**Summary of LC-3 instructions:**

Operations:

ADD R2, R3, R4 ; R2 <- (R3) + (R4)

ADD R2, R3, imm5 ; R2 <- (R3) + SEXT(imm5) imm5 is 5-bit 2's complement, i.e. range #-16 to #+15

AND R2, R3, R4 ; R2 <- (R3) bitwise AND (R4)

AND R2, R3, imm5 ; R2 <- (R3) bitwise AND SEXT(imm5)

NOT R2, R3 ; R2 <- bitwise NOT (R3)

Data Movement:

LD R1, label ;load direct (label holds data; label is 9-bit 2's comp, i.e. range #-256 to #+255)

LDR R1, R2, offset6 ;load relative (R2 is pointer register; we will always set offset6 to 0)

LDI R1, label ;load indirect (label holds pointer)

ST R1, label ;store direct (label holds data)

STR R1, R2, offset6 ;store relative (R2 is pointer register; we will always set offset6 to 0)

STI R1, label ;store indirect (label holds pointer)

LEA R1, label ;address corresponding to label is loaded to R1 (no memory access)

Control:

These instructions determine what address to fetch the next instruction from:

BRnzp label ;use any combination of n, z, p to decide whether to branch to the instruction at label

JMP R2 ; Jump to address stored in R2

; i.e. the next instruction to be loaded & executed will the one at the address in R2

RET ;Return from subroutine

; (actually, this is just an alias for JMP R7 meaning transfer control to address in R7)

JSR label ;Jump to subroutine starting at address corresponding to label

JSRR R2 ;Jump to subroutine starting at address held in R2

*These are actually two versions of the same instruction, distinguished by an addressing mode flag in bit[11].  
Both start by capturing the current PC into R7, thus allowing return from subroutine using RET = JMP R7*

RTI ;you don’t want to know (at least not yet!)

TRAP trap vector

Trap vector table:

x20 GETC Read a single character from keyboard & return it in R0

x21 OUT Write the character in R0 to console

x23 IN Prompt user to input a single character, read it in from keyboard, echo it back to console, and return it in R0

x22 PUTS Write a string of ASCII characters (Null terminated) to the console. The starting address of the array must be in R0

x25 HALT HALT the processor.

Some useful ASCII hex values *(never quote ascii as decimal values!!)*

|  |  |  |  |
| --- | --- | --- | --- |
| newline x0A | ‘0’ x30 | ‘A’ x41 | ‘+’ x2B |
| space x20 | ‘9’ x39 | ‘a’ x61 | ‘-’ x2D |