

8085 Instruction Set Reference Guide

The 8085 instruction set is classified into five categories based on the function they perform. The processor uses an 8-bit Data Bus and a 16-bit Address Bus, which dictates the size of its registers and memory operations.

1. Data Transfer Instructions

These instructions move data between registers, between a register and memory, or load immediate data. Note: These operations do not affect the flags (except for the flags being affected by data loaded from memory).

a) MOV

MOV Rd, Rs: Move 8-bit data from Source Register (Rs) to Destination Register (Rd).

MOV Rd, M: Move 8-bit data from Memory (M, addressed by HL) to Register (Rd).

MOV M, Rs: Move 8-bit data from Register (Rs) to Memory (M, addressed by HL).

b) MVI

MVI R, Data: Move 8-bit Immediate Data into Register R.

MVI M, Data: Move 8-bit Immediate Data into Memory (M, addressed by HL).

c) LXI

LXI Rp, Data16: Load 16-bit Immediate Data into a Register Pair (BC, DE, or HL) or the Stack Pointer (SP).

d) LDA

LDA Addr16: Load Accumulator with the 8-bit content of the memory address specified by 16-bit operand.

e) STA

STA Addr16: Store Accumulator content into the memory address specified by 16-bit operand.

f) LHLD

LHLD Addr16: Load H and L registers from memory. (L gets content of Addr16, H gets content of Addr16 + 1).

g) SHLD

SHLD Addr16: Store H and L registers into memory. (L stored at Addr16, H stored at Addr16 + 1).

h) XCHG

XCHG: Exchange the contents of the HL pair with the DE pair.

i) SPHL

SPHL: Move the content of the HL register pair to the Stack Pointer (SP).

j) PUSH

PUSH Rp: Push the content of the register pair (BC, DE, HL, or PSW) onto the stack. Decrements SP by 2.

k) POP

POP Rp

Pop 16-bit data from the stack into the specified register pair. Increments SP by 2.

2. Arithmetic Instructions

These instructions perform addition, subtraction, increment, and decrement operations. They all affect the flags (Zero, Carry, Sign, Parity, Aux. Carry).

a) ADD

ADD R

Add Register (R) content to Accumulator (A). Result in A.

ADD M

Add Memory (M, addressed by HL) content to Accumulator (A). Result in A.

b) ADC

ADC R

Add Register (R) content and Carry Flag to Accumulator (A). Result in A.

ADC M

Add Memory (M) content and Carry Flag to Accumulator (A). Result in A.

c) ADI

ADI Data

Add 8-bit Immediate Data to Accumulator (A). Result in A.

d) ACI

ACI Data

Add 8-bit Immediate Data and Carry Flag to Accumulator (A). Result in A.

e) SUB

SUB R

Subtract Register (R) content from Accumulator (A). Result in A.

SUB M

Subtract Memory (M) content from Accumulator (A). Result in A.

f) SBB

SBB R

Subtract Register (R) content and Borrow (Carry Flag) from Accumulator (A). Result in A.

SBB M

Subtract Memory (M) content and Borrow (Carry Flag) from Accumulator (A). Result in A.

g) SUI

SUI Data

Subtract 8-bit Immediate Data from Accumulator (A). Result in A.

h) SBI

SBI Data

Subtract 8-bit Immediate Data and Borrow (Carry Flag) from Accumulator (A). Result in A.

i) INR

INR R

Increment the content of Register (R) by 1. (Affects all flags except Carry).

INR M

Increment the content of Memory (M) by 1. (Affects all flags except Carry).

j) DCR

DCR R

Decrement the content of Register (R) by 1. (Affects all flags except Carry).

DCR M

Decrement the content of Memory (M) by 1. (Affects all flags except Carry).

k) INX

INX Rp

Increment the content of the Register Pair (Rp) by 1. (Does NOT affect any flags).

l) DCX

DCX Rp

Decrement the content of the Register Pair (Rp) by 1. (Does NOT affect any flags).

m) DAD

DAD Rp

Double Addition: Add the content of the Register Pair (Rp) to the HL pair.

Result in HL. (Only affects Carry flag).

n) DAA

DAA

Decimal Adjust Accumulator. Converts the binary content of A into two BCD digits after an addition operation.

2. Logical Instructions

These instructions perform Boolean operations (AND, OR, XOR), compare, and rotate bits. They all affect the flags.

a) ANA

ANA R

Logical AND Register (R) with Accumulator (A). Result in A. (Zero flag affected, Carry flag always reset).

ANA M

Logical AND Memory (M) with Accumulator (A). Result in A.

b) ANI

ANI Data

Logical AND 8-bit Immediate Data with Accumulator (A). Result in A.

c) ORA

ORA R

Logical OR Register (R) with Accumulator (A). Result in A. (Carry flag always reset).

ORA M

Logical OR Memory (M) with Accumulator (A). Result in A.

d) ORI

ORI Data

Logical OR 8-bit Immediate Data with Accumulator (A). Result in A.

e) XRA

XRA R

Logical Exclusive-OR Register (R) with Accumulator (A). Result in A. (Carry flag always reset).

XRA M

Logical Exclusive-OR Memory (M) with Accumulator (A). Result in A.

f) XRI

XRI Data

Logical Exclusive-OR 8-bit Immediate Data with Accumulator (A). Result in A.

g) CMP

CMP R

Compare Register (R) with Accumulator (A) by subtraction ($A - R$) but without storing the result. Flags are set based on the comparison (e.g., Z=1 if $A=R$, C=1 if $A < R$).

CMP M

Compare Memory (M) with Accumulator (A).

h) CPI

CPI Data

Compare 8-bit Immediate Data with Accumulator (A).

i) RLC

RLC

Rotate Accumulator Left by 1 bit. Bit D7 moves to Carry and D0.

j) RRC

RRC

Rotate Accumulator Right by 1 bit. Bit D0 moves to Carry and D7.

k) RAL

RAL

Rotate Accumulator Left through Carry. Bit D7 moves to Carry, Carry moves to D0.

l) RAR

RAR

Rotate Accumulator Right through Carry. Bit D0 moves to Carry, Carry moves to D7.

m) CMA

CMA

Complement Accumulator (1's complement). (Does NOT affect flags).

n) STC

STC

Set Carry Flag (C=1). (Does NOT affect other flags).

o) CMC

CMC

Complement Carry Flag (C = NOT C). (Does NOT affect other flags).

3. Branching/Control Instructions

These instructions change the normal sequential flow of program execution by changing the content of the Program Counter (PC).

a) JMP

JMP Addr16:

Unconditional Jump: Jump to the 16-bit address specified.

J condition Addr16:

Conditional Jumps

Jumps to the specified address only if the specified flag condition is met.

b) JZ/JNZ

Jump if Zero / Jump if Not Zero

c) JC/JNC

Jump if Carry / Jump if No Carry

d) JP/JM

Jump if Plus (Sign=0) / Jump if Minus (Sign=1)

e) JPE/JPO

Jump if Parity Even / Jump if Parity Odd

f) CALL

CALL Addr16

Unconditional Call: Jumps to a subroutine at the specified address. The return address (PC+3) is pushed onto the stack.

Conditional Calls

C condition Addr16

Calls the subroutine only if the specified flag condition is met. (e.g., CZ, CNZ, CC, CNC, etc.)

g) RET

RET

Unconditional Return: Returns from a subroutine. The return address is popped from the stack into the PC.

Conditional Returns

R condition

Returns from a subroutine only if the specified flag condition is met. (e.g., RZ, RNZ, RC, RNC, etc.)

h) RST

RST n

Restart (Interrupt): Transfers program control to one of eight memory locations (0000H, 0008H, ..., 0038H). Similar to a CALL to a fixed address.

i) PCHL

PCHL

Program Counter Load: Load the content of the HL pair into the Program Counter (PC).

4. Machine Control and I/O Instructions

These instructions control the processor, handle external device communication, and perform interrupt management.

a) IN

IN Port8

Read 8-bit data from the I/O port specified by the 8-bit address into the Accumulator (A).

b) OUT

OUT Port8

Write 8-bit content of the Accumulator (A) to the I/O port specified by the 8-bit address.

c) HLT

HLT

Halt/Stop processing and wait for an interrupt or hardware reset.

d) NOP

NOP

No Operation. The instruction is fetched and decoded but no operation is executed. Used for timing delays.

e) EI

EI

Enable Interrupts. Sets the Interrupt Enable Flip-Flop.

f) DI

DI

Disable Interrupts. Resets the Interrupt Enable Flip-Flop.

g) RIM

RIM

Read Interrupt Mask. Loads the Accumulator with the current interrupt masks and pending interrupts.

h) SIM

SIM

Set Interrupt Mask. Used to set the interrupt masks and perform serial output.

END

“In C, You can shoot yourself in the foot. In Assembly, you build the gun, manufacture the bullet , and then carefully calculate the trajectory to blow your entire leg off efficiently.”

— Bjarne Stroustrup

Creator of C++

Prepared with ❤ by Mutugi Gakenge