchange D-E & H-L
 0017   CD2100 call LSHL        ;left shift H-L one bit
 001A   EB      xchg    ;re-xchange D-E & H-L
 001B   E67F    ani 7Fh ;mask out leftmost bit of acc
 001D   C30B00 jmp loop;return in loop
 0020   76      exit:    hlt;      the job is now
complete
;*****************code continues************
```

Input

A		B		D		H	**EF**
F		C		E		L	**F0**

S		Z		A		P		S	

Output

A	**00**	B		D	**F0**	H	**E0**
F	**44**	C		E	**00**	L	**10**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

16-Bit Multiplication Algorithm

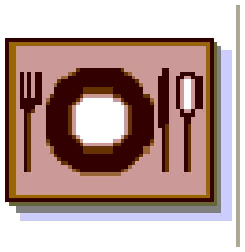
Program [mult.asm]

```
;*********************************
;****Procedure: Left Shift HL*****
;left shifts contents of HL left one bit
;H-L: Data to be shifted
;Flags Effected: CY
;*********************************
LSHL:
 0021   47      mov b,a         ;copy acc to b
 0022   37      stc             ; set the carry flag
 0023   3F      cmc             ; this sets carry it to zero
 0024   7D      mov a,l         ; copy lower 8 bits of data
 0025   17      ral             ; rotate accumulator left trough carry
 0026   6F      mov l,a         ; copy back the contents to l
 0027   7C      mov a,h         ;copy upper 8 bits
 0028   17      ral             ; rotate it through carry
 0029   67      mov h,a         ; copy back the contents
 002A   78      mov a,b         ;copy previous contents of acc
 002B   C9      ret             ;job is one and now return

;**********code concludes***************
SYMBOL TABLE :

LOOP            000B
COND1            0016
EXIT            0020
LSHL            0021
;************* **************************
;************* **************************
```

Removal of Duplicates

Objective

To implement a program to perform inline duplicate removal from a sorted array of 8-bit unsigned numbers. Takes input as address of array H-L pair and number of elements in C. After removal of duplicates count of unique elements is stores in B



Statistics

Code Size	34 Bytes
T-State Count	1397
Execution Time	698 x 10^{-6}sec @ 2.0 MHz

Program [dup.asm]

```
              ;removal of duplicates from a sorted array
              ; stores no. of unique elements in c
0000 21D007    lxi h,2000          ;address of array
0003 0E16      mvi c,16h; count of no. of elements
0005 E5        push h
 0006 54       mov d,h
 0007 5D       mov e,l
 0008 1A       loop:    ldax d
 0009 BE       cmp m
 000A CA0E00   jz lout ; if equal skip following steps
 000D 13       inx d
 000E 7E       lout:    mov a,m
 000F 12       stax d
 0010 23       inx h
 0011 0D       dcr c
 0012 C20800   jnz loop
 0015 E1       exit:    pop h
 0016 EB       xchg
 0017 7C       mov a,h
 0018 2F       cma
 0019 67       mov h,a
 001A 7D       mov a,l
 001B 2F       cma
 001C 6F       mov l,a
 001D 19       dad d
 001E 7D       mov a,l
 001F 2F       cma
 0020 4F       mov c,a
 0021 0C                inr c
 0022 76       hlt


 SYMBOL TABLE :
 LOOP 0008 LOUT   000E EXIT 0015
```

Input

A		B		D		H	**20**
F		C	**16**	E		L	**00**

S		Z		A		P		S	

Memory

2000	00	01	01	02	02
2005	02	03	03	03	04
200A	04	04	04	05	05
200F	05	05	05	06	06
2014	06	06	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**06**	B	**00**	D	**20**	H	**FF**
F	**00**	C	**07**	E	**00**	L	**F9**

S	**0**	Z	**0**	A	**0**	P	**0**	S	**0**

Memory

2000	01	02	03	04	05
2005	06	03	03	03	04
200A	04	04	04	05	05
200F	05	05	05	06	06
2014	06	06	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Sorted Array Merging

Objective

To implement a program to amalgate two bounded and sorted arrays into a third bounded array. A bounded array is a object having count of number of elements in it present on its top. The program takes input of first source array in H-L , second source array in D-E & the destination array in B-C.

Statistics

Code Size	71 Bytes
T-State Count	1508
Execution Time	754 x 10^{-6}sec @ 2.0 MHz

Program [arrmerge2.asm]

```
            ;merging of two sorted arrays into an
            ;sorted array
0000 21D007    lxi h,2000;source bounded array1
 0003 11B80B   lxi d,3000;source bounded array2
 0006 01A00F   lxi b,4000;destination bounded array
            ;*************************************
 0009 1A        ldax d   ;load count of arr2
 000A 86        add m   ;add count of arr1 to that of
arr1
 000B 02        stax b   ;store length in dest.
bounded array
 000C E5        push h   ;save address of arr1
 000D 62        mov h,d;
 000E 6B        mov l,e  ;load address of arr2 in h-l
 000F 50        mov d,b;
 0010 59        mov e,c;load address of dest.arr in
d-e
 0011 4E        mov c,m           ;load count of arr2 in
b
 0012 23        inx h    ;incr p->arr2 to storage area
 0013 E3        xthl     ;get p->arr1
 0014 46        mov b,m;load count of arr1 in b
;*************************************
loop:     ;m->arr1
 0017 AF        xra a    ;set a to zero
 0018 B0        ora b    ;check if b is zero
 0019 CA3600    jz lout
 001C E3        xthl
            ;m->arr2
 001D AF        xra a    ;set a to zero
 001E B1        ora c    ;check if c is zero
 001F CA3600    jz lout
            ;m->arr2
 0022 7E        mov a,m           ;m points to arr2
 0023 0D        dcr c    ;assuming that element

 0024 23        inx h    ;in arr2 i.e acc is least
;/**********conde continues**************
```

Input

A		B	**20**	D	**20**	H	**20**
F		C	**0B**	E	**06**	L	**00**

S		Z		A		P		S	

Memory

2000	09	01	03	05	07
2005	09	0B	0D	0F	11
3000	04	00	02	04	06
3005	00	00	00	00	00
4000	00	00	00	00	00
4005	00	00	00	00	00
400A	00	00	00	00	00

Output

A	**11**	B	**06**	D	**40**	H	**20**
F	**44**	C	**00**	E	**0E**	L	**0A**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	09	01	03	05	07
2005	09	0B	0D	0F	11
3000	04	00	02	04	06
3005	00	00	00	00	00
4000	0D	00	01	02	03
4005	04	05	06	07	09
400A	0B	0D	0F	11	00

Sorted Array Merging

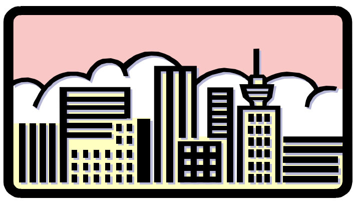
Program [arrmerge2.asm]

```
0025 E3           xthl      ;m->arr1
 0026 BE          cmp m  ;m points to arr1
 0027 DA3100   jc cond2           ;jmp if assumption is true
           ;*************************************
 002A 7E          mov a,m           ;rectify the mistake
 002B 23          inx h     ;incr p->arr1 to new pos
 002C 05          dcr b     ;set new count of elems in arr1

 002D E3          xthl      ;m->arr2
 002E 2B          dcx h    ;decr p->arr2 back
 002F 0C                   inr c      ;incr c back

 0030 E3          xthl      ;reenforce the previous situation
           ;i.e m->arr1
           ;*************************************
 0031 12          cond2:  stax d   ;a contains min element
           ;store it to new dest.
 0032 13                            inx d      ;set new dest pointer
 0033 C31700   jmp loop
           ;************************************
 0036 E1          lout:      pop h    ;get pointer to remaing

           ;elements
 0037 AF          xra a     ;set a to 0
 0038 B0          ora b     ;get count in b
 0039 B1          ora c     ;get count in c
 003A 3C          inr a     ;increment a for zero safety
 003B 4F          mov c,a;set generated count in c
           ;***loop to copy remaining elements*******
   003C 0D        loop2:   dcr c
 003D CA4700  jz exit
 0040 7E          mov a,m
 0041 12          stax d
 0042 13          inx d
 0043 23          inx h
 0044 C33C00   jmp loop2
           ;****************************************
 0047 76          exit:      hlt

;/**********conde concludes**************
 SYMBOL TABLE :

 LOOP 0017 COND2 0031 LOUT 0036 LOOP2  003C
 EXIT 0047
```

32-Bit Integer Addition

Objective

To implement program to perform 32-bit addition. The integers are stores in memory in reverse byte order i.e. LSB lies in lower memory and MSB lies in higher memory.



Statistics

Code Size	24 Bytes
T-State Count	359
Execution Time	179 x 10^{-6}sec @ 2.0 MHz

Program [add32bit.asm]

```
;32-bit addition
;take address of operand1 in h-l &
;& address of operand2 in d-e
;stores result at operand1
;operands are stored in BIG-Edian format
 0000   21D007  lxi h,2000
 0003   11D407  lxi d,2004

;***************************
 0006   0E04    mvi c,4
 0008   AF      xra a; set carry flag to zero
 0009   F5      push psw; save flags
                loop:
 000A   F1      pop psw ; restore flags
 000B   3E00    mvi a,0
 000D   1A      ldax d
 000E   8E      adc m
 000F   77      mov m,a
 0010   13      inx d
 0011   23      inx h
 0012   F5      push psw; save flags
 0013   0D      dcr c
 0014   C20A00 jnz loop
 0017   F1      pop psw
 0018   76      hlt


 SYMBOL TABLE :

 LOOP 000A
```

Input

A		B		D	**20**	H	**20**
F		C	**04**	E	**08**	L	**04**

S		Z		A		P		S	

Memory

2000	1	02	03	04	05
2005	06	07	08	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**0C**	B		D	**20**	H	**20**
F	**04**	C		E	**08**	L	**04**

S	**0**	Z	**0**	A	**0**	P	**1**	S	**0**

Memory

2000	06	08	0A	0C	05
2005	06	07	08	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

32-Bit Integer Subtraction

Objective

To implement program to perform 32-bit subtraction. The integers are stores in memory in reverse byte order i.e. LSB lies in lower memory and MSB lies in higher memory.

Statistics

Code Size	25 Bytes
T-State Count	363
Execution Time	181 x 10^{-6}sec @ 2.0 MHz

Program [sub32bit.asm]

```
;32-bit subtraction
;take address of operand1 in h-l &
;& address of operand2 in d-e
;stores result at operand1
;operands are stored in BIG-Edian format
 0000   21D007        lxi h,2000
 0003   11D407        lxi d,2004

**********************************************
 0006   0E04    mvi c,4
 0008   AF      xra a; set carry flag to zero
 0009   F5      push psw; save flags
 000A   EB      xchg;xchange addresses of perands
        loop:
 000B   F1      pop psw         ; restore flags
 000C   1A      ldax d  ;get operand1 byte
 000D   9E      sbb m   ;subtract with borrow
 000E   12      stax d
 000F   3E00    mvi a,0
 0011   F5      push psw        ;save flags and acc
 0012   13      inx d
 0013   23      inx h
 0014   0D      dcr c
 0015   C20B00          jnz loop
 0018   F1      pop psw
 0019   76      hlt
;*******************************************
 SYMBOL TABLE :
 LOOP           000B
```

Input

A		B		D	**20**	H	**20**
F		C	**04**	E	**04**	L	**00**

S		Z		A		P		S	

Memory

2000	01	02	03	04	05
2005	06	07	08	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B		D	**20**	H	**20**
F	**91**	C		E	**04**	L	**08**

S	**1**	Z	**0**	A	**1**	P	**1**	S	**0**

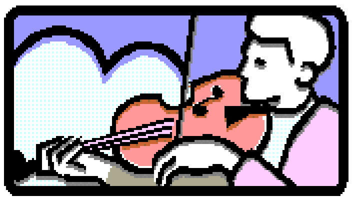
Memory

2000	FC	FB	FB	FB	05
2005	06	07	08	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

32-Bit Left Shift & Right Shift

Objective

To implement a program to perform Left & right Shift on 32-bit numbers stored in Big –Edian format. The function Shr32 performs Shift to right with carry bit fill in the empty bit. And the function Shl32 performs shift to left with carry bit being places in the empty LSB.



Statistics

Code Size	91 Bytes
T-State Count	1275
Execution Time	637 x 10^{-6}sec @ 2.0 MHz

Program [Shift32bit.asm]

```
;*******32 bit Shifting*********
 0000   31FFFF          lxi sp,ffffh
 0003   21D007          lxi h,2000
 0006   CD3200          call Shr32
 0009   CD1000          call Shl32
 000C   CD1000          call Shl32
 000F    76        hlt
;******Shift Left 32-bit*************
;Shift Left a 32 bit number pointed by H-L
;through Cary Fill
;operands: H-L address of number
;modifies CY flag
;*******************************
                Shl32:
 0010   E5      push h
 0011   D5      push d
 0012   C5      push b
 0013   F5      push psw
 0014   0600    mvi b,0h          ; set B to zero
 0016   0E04    mvi c,4h          ; set c to count
 0018   50      mov d,b; set D to zero
                Shlloop:
 0019   7E      mov a,m
 001A   17      ral
 001B   D21F00 jnc Shlnc1; if no cary then SKIP
 001E   14      inr d    ; Save present CY in D
                Shlnc1:
 001F    80     add b    ;add previous CY to num.
 0020   77      mov m,a           ;save it back
 0021   42      mov b,d;save Cy of current Op to b
 0022   1600    mvi d,0 ;NFC==> set D to zero
 0024   23      inx h
 0025   0D      dcr c
 0026   C21900          jnz Shlloop
;*******Code Continues*********
```

Input

A		B		D		H	**20**
F		C		E		L	**00**

S		Z		A		P		S	

Memory

2000	01	02	03	04	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B	**00**	D	**00**	H	**20**
F	**00**	C	**00**	E	**00**	L	**00**

S	**0**	Z	**0**	A	**0**	P	**0**	S	**0**

Memory

2000	02	04	06	08	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

32-Bit Left Shift & Right Shift

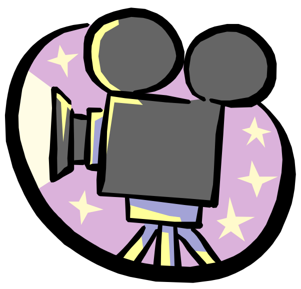
Program [Shift32bit.asm]

```
 0029   78      mov a,b; save last carry to Acc
 002A   C1      pop b   ; pop Acc & psw to B-C
 002B   47      mov b,a; set flags for bit have CY
 002C   C5      push b  ; push the Acc & new Flags
 002D   F1      pop psw         ; pop new data to flgs & acc
 002E   C1      pop b   ; pop old b-c
 002F   D1      pop d   ; pop old d-e
 0030   E1      pop h   ; pop old h-l
 0031   C9      ret
;******************************
;******Shift Right 32-bit*************
;Shift Right a 32 bit number pointed by H-L
;through Cary Fill
;operands: H-L address of number
;modifies CY flag
;*******************************
Shr32:
 0032   E5      push h
 0033   C5      push b
 0034   F5      push psw
 0035   F5      push psw        ;save flg+CY for operations
 0036   0E04    mvi c,4h        ;load count in C
 0038   0600    mvi b,0 ;set b-c pair to count
 003A   09      dad b   ;add to h-l pair to get
 003B   2B      dcx h   ;bottom of integer
 Shr32loop1:
 003C   F1      pop psw         ;get the previous flags
 003D   7E      mov a,m         ;get the data chuck
 003E   1F      rar     ;rotate right through CY
 003F   F5      push psw        ;save any CY generated
 0040   77      mov m,a         ;save data chuck back to mem
 0041   2B      dcx h   ;to new data chuck
 0042   0D      dcr c   ;decrement counter
 0043   C23C00          jnz Shr32loop1
 0046   F1      pop psw         ;restore previous flag
 0047   3E00    mvi a,0 ;set acc to zero
 0049   CE00    aci 0   ;trap the Cy in Acc
 004B   C1      pop b   ;pop original psw in b
 004C   47      mov b,a;copy the CY to stacked PSW
 004D   C5      push b  ;push new PSW
 004E   F1      pop psw         ;pop original PSW
 004F   C1      pop b   ;pop original B
 0050   E1      pop h   ;pop original h
 0051   C9      ret
;******************************
 SYMBOL TABLE :
SHL32         0010
SHLLOOP       0019
SHLNC1        001F
SHR32         0032
SHR32LOOP1    003C
```

32-Bit Bitwise Rotation

Objective

To implement a program to perform left & right rotate on 32-bit numbers stored in Big –Edian format. The function ROR32 performs rotation to right with carry bit set if overflow And the function ROL32 performs shift to left with carry bit set in case LSB being set.



Statistics

Code Size	63 Bytes
T-State Count	1095
Execution Time	547 x 10^{-6}sec @ 2.0 MHz

Program [Rotate32t.asm]

```
;32 bit rotation
 0000   21D007 lxi h,2000
 0003   CD2500          call ROR32
 0006   CD0D00          call ROL32
 0009   CD0D00          call ROL32
 000C   76        hlt
;******ROtate Left 32-bit*************
;Rotates Left a 32 bit number pointed by H-L
;operands: H-L address of number
;modifies CY flag , Acc
;********************************
                ROL32:
 000D   E5      push h
 000E   A7      ana a    ;reset CY flag
 000F   0E04    mvi c,4
 0011   F5      push psw
                loop:
 0012   F1      pop psw
 0013   7E      mov a,m
 0014   17      ral
 0015   77      mov m,a
 0016   23      inx h
 0017   F5      push psw
 0018   0D      dcr c
 0019   C21200          jnz loop
 001C   F1      pop psw
 001D   E1      pop h
 001E   7E      mov a,m
 001F   CE00    aci 0    ; copy the cary bit to LSB
 0021   77      mov m,a
 0022   0F      rrc
 0023   07      rlc
 0024   C9      ret
;*******************************
;******ROtate Right 32-bit*************
;Rotates right a 32 bit number pointed by H-L
;operands: H-L address of number
;**************code continues***********
```

Input

A		B		D		H	**20**
F		C		E		L	**00**

S		Z		A		P		S	

Memory

2000	01	02	03	04	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**02**	B		D		H	**20**
F		C		E		L	**00**

S	**0**	Z	**0**	A	**0**	P	**0**	S	**0**

Memory

2000	02	04	06	08	00
2005	00	00	00	00	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

32-Bit Bitwise Rotation

Program [Rotate32t.asm]

```
;modifies CY flag,Acc
;*******************************
                ROR32:
 0025   E5      push h
 0026   A7      ana a    ;reset CY flag
 0027   0E04    mvi c,4
 0029   23      inx h;
 002A   23      inx h;
 002B   23      inx h;
 002C   E5      push h;save this address for later refrence
 002D   F5      push psw
                ROR32loop:
 002E   F1      pop psw
 002F   7E      mov a,m
 0030   1F      rar
 0031   77      mov m,a
 0032   2B      dcx h
 0033   F5      push psw
 0034   0D      dcr c
 0035   C22E00          jnz ROR32loop
 0038   F1      pop psw
 0039   E1      pop h ; get address of last byte
 003A   7E      mov a,m
 003B   17      ral
 003C   0F      rrc

 003D   77      mov m,a
 003E   E1      pop h
 003F   C9      ret
 ;***************code concludes***************

SYMBOL TABLE :
ROL32           000D
LOOP            0012
ROR32           0025
ROR32LOOP       002E
```

Implementing Auxiliary Stack

Objective

To implement an auxiliary byte stack library which provided functionality of a stack apart from main stack. The stack only stores bytes instead of words. It consists of following functions:

- **Stkinit:** Initializes the stack. Set storage area to given location and length of the stack
- **Stkpush:** Pushes the byte present in accumulator to the stack. Sets CY flag on stack overflow.
- **Stkpop:** Pop a byte from the stack and places it in the accumulator. Sets CY flag on stack underflow.

Program [auxstk.asm]

```
;Auxilarry stack Library
;Provides byte storing stack
;stack can be placed on any position in memory
;and of any length
;(c) Vaibhav Jain
;date: 15/4/04
;************************
 0000   31FFFF lxi sp,ffffh
                          ;initialize processor stack
 0003   210020 lxi h,Array1
                          ;load stack storage address
 0006   0E09    mvi c,09h ;load byte count
 0008   CD2900call Stkinit ;initialize the stack
 000B   AF      xra a
 000C   4F      mov c,a
 000D   3C      loop: inr a
 000E   0C      inr c
 000F   CD4F00          call Stkpush;
                          ;push the data in stack
 0012   D20D00          jnc loop
                ;push data untill stack overflow
 0015   0E00    mvi c,00h
 0017   210920 lxi h,Array2
 001A   2B      dcx h

 001B   23      loop2:  inx h
 001C   0C      inr c
 001D   CD3700          call Stkpop
                ; get the data back from stack
0020    77      mov m,a
0021    D21B00          jnc loop2         ;pop untill
stack underflow
 0024   76      hlt
;****************************************
;***********Auxillary Stack Library*****
;Local Data
;//********code continues************
```

Input

A		B		D		H	**20**
F		C	**09**	E		L	**00**

S		Z		A		P		S	

Memory

2000	01	02	03	04	05
2005	06	07	08	09	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**00**	B	**00**	D	**00**	H	**20**
F	**44**	C	**0A**	E	**00**	L	**12**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	01	02	03	04	05
2005	06	07	08	09	09
200A	08	07	06	05	04
200F	03	02	01	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Implementing Auxiliary Stack

Program [auxstk.asm]

```
;relocate to suitable RAM Location
 0025    00       stkcount: db 00 ;count of current data
 0026    00       stklimit: db 00 ;max elements
 0027    0000     stkaddress: dw 0000 ;stack address
;
;*********Initialize Stack**********
;initializes the stack
;Input: c= no. of byte of stack
; H-L=address of stack memory
;destroys: nothing
;***********************************
                  Stkinit:
 0029    F5       push psw
 002A    AF       xra a ;set a to zero
 002B    322500 sta stkcount ;store counter in count
 002E    79       mov a,c
 002F    322600 sta stklimit
 0032    222700 shld stkaddress
 0035    F1       pop psw
 0036    C9       ret
;
;*********Pop Stack data**********
;pop a byte from stack to Accumulator
;Input: nothing
;modifies: Acc with poped data
; CY is set if stack is empty
;***********************************
                  Stkpop:
 0037    3A2500  lda stkcount   ;load stack counter
 003A    B7       ora a ;check if zero
 003B    C24000            jnz Stkpop_c1
 003E    37       stc ;set CY to indicate error
 003F    C9       ret
                  Stkpop_c1:
 0040    E5       push h ;save hl pair
 0041    2A2700 lhld stkaddress ;load address of stack
 0044    2B       dcx h ;decrement stack top address
 0045    3D       dcr a ;decrement stack counter
 0046    322500  sta stkcount ;save new stack counter
 0049    222700   shld stkaddress ;save new stack top
 004C    7E       mov a,m ;get data to pop
 004D    E1       pop h ;restore h-l pair
 004E    C9       ret
 ; ;*********Push data in Stack**********
;push a byte in stack from Accumulator
;Input: Acc with data
;modifies: CY is set if stack is full
;************************************
004F     C5       Stkpush: push b   ; save B-C pair
0050     47       mov b,a ; save the push data
0051     3A2500 lda stkcount ; load stack counter
;//**************************code continues*****************************
```

Implementing Auxiliary Stack

Program [auxstk.asm]

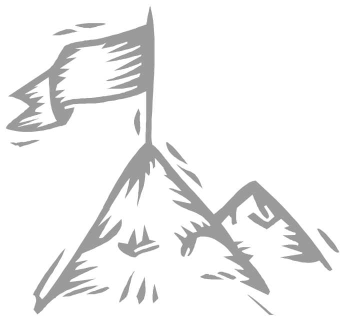
```
0054   4F       mov c,a ; copy it to C
0055   3A2600   lda stklimit ; get max limit
0058   B9       cmp c  ; compair it to C i.e counter
0059   C26000   jnz Stkpush_c1 ; if count != maxlimit jump
005C   78       mov a,b ; restore the data in Acc
005D   C1       pop b ; restore B-C pair
005E   37       stc ; Signal Error
005F   C9       ret ; return
                Stkpush_c1:
0060   0C       inr c ;increment counter
0061   79       mov a,c ;copy it to acc
0062   322500   sta stkcount ;save new counter

0065   E5       push h   ;save h-l pair
0066   78       mov a,b ;get push data from B
0067   2A2700   lhld stkaddress ;get stack address
006A   77       mov m,a ;copy the data to stack
006B   23       inx h ;increment to new stack location
006C   222700   shld stkaddress ;save new stack top
006F   E1       pop h ;restore the h-l pair
0070   C1       pop b ;restore the b-c pair
0071   C9       ret ; job done OK bye
;**************************************************
;**************************************************
                org 2000h
2000   00       Array1:  db 00h  ; Storage for stack
2001   00       db 00h
2002   00       db 00h
2003   00       db 00h
2004   00       db 00h
2005   00       db 00h
2006   00       db 00h
2007   00       db 00h
2008   00       db 00h

2009   00       Array2:  db 00h  ; Storage for Array
200A   00       db 00h
200B   00       db 00h
200C   00       db 00h
200D   00       db 00h
200E   00       db 00h
200F   00       db 00h
2010   00       db 00h
2011   00       db 00h
;**************************************************
;**************************************************
;//**************************code concludes*****
```

Implementing Queue Data-Structure

Objective

To implement a Circular Queue which is an FIFO data-structure in 8085 assembly language. The queue will store bytes instead of words, and its size can be set by the user. It consists of following functions:

- **Queinit:** Initializes the queue. Set storage area to given location and length of the queue
- **Quepush:** Pushes the byte present in accumulator into the queue. Sets CY flag on queue overflow.
- **Quepop:** Pops a byte from the queue and places it in the accumulator. Sets CY flag on queue underflow.



Program [queue.asm]

```
;Queue Library
;Implements a Queue storage a user
;define area and of selected size
;(c) Vaibhav Jain
;date: 15/4/04
;********************************
 0000   31FFFF lxi sp,ffffh ;init stack
 0003   210020  lxi h ,2000h ;storage area
 0006   0E0A    mvi c,0Ah
;the storage size will provide only 9
 0008   CD2D00 call Queinit ;Init the queue
 000B   AF       xra a ;set acc to 0
 000C   4F       mov c,a
                 loop:
 000D   3C       inr a
 000E   3C       inr a
 000F   0C       inr c
 0010   CD6B00 call Quepush; push Acc in queue
 0013   D20D00 jnc loop; loop until error is signalled
 0016   AF       xra a;
 0017   4F       mov c,a;
 0018   211020 lxi h,array2; load address of array2
 001B   CD3E00           loop2: call Quepop
; fetch data in Acc
 001E   DA2700   jc exit ;if error signalled then exit
 0021   77       mov m,a ;save it in array2
 0022   23       inx h
 0023   0C       inr c
 0024   C31B00   jmp loop2

 0027   76       exit: hlt ;exit
;*************Queue Library********
;*********************************
;*******Queue Local Variables******
 0028   00   quelimit: db 00h ;max length of queue
 0029   00  quetop: db 00h ;top of the queue
;//********code continues************
```

Input

A		B		D		H	**20**
F		C	**0A**	E		L	**00**

S		Z		A		P		S	

Memory

2000	02	04	06	08	0A
2005	0C	0E	10	12	00
200A	00	00	00	00	00
200F	00	00	00	00	00
2014	00	00	00	00	00
2019	00	00	00	00	00
201E	00	00	00	00	00

Output

A	**12**	B	**00**	D	**00**	H	**20**
F	**44**	C	**09**	E	**00**	L	**19**

S	**0**	Z	**1**	A	**0**	P	**1**	S	**0**

Memory

2000	02	04	06	08	0A
2005	0C	0E	10	12	00
200A	00	00	00	00	00
200F	00	02	04	06	08
2014	0A	0C	0E	10	12
2019	00	00	00	00	00
201E	00	00	00	00	00

Implementing Queue Data-Structure

Program [queue.asm]

```
 002A   00       quebottom:      db 00h ;bottom of the queue
 002B   0000     queaddress:     dw 00h ;address of queue
;**********************************
;**********Queue Init*************
;Initializes the Queue with give data
;Input : H-L : Storage Address
; C   : No. of elements
;!!!Caution!!! Actual storage available is 1byte less
;Modifies: Nothing
;*********************************
         Queinit:
 002D   F5       push psw ;save Acc and Flags
 002E   79       mov a,c   ;copy count to Acc
 002F   322800 sta quelimit ;copy to memory variable
 0032   222B00 shld queaddress ;copy stack address to variable
 0035   AF       xra a ;set Acc to zero
 0036   322900 sta quetop ;set top=0
 0039   322A00   sta quebottom ;set bottom=0;
 003C   F1       pop psw ;restore acc and flags
 003D   C9       ret ;stack is now intialized
;******************************
;**********Queue Pop*************
;Pop an element from the Queue
;Input : nothing
;Modifies: Acc: containts poped element
; Cy: in case of Queue Underflow
;*********************************
Quepop:
 003E   C5       push b ;save B-C pair
 003F   4F       mov c,a ;save contents of A in C
 0040   3A2A00 lda quebottom ;get the Queue Bottom Index
 0043   47       mov b,a ;copy it to B
 0044   3A2900 lda quetop ;get the Queue Top index
 0047   B8       cmp b ;compare Top to Bottom
 0048   C24F00 jnz Quepopc2 ;Are they Equal?
 004B   79       mov a,c ;restore previous contents of A
 004C   C1       pop b ;this is the underflow error
 004D   37       stc ;set Cy to indicate Error
 004E   C9       ret ;Bye-Bye Sweet Heart!!!
Quepopc2:
 004F   47       mov b,a ;save top-count in B
 0050   E5       push h ;save h-l pair
 0051   2A2B00 lhld queaddress ;get the storage address in H-L
 0054   85       add l ;add this index to Lower Queue Address
 0055   6F       mov l,a   ;save this back to L
 0056   D25A00 jnc Quepopnc3 ;if nocarry skip these steps
 0059   24       inr h ;add this carry bit to H
Quepopnc3:
 005A   3A2800 lda quelimit ;get the queue length
 005D   04       inr b   ;increment value of top
 005E   B8       cmp b ;Is top=Queue Length
;//**************code continues******************
```

Implementing Queue Data-Structure

Program [queue.asm]

```
 005F   78      mov a,b ;NFC==> new top copied to Acc
 0060   C26400 jnz Quepopne3 ;if not skip following steps
 0063   AF      xra a ;set new top to zero
Quepopne3:
 0064   322900 sta quetop ;copy new queue top to mem-var
 0067   7E      mov a,m ;get the poped data from memory
 0068   E1      pop h ;restore h-l pair
 0069   C1      pop b ;restore b-c pair
 006A   C9      ret ;job is done Have a nice execution

;*******************************
;**********Queue Push*************
;Pushes an element into the Queue
;Input : Acc: data to push
;Modifies: Cy: in case of Queue Overflow
;*********************************
Quepush:
 006B   C5      push b ;save B-C pair
 006C   E5      push h ;save H-L pair
 006D   4F      mov c,a ;save push data in C
 006E   3A2A00 lda quebottom ;get the Queue Bottom Index
 0071   47      mov b,a ;copy bottom to B saved
 0072   2A2B00 lhld queaddress ;load the Queue Address
 0075   85      add l ;add Bottom index to L
 0076   6F      mov l,a ;NFC==>copy this address to L
 0077   D27B00 jnc Quepushnc1 ;if no carry from addition SKIP
 007A   24      inr h ;add this carry bit to H
Quepushnc1: ;address of bottom is in H-L
 007B   04      inr b ;to new value of bottom
 007C   3A2800 lda quelimit ;get the queue size
 007F   B8      cmp b ;if botttom=queuesize
 0080   C28500 jnz Quepushne2 ;SKIP if bottom!=size
 0083   0600     mvi b,0 ;set new bottom to 0
Quepushne2:
 0085   3A2900  lda quetop ;get the queue top
 0088   B8      cmp b ;if new bottom=top
 0089   C29100 jnz Quepushne3 ;SKIP if equal
 ;//The Queue has Over Flown RUN!!!!!
 008C   79      mov a,c ;restore push data back
 008D   E1       pop h ;restore H-L pair
 008E   C1      pop b ;restore B-C pair
 008F   37      stc ;signal Queue Overflow
 0090   C9      ret ;What a bad day- to Home
Quepushne3:
 0091   78      mov a,b   ;copy new bottom to Acc
 0092   322A00 sta quebottom ;store it to mem-var
 0095   79      mov a,c ;restore push data to acc
 0096   77      mov m,a   ;save Quity-Quity to memory
 0097   E1      pop h ;restore H-L pair
 0098   C1      pop b ;restore B-C pair
 0099   B7      ora a   ;reset the CY flag
 009A   C9      ret ;job is done Return home
;//*************code continues******************
```

Implementing Queue Data-Structure

Program [queue.asm]

```
;*******************************
;*******************************
                   org 2000h
 2000   0000     array: dw 0000h; Array of 10 bytes for queue
 2002   0000     dw 0000h;
 2004   0000     dw 0000h;
 2006   0000     dw 0000h;
 2008   0000     dw 0000h;
                 org 2010h
 2010   0000     array2: dw 0000; Array of 9 bytes for data
 2012   0000     dw 00;
 2014   0000     dw 00;
 2016   0000     dw 00;
 2018   00       db 00;
;//**************code concludes******************
 SYMBOL TABLE :

LOOP            000D
LOOP2           001B
EXIT            0027
QUELIMIT        0028
QUETOP          0029
QUEBOTTOM       002A
QUEADDRESS      002B
QUEINIT         002D
QUEPOP          003E
QUEPOPC2        004F
QUEPOPNC3       005A
QUEPOPNE3       0064
QUEPUSH         006B
QUEPUSHNC1      007B
QUEPUSHNE2      0085
QUEPUSHNE3      0091
ARRAY           2000
ARRAY2          2010
```