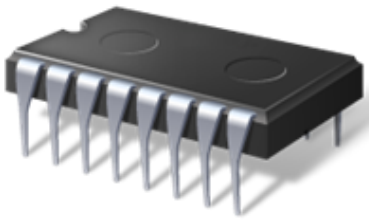


8085 simulator

V 1.0

Development Platform: Microsoft visual basic 2010

System requirements

Operating system : Windows XP (sp3), Windows Vista, Windows 7 32 and 64 bit editions

Minimum Screen resolution : 1024x768

.net frame work : 4.0.3210 (Included with the package)

Installation Notes

The 8085 simulator is devolved with Microsoft visual studio 2010 since it requires .net framework 4.0.3210 version installed on your system, if your system do not contain that version the corresponding version will installed before the installation of 8085 Simulator starts. For windows xp to install .net framework it is necessary to have windows installer is installed on you system that is if the system do not contain windows installer it need to be installed. So the installation of windows xp requires at least 2 restarts.

8085 Simulator User Interface Guide

8085 simple simulator

It allows you to enter into simple simulation of your Micro 85-EB training kit.it having a text of like 7 segment display and RES and INT Buttons

8085 Advanced simulator

It allows you to enter into advanced simulation in which you can enter instructions, controls the execution speed and viewing the memeory and registry values etc. Advanced simulator programs can be saved into a file with 8085 extension.

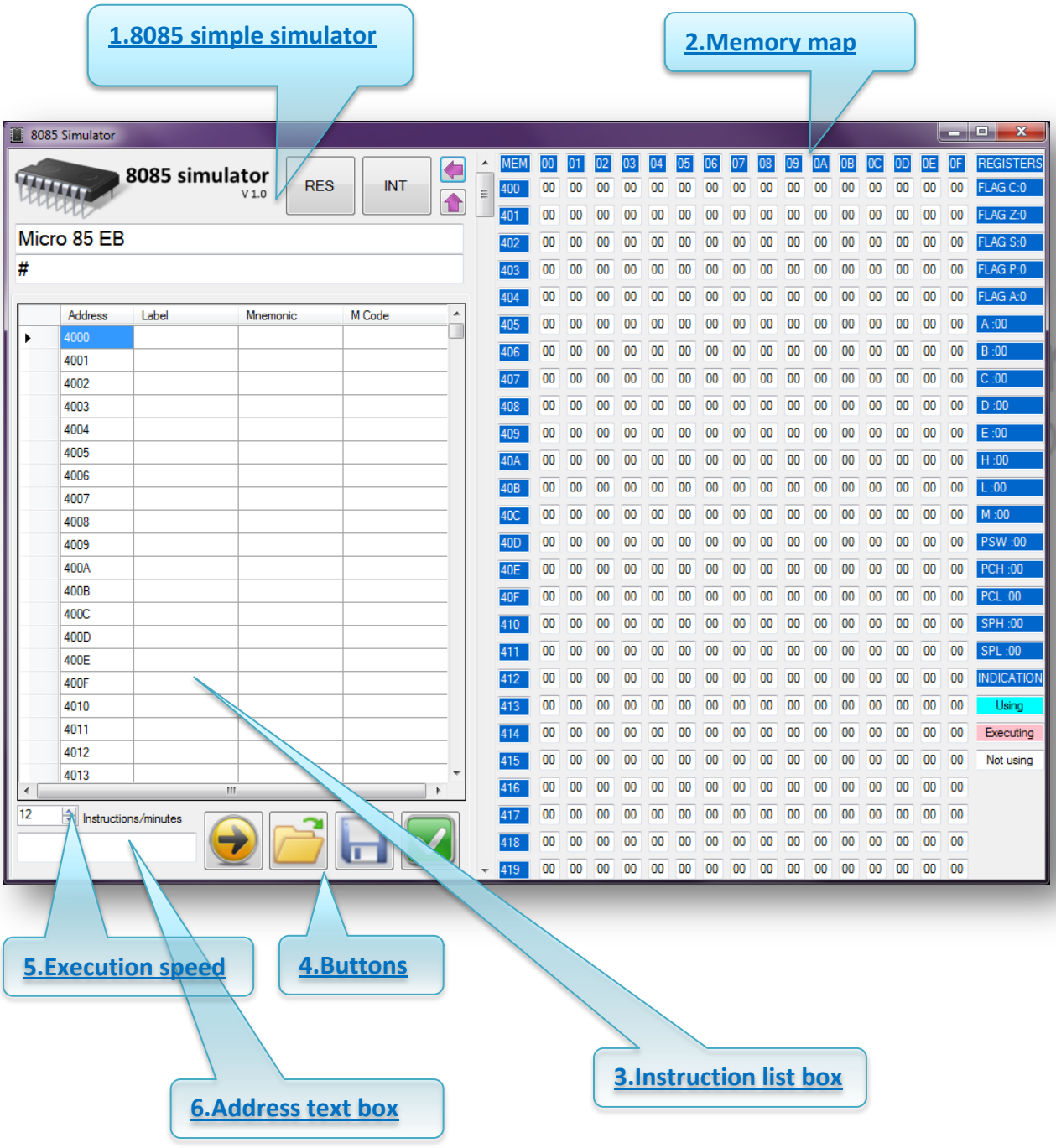
Settings

Setting window will allow you to change the default values of address range of memory and execution speed

Help

In this window you can view the available help topics and external links

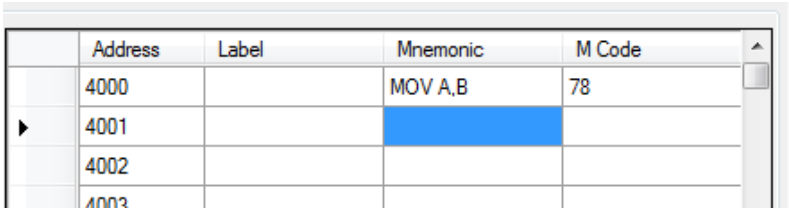
Advanced Simulator



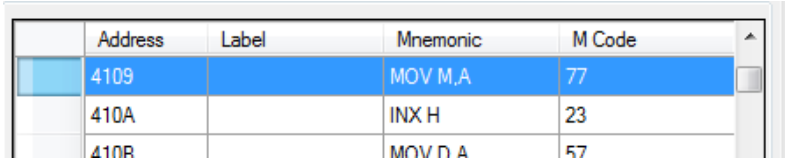
1. 8085 simple simulator
   In 8085 you can enter instructions like a Micro-85 EB training kit. And RES and INT buttons for the corresponding functions
2. Memory map
   In memory map you can view current value of memory locations and can edit values in it. you can use vertical scrollbar to access all memory locations
3. Instruction list
   In instruction list box you can edit label column and mnemonic column .you can enter mnemonic instructions with operands. Mnemonic can contain labels .when entering mnemonic with labels please make sure that the label it is already defined. E.g. “JMP label”
4. Control buttons
   There are following buttons used for specific purpose
   - “Execute”-Which will start execute from address given in address text box
   - “Open file”-Opens a file with 8085 extension and loaded into simulator
   - “Save file”-Saves the current values of memory and registry into a file with 8085 extension
   - “Clear all”-Clears the values of memory and registers
5. Execution speed
   In this you can change the execution speed. Execution speed is given in Instructions per seconds
6. Address text box
   When a valid address is entered in to this text box the instruction list box and memory map will focused into that address.

How I do it?

- How I enter an instruction?
  You can enter mnemonic instructions in the mnemonic column by clicking oneach cells . after entering a instruction when enter key is pressed the corresponding mechene code is loded into mechene code column

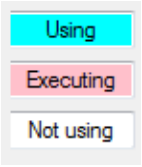


- How I determine currently executing instruction ?
  It can done in two ways
  1. using instruction list box selected row will be the currently executing instruction



  2. Using memory map the memory location given in pink color is the currently executing instruction

- What are the indicators used in memory map?



- How I start execution?
  This can be done in two ways
  1. Using simple simulator instruction G <starting address>
  2. By using start execution button
- Can I have the options to edit the register values?
  No you can't edit the values of registers
- What are the details stored to file
  The .8085 file contain values of registers , memory, execution speed, memory address range etc.
- Where I search for updates and support
  For updates: http://jtechsoftwares.wetpaint.com/page/8085+Simulator
  For support: http://jtechsoftwares.wetpaint.com/page/Help

Before Use

Please read following instructions before use

1.8085 Simulator is do not defined following instructions

RST –restart instructions
INT-interrupt instructions
OUT, IN –instructions

2. Do not consider simulator as the final decisions. I make simulator to avoid maximum errors as possible. If you find an error please info me through mail.

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description
ADD	**R**	Add register or memory, to	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of

	M	accumulator	the addition. **Example: ADD B or ADD M**
ADC	**R** **M**	Add register to accumulator with carry	The contents of the operand (register or memory) and M the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. **Example: ADC B or ADC M**
ADI	**8-bit data**	Add immediate to accumulator	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. **Example: ADI 45H**
ACI	**8-bit data**	Add immediate to accumulator with carry	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. **Example: ACI 45H**
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand. **Example: LXI H, 2034H or LXI H, XYZ**
DAD	**Reg. pair**	Add register pair to H and L registers	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. **Example: DAD H**
SUB	**R** **M**	Subtract register or memory from accumulator	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example: SUB B or SUB M**
SBB	**R** **M**	Subtract source and borrow from accumulator	The contents of the operand (register or memory) and M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example: SBB B or SBB M**
SUI	**8-bit data**	Subtract immediate from accumulator	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. **Example: SUI 45H**
SBI	**8-bit data**	Subtract immediate from accumulator with borrow	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. **Example: XCHG**
INR	**R**	Increment register	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory

	M	or memory by 1	location, its location is specified by the contents of the HL registers. **Example: INR B or INR M**
INX	**R**	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. **Example: INX H**
DCR	**R** **M**	Decrement register or memory by 1	The contents of the designated register or memory are M decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example: DCR B or DCR M**
DCX	**R**	Decrement register pair by 1	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. **Example: DCX H**
DAA	**none**	Decimal adjust accumulator	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. **Example: DAA**

BRANCHING INSTRUCTIONS

<table>
<tr><th colspan="3">Opcode</th><th>Operand</th><th>Explanation of Instruction</th><th>Description</th></tr>
<tr><td colspan="3">JMP</td><td>16-bit address</td><td>Jump unconditionally</td><td>The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

Example: JMP 2034H or JMP XYZ</td></tr>
<tr><td colspan="3"><table><tr><td>Opcode</td><td>Description</td><td>Flag Status</td></tr><tr><td>JC</td><td>Jump on Carry</td><td>CY = 1</td></tr><tr><td>JNC</td><td>Jump on no Carry</td><td>CY = 0</td></tr><tr><td>JP</td><td>Jump on positive</td><td>S = 0</td></tr></table></td><td>16-bit address</td><td>Jump conditionally</td><td>The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.

Example: JZ 2034H or JZ XYZ</td></tr>
</table>

JM	**Jump on minus**	**S = 1**			
JZ	**Jump on zero**	**Z = 1**			
JNZ	**Jump on no zero**	**Z = 0**			
JPE	**Jump on parity even**	**P = 1**			
JPO	**Jump on parity odd**	**P = 0**			
Opcode	Description	Flag Status	**16-bit address**	Unconditional subroutine call	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. **Example: CALL 2034H or CALL XYZ**
CC	**Call on Carry**	**CY = 1**			
CNC	**Call on no Carry**	**CY = 0**			
CP	**Call on positive**	**S = 0**			
CM	**Call on minus**	**S = 1**			
CZ	**Call on zero**	**Z = 1**			
CNZ	**Call on no zero**	**Z = 0**			
CPE	**Call on parity even**	**P = 1**			
CPO	**Call on parity odd**	**P = 0**			
RET			**none**	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter,and program execution begins at the new address. **Example: RET**
Opcode	Description	Flag Status	**none**	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. **Example: RZ**
RC	**Return on Carry**	**CY = 1**			
RNC	**Return on no Carry**	**CY = 0**			
RP	**Return on positive**	**S = 0**			
RM	**Return on minus**	**S = 1**			
RZ	**Return on zero**	**Z = 1**			

<table>
<tr><td><table>
<tr><td>RNZ</td><td>Return on no zero</td><td>Z = 0</td></tr>
<tr><td>RPE</td><td>Return on parity even</td><td>P = 1</td></tr>
<tr><td>RPO</td><td>Return on parity odd</td><td>P = 0</td></tr>
</table></td><td></td><td></td><td></td></tr>
<tr><td>PCHL</td><td>none</td><td>Load program counter with HL contents</td><td>The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example: PCHL</td></tr>
<tr><td>RST</td><td>0-7</td><td>Restart</td><td>The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

<table><tr><th>Instruction</th><th>Restart Address</th></tr><tr><td>RST 0</td><td>0000H</td></tr><tr><td>RST1</td><td>0008H</td></tr><tr><td>RST 2</td><td>0010H</td></tr><tr><td>RST 3</td><td>0018H</td></tr><tr><td>RST 4</td><td>0020H</td></tr><tr><td>RST 5</td><td>0028H</td></tr><tr><td>RST 6</td><td>0030H</td></tr><tr><td>RST 7</td><td>0038H</td></tr></table>
The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:</td></tr>
</table>

				Interrupt / Restart Address **TRAP** **0024H** **RST 5.5** **002CH** **RST 6.5** **0034H** **RST 7.5** **003CH**

CONTROL INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description
NOP	**none**	No operation	No operation is performed. The instruction is fetched and decoded. However no operation is executed. **Example: NOP**
HLT	**none**	Halt and enter wait state	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. **Example: HLT**
DI	**none**	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. **Example: DI**
EI	**none**	Enable interrupts	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flipflop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). **Example: EI**
RIM	**none**	Read interrupt mas	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations. **Example: RIM**
SIM	**none**	Set interrupt mask	

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Explanation of	Description

		Instruction	
MOV	**Rd, Rs** **M, Rs** **Rd, M**	Copy from source(Rs) to destination(Rd)	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. **Example: MOV B, C or MOV B, M**
MVI	**Rd, data** **M, data**	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example: MVI B, 57H or MVI M, 57H**
LDA	**16-bit address**	Load accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. **Example: LDA 2034H**
LDAX	**B/D Reg. pair**	Load accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. **Example: LDAX B**
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand. **Example: LXI H, 2034H or LXI H, XYZ**
LHLD	**16-bit address**	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. **Example: LHLD 2040H**
STA	**16-bit address**	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. **Example: STA 4350H**
STAX	**Reg. pair**	Store accumulator indirect	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. **Example: STAX B**
SHLD	**16-bit address**	Store H and L registers direct	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. **Example: SHLD 2470H**
XCHG	**none**	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register

			E. **Example: XCHG**
SPHL	**none**	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. **Example: SPHL**
XTHL	**none**	Exchange H and L with top of stack	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. **Example: XTHL**
PUSH	**Reg. pair**	Push register pair onto stack	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. **Example: PUSH B or PUSH A**
POP	**Reg. pair**	Pop off stack to register pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. **Example: POP H or POP A**
OUT	**8-bit port address**	Output data from accumulator to a port with 8-bit address	The contents of the accumulator are copied into the I/O port specified by the operand. **Example: OUT F8H**
IN	**8-bit port address**	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the accumulator. **Example: IN 8CH**

LOGICAL INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description

CMP	**R**
	
	M
		Compare register or memory with accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < (reg/mem): carry flag is set
			if (A) = (reg/mem): zero flag is set
			if (A) > (reg/mem): carry and zero flags are reset
			
			Example: CMP B or CMP M

CPI	**8-bit data**	Compare immediate with accumulator	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < data: carry flag is set
			if (A) = data: zero flag is set
			if (A) > data: carry and zero flags are reset
			
			Example: CPI 89H

ANA	**R** **M**	Logical AND register or memory with accumulator	The contents of the accumulator are logically ANDed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. **Example: ANA B or ANA M**
ANI	**8-bit data**	Logical AND immediate with accumulator	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. **Example: ANI 86H**
XRA	**R** **M**	Exclusive OR register or memory with accumulator	The contents of the accumulator are Exclusive ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: XRA B or XRA M**
XRI	**8-bit data**	Exclusive OR immediate with accumulator	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: XRI 86H**
ORA	**R** **M**	Logical OR register or memory with accumulator	The contents of the accumulator are logically ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: ORA B or ORA M**
ORI	**8-bit data**	Logical OR immediate with accumulator	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: ORI 86H**
RLC	**none**	Rotate accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected. **Example: RLC**
RRC	**none**	Rotate accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. **Example: RRC**

RAL	**none**	Rotate accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. **Example: RAL**
RAR	**none**	Rotate accumulator right through carry	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. **Example: RAR**
CMA	**none**	Complement accumulator	The contents of the accumulator are complemented. No flags are affected. **Example: CMA**
CMC	**none**	Complement carry	The Carry flag is complemented. No other flags are affected. **Example: CMC**
STC	**none**	Set Carry	Set Carry **Example: STC**