

CPU1 Instruction Set

CPU1 Instruction Set

MNEMONIC	Opcode	Binary	Operand	Operation	Description
ADR	80	10000000	XRRRYRRR	$RX \leftarrow RX + RY$	Add register: RY to RX and store back in RX
ADI	88, 89, 8A, 8B, 8C, 8D, 8E, 8F	10001RRR	VVVVVVVV	$R \leftarrow R + (PC+1)$	Add immediate: Add byte at PC+1 to contents of R and store in R.
ADM	90, 91, 92, 93, 94, 95, 96, 97	10010RRR	MMMMMMMM MMMMMMMM	$R \leftarrow R + (M)$	Add memory: Add contents of memory at address M to contents of R and store in R
ADRC	81	10000001	XRRRYRRR	$RX \leftarrow RX + RY + C$	Add registers with carry: Add RX to RY and the Carry bit (flag). Result in RX.
ADIC	A0, A1, A2, A3, A4, A5, A6, A7	10100RRR	VVVVVVVV	$R \leftarrow R + (PC+1) + C$	Add w/carry immediate: Add content of register R to byte immediately following opcode plus the carry bit (flag).
ADMC	A8, A9, AA, AB, AC, AD, AE, AF	10101RRR	MMMMMMMM MMMMMMMM	$R \leftarrow R + (M) + C$	Add memory w/carry: Add content of register R with byte at address M plus carry bit.
SUB	82	10000010	XRRRYRRR	$RX \leftarrow RX - RY$	Subtract: Subtract contents of RY from RX and store back in RX. Set Carry and Negative flags as appropriate.
SUBI	B8, B9, BA, BB, BC, BD, BE, BF	10111RRR	VVVVVVVV	$R \leftarrow R - (PC+1)$	Subtract immediate: Subtract the contents of byte following the opcode from contents of R and store back in R. Set Carry and Negative flags as appropriate.
SUBM	C0, C1, C2, C3, C4, C5, C6, C7	11000RRR	MMMMMMMM MMMMMMMM	$R \leftarrow R - (M)$	Subtract memory: Subtract the byte at address M from the content of R and store back in R. Set Carry and Negative flags as appropriate.
SUBC	83	10000011	XRRRYRRR	$RX \leftarrow RX - RY - C - (\text{NOT } C)$	Subtract register w/borrow: Subtract RY from RX and subtract carry bit from that. Result to RX.
SUBIC	D0, D1, D2, D3, D4, D5, D6, D7	11010RRR	VVVVVVVV	$R \leftarrow R - (PC+1) - C - (\text{NOT } C)$	Subtract immediate w/borrow: Subtract byte following opcode from R. Subtract carry bit from that and store in R.
SUBMC	D8, D9, DA, DB, DC, DD, DE, DF	11011RRR	MMMMMMMM MMMMMMMM	$R \leftarrow R - (M) - C - (\text{NOT } C)$	Subtract immediate memory w/borrow: Subtract byte at address M from R. Subtract carry bit. Store result in R.
LDI	E0, E1, E2, E3, E4, E5, E6, E7	11100RRR	VVVVVVVV	$R \leftarrow (PC+1)$	Load immediate: Store the byte immediately after the opcode in register R
STI	E8, E9, EA, EB, EC, ED, EE, EF	11101RRR	MMMMMMMM MMMMMMMM	$(M) \leftarrow R$	Store immediate: Store the content of R at address M
LDM	F0, F1, F2, F3, F4, F5, F6, F7	11110RRR	MMMMMMMM MMMMMMMM	$R \leftarrow (M)$	Load memory: Store the byte at address M into register R
EX	84	10000100	XRRRYRRR	$RX \leftarrow RY; RY \leftarrow RX$	Exchange registers: Swap content of registers RX and RY
CMP	85	10000101	XRRRYRRR	IF $RX = RY$, CP=true, else CP=false	Compare register RX with RY and set compare flag true if equal
AND	86	10000110	XRRRYRRR	$RX \leftarrow RX \text{ AND } RY$	AND: Logical AND of RX and RY. Result into RX.
OR	87	10000111	XRRRYRRR	$RX \leftarrow RX \text{ OR } RY$	OR: Logical OR of RX and RY. Result into RX.
XOR	19	00011001	XRRRYRRR	$RX \leftarrow RX \text{ XOR } RY$	XOR: Exclusive OR of RX and RY. Result into RX.
ANI	50, 51, 52, 53, 54, 55, 56, 57	01010RRR	VVVVVVVV	$R \leftarrow R \text{ AND } (PC+1)$	AND immediate: Logical AND of R with byte immediately following the opcode.

CPU1 Instruction Set

MNEMONIC	Opcode	Binary	Operand	Operation	Description
ORI	58, 59, 5A, 5B, 5C, 5D, 5E, 5F	01011RRR	VVVVVVVV	R <- R OR (PC+1)	OR immediate: Logical OR of R with byte immediately following the opcode.
XRI	60, 61, 62, 63, 64, 65, 66, 67	01100RRR	VVVVVVVV	R <- R XOR (PC+1)	XOR immediate: Logical XOR of R with byte immediately following the opcode.
SHR	68, 69, 6A, 6B, 6C, 6D, 6E, 6F	01101RRR		R <- R >> 1	Shift right: Shift register R right one bit. Fill with 0 on left.
SHRC	70, 71, 72, 73, 74, 75, 76, 77	01110RRR		R <- R >> 1	Shift right with carry: Shift register R right one bit and fill most significant bit with carry bit.
SHL	78, 79, 7A, 7B, 7C, 7D, 7E, 7F	01111RRR		R <- R << 1	Shift left: Shift register R left one bit. Fill least significant bit with 0.
SHLC	20, 21, 22, 23, 24, 25, 26, 27	00100RRR		R <- R << 1	Shift left w/carry: Shift register R left one bit. Fill least significant bit with carry bit.
INC	28, 29, 2A, 2B, 2C, 2D, 2E, 2F	00101RRR		R <- R + 1	Increment: Increment register R by 1.
DEC	30, 31, 32, 33, 34, 35, 36, 37	00110RRR		R <- R -1	Decrement: Decrement register R by 1.
NOP	00	00000000		PC <- PC + 1	NOP: continue to next instruction by incrementing PC by one.
HALT	01	00000001		PC <- PC	HALT: Stop CPU clock and instruction execution at current PC.
SETQ	38, 39, 3A, 3B, 3C, 3D, 3E, 3F	00111QQQ		QN <- true (1)	Set QN: Sets specified I/O line to true (1)
RESETQ	10, 11, 12, 13, 14, 15, 16, 17	00010QQQ		QN <- false (0)	Resets QN: Sets specified I/O line to false (0)
LBRC	18	00011000	MMMMMMMM MMMMMMMM	IF CP=true, PC <- M, else PC <- PC + 2	Long branch: If compare bit is true (1), jump to address M. Otherwise, increment PC to next instruction.
LBRQ	08, 09, 0A, 0B, 0C, 0D, 0E, 0F	00001QQQ	MMMMMMMM MMMMMMMM	if QN, PC <- M, else PC <- PC + 2	Branch if QN: Jump to address M if QN is true, otherwise, skip to next instruction.
PUSH	40, 41, 42, 43, 44, 45, 46, 47	01000RRR		SP <- SP -1; (SP) <- R	Push register: First decrement SP by one byte. Then store R at top of stack.
POP	48, 49, 4A, 4B, 4C, 4D, 4E, 4F	01001RRR		R <- (SP)	Pop register: Load R with byte at top of stack, then increment SP to new top.
CALL	02	00000010	MMMMMMMM MMMMMMMM	PC <- PC + 3; SP <- SP - 1; (SP) <- PC	Call subroutine: Increment PC to point past the next two bytes to the return address. Decrement SP to new top. Store least significant byte of PC on stack. Then. decrement SP again and store most significant byte of PC on stack. SP now points to most significant byte of the return location.
RET	03	00000011		PC.1 <- (SP); SP + 1; PC.0 <- (SP); SP + 1	Return from subroutine: Places byte at top of stack into most significant byte of PC. Increments SP and places next byte into least significant byte of PC. Finally, increment SP to top of stack.