

## Instruction Set

Instruction	Description	Function
<b>SPECIAL INSTRUCTIONS</b>		
HALT	The processor idles indefinitely	Do nothing indefinitely
NOP	The processor stalls for one clock cycle.	Do nothing for one clock cycle
<b>MOVE INSTRUCTIONS</b>		
MOV rA {k, const}	Move literal k to register rA	rA <= k
MOV rA rB	Move the value in register rB to register rA	rA <= rB
MOVE rA {k, const}	Move literal k to register rA if the ZERO flag is set.	If ZERO = 1, rA <= k
MOVE rA rB	Move the value in register rB to register rA if the ZERO flag is set.	If ZERO = 1, rA <= rB
MOVNE rA {k , const}	Move literal k to register rA if the ZERO flag is not set.	If ZERO = 0, rA <= k
MOVNE rA rB	Move the value in register rB to register rA if the ZERO flag is not set.	If ZERO = 0, rA <= rB
MOVC rA {k, const}	Move literal k to register rA if the CARRY flag is set.	If CARRY = 1, rA <= k
MOVC rA rB	Move the value in register rB to register rA if the CARRY flag is set.	If CARRY = 1, rA <= rB

MOVNC rA {k, const}	Move literal k to register rA if the CARRY flag is not set.	If CARRY = 0, rA <= k
MOVNC rA rB	Move the value in register rB to register rA if the CARRY flag is not set.	If CARRY = 0, rA <= rB
MOVL rA {k, const}	Move literal k to register rA if the SIGN flag is set.	If SIGN = 1, rA <= k
MOVL rA rB	Move the value in register rB to register rA if the SIGN flag is set.	If SIGN = 1, rA <= rB
MOVG rA {k, const}	Move literal k to register rA if the SIGN flag is not set.	If SIGN = 0, rA <= k
MOVG rA rB	Move the value in register rB to register rA if the SIGN flag is not set.	If SIGN = 0, rA <= rB
<b>ALU INSTRUCTIONS</b>		
ADD rA {k, const}	Add register rA with literal k	rA <= rA + k
ADD rA rB	Add register rA with register rB	rA <= rA + rB
ADDC rA {k, const}	Add register with literal k and CARRY	rA <= rA + k + CARRY
ADDC rA rB	Add register rA with register rB and CARRY	rA <= rA + rB + CARRY
AND rA {k, const}	And register rA with literal k	rA <= rA AND k
AND rA rB	And register rA with register rB	rA <= rA AND rB

CMP rA {k, const}	Compare register rA with literal k. Set CARRY and ZERO flags. Registers are unaffected.	If rA = k, ZERO <= 1 If rA < k, SIGN <= 1 If overflow or underflow, CARRY <= 1
CMP rA rB	Compare register rA with register rB. Set CARRY and ZERO flags. Registers are unaffected.	If rA = rB, ZERO <= 1 If rA < rB, SIGN <= 1 If overflow or underflow, CARRY <= 1
MVN rA {k, const}	Invert the bits of k and move the result to register rA	rA <= NOT k
MVN rA rB	Invert the bits of the value stored in register rB and move the result to register rA.	rA <= NOT rB
NOT rA {k, const}	Bitwise NOT register rA with literal k.	rA <= rA NOT k
NOT rA rB	Bitwise NOT register rA with register rB.	rA <= rA NOT rB
OR rA {k, const}	OR register rA with literal k.	rA <= rA OR k
OR rA rB	OR register rA with register rB.	rA <= rA OR rB
RL rA	Rotate register rA left.	rA <= {rA[6:0], rA[7]} CARRY <= rA[7]
RR rA	Rotate register rA right.	rA <= {rA[0], rA[7:1]} CARRY <= rA[0]
SL0 rA	Shift rA left, zero fill	rA <= {rA[6:0], 0} CARRY <= rA[7]
SL1 rA	Shift rA left, one fill	rA <= {rA[6:0], 1} CARRY <= rA[7]
SR0 rA	Shift rA right, zero fill	rA <= {0, rA[7:1]} CARRY <= rA[0]

SR1 rA	Shift rA right, one fill	$rA \leq \{1, rA[7:1]\}$ CARRY $\leq rA[0]$
SUB rA {k, const}	Subtract register rA from literal k	$rA \leq rA - k$
SUB rA rB	Subtract register rA from register rB	$rA \leq rA - rB$
SUBC rA {k, const}	Subtract register rA from literal k and CARRY	$rA \leq rA - k - \text{CARRY}$
SUBC rA rB	Subtract register rA from register rB and CARRY	$rA \leq rA - rB - \text{CARRY}$
TEST rA {k, const}	Test bits in register rA against literal k. Update CARRY and ZERO flags. Registers are unaffected.	If $(rA \text{ AND } k) = 0$ , ZERO $\leq 1$ CARRY $\leq$ odd parity of $(rA \text{ AND } k)[7]$ SIGN $\leq (rA \text{ AND } k)[7]$
TEST rA rB	Test bits in register rA against register rB. Update CARRY and ZERO. Registers are unaffected.	If $(rA \text{ AND } rB) = 0$ , ZERO $\leq 1$ CARRY $\leq$ odd parity of $(rA \text{ AND } rB)[7]$ SIGN $\leq (rA \text{ AND } rB)[7]$
XOR rA {k, const}	Bitwise XOR register rA with literal k.	$rA \leq rA \text{ XOR } k$
XOR rA rB	Bitwise XOR register rA with register rB	$rA \leq rA \text{ XOR } rB$
<b>BRANCHING INSTRUCTIONS</b>		
JMP {k, const, label}	Jump to the address k	$PC \leq k$
JE {k, const, label}	Jump to the address k if the ZERO flag is set.	If ZERO = 1, $PC \leq k$
JNE {k, const, label}	Jump to the address k if the ZERO flag is not set.	If ZERO = 0, $PC \leq k$

JL {k, const, label}	Jump to the address k if the SIGN flag is set.	If SIGN = 1, PC <= k
JG {k, const, label}	Jump to the address k if the SIGN flag is not set.	If SIGN = 0, PC <= k
<b>MEMORY INSTRUCTIONS</b>		
FETCH rA {k, const}	Fetch the value stored at address k in RAM and store it in register rA	rA <= RAM[k]
FETCH rA rB	Fetch the value from the address stored in register rB from RAM and store it in register rA.	rA <= RAM[rB]
STORE rA {k, const}	Store the value in register rA at address k in RAM.	RAM[k] <= rA
STORE rA rB	Store the value in register rA at the address stored in register rB in RAM.	RAM[rB] <= rA
<b>CALL &amp; RETURN</b>		
CALL {k, const, label}	Jump to the subroutine at address denoted by the text literal <label> and store the return address on the stack.	STACK[SP] <= PC + 1; PC <= ADDR[<label>]
RETURN	Jump to the address pointed to by the stack pointer	PC <= STACK[SP]
<b>CONSTANT</b>		
CONSTANT <name> k	Define a constant named <name> that holds the literal k. It may be used in place of a literal in instructions that use a literal	N/A

**Notes:**

- rA and rB are 8-bit registers
- k is an 8-bit literal in hexadecimal prefixed with “0x”
- const is a constant holding an 8-bit literal
- label is a literal label marking subroutines that represent the 8-bit address of the closest instruction beside/beneath it
- {a, b, c} Denotes that only one of a, b, or c can be used as an argument in an instruction
- ALU instructions update the status flags by the following, unless indicated otherwise:
  - ZERO = 1 if ALU result is 0, and ZERO = 1 otherwise
  - CARRY = 1 if overflow or underflow occurs, and CARRY = 0 otherwise
  - SIGN = MSB of result