== LC3 Instruction Set Summary

==

== For details, see, PP_Append_C-LC3_Instruction_Formats_and_Semantics

```
Instruction Format     Assembly Language   Operation
and Example Code       for Example Code    Semantics
--------------------   -----------------   --------------------

 Op  DR  SR1 i.. SR2    ADD R5 R6 R7        ADD R6 plus R7,
0001 101 110 000 111                        write result into R5. (see Note 0)

 Op  DR  SR1 i imm5     ADD R1 R6 x7        ADD R6 plus 0000000000000111,
0001 001 110 1 00111                        write result into R1. (Note 1)

 Op  DR  SR1 i.. SR2    AND R5 R7 R6        R5 gets Bitwise AND of R7 with
0101 101 111 000 110                        R6.

 Op  DR  SR1 i imm5     AND R1 R3 x6        R1 gets Bitwise AND of R3 with
0101 001 011 1 00110                        0000000000000110. (Note 1)

 Op  DR  SR1 ... ...    NOT R1 R2           R1 gets Bitwise NOT of R2.
1001 001 010 000 000

 Op  DR  imm9           LD  R1 x89          R1 gets contents of memory at
0010 001 0_1000_1001                        address = PC + x89. (Note 2)

 Op  DR  imm9           ST  R1 x89          R1 written to memory at
0011 001 0_1000_1001                        address = PC + x89.

 Op  DR  SR1 imm6       LDR R1 R6 x13       R1 gets contents of memory at
0110 001 110 01_0011                        address = R6 + x13. (Note 2)

 Op  DR  SR1 imm6       STR R1 R6 x13       R1 written to memory at
0111 001 110 01_0011                        address = R6 + x13.

 Op  DR  imm9           LDI R1 x89          R1 gets contents of memory via
pointer at
1010 001 0_1000_1001                        address = PC + x89. (Note 3)

 Op  DR  imm9           STI R1 x89          R1 written to memory via pointer at
1011 001 0_1000_1001                        address = PC + x89.

 Op  DR  imm9           LEA R1 x89          R1 gets PC + x89.
1110 001 1_1000_1001

 Op  nzp imm9           BRp    x89          PC get PC + x89, if
0000 001 0_1000_1001                        PSR.NZP = IR.nzp = 001. (Note 4)

 Op  ... SR1  ......    JMP R1              PC gets content of R7.
1100 000 111  000000    JMP R1

 OP  i.. SR1  ......    JSRR R6             PC gets R6, and simultaneously
0100 000 110  000000                        R7 gets PC. (Note 5)

 Op  i   imm11          JSR  x3F            PC gets PC + x3F, and simultaneously
0100 1   0_0011_1111                        R7 gets PC.

 Op  ... ... ... ...    RTI                 PC gets memory at R6; R6++;
1000 000 000 000 000    RTI                 PSR get memory at R6; R6++. (Note 6)

 Op  ....  vector       TRAP x3F            PC gets memory at address x003F, and
1111 0000  0011_1111    TRAP x08            simultaneously, R7 gets PC.
```

(Note 0) The "." in the instruction format means the bit is ignored when decoding and executing the instruction.

(Note 1) An "immediate" value, imm5, contained in IR[4:0] is converted from a 5-bit 2s-Complement representation to 16-bit 2s-Complement by duplicating bit IR[4] to all 11 most-significant bits. This is called "sign extension". The result is fed to the ALU.

(Note 2) The 9-bit immediate value, imm9, in IR[8:0] is sign extended to 16-bits.

(Note 3) A "pointer" is a memory location whose content is treated as an address. In this case, the memory location at address = PC + x89 contains a 16-bit address that is first fetched and then sent to the memory to do a second fetch. This is called "dereferencing a pointer."

(Note 4) The Processor Status Register (PSR) records the status of executed instructions, among other information. Part of the PSR is the branch condition codes (NZP). PSR.NZP is a 3-bit field that records whether the value written into a register was negative (PSR.NZP == 100) or was zero (PSR.NZP == 010) or was positive (PSR.NZP == 001). A branch instruction specifies when a branch will be taken by its IR.nzp bits, IR[11:9]. If a non-zero IR.nzp bit matches the non-zero PSR.NZP bit, the branch it taken. E.g., suppose the last register value written was zero. Then, PSR.NZP == 010. Suppose a branch instruction's bits are IR[11:9] = IR.nzp = 011. Then the branch will be taken because PSR.Z == 1 == IR.z. Note that the branch also would be taken if PSR.NZP = 001. This instruction is thus a branch on non-negative, written in LC3 assembly language as "BRzp".

(Note 5) A jump to a subroutine requires that the address immediately following the jump instruction be recorded so that the subroutine code can jump back when it completes its job. In the LC3, the "return address" is recorded in R7. The jump address (the address written into the PC that causes the jump) is calculated in the same way memory addresses are. In this case, PC <=== R6. That is, the PC flipflop gets written with the content of register R6. Because the PC is a flipflop, it can also be read simultaneously. So, the valu

(Note 6) The register R6 has a special usage: it is the "stack" pointer. It holds a memory address. Information can be written to an area of memory called the "stack". By convention, whenever we read or write the stack area, we use R6 as the address. We adjust R6 so that we can use the next stack location. When we write, we do R6--, when we read, we do R6++. This is called "pushing" and "popping" to/from the stack. Interrupts are a special kind of jump that is not controlled by an instruction. The PC is loaded from a special source (either memory or for some architectures from a special register.) This is triggered by a signal coming from external hardware. As a consequence, the PC is clobbered and we cannot return to the "interrupted" instruction. To deal with this, the architecture must record the address that was clobbered. So, part of what happens is the PC is pushed to the stack. When the interrupt code that was jumped to is finished, it jumps back to the interrupted instruction by popping the stack into the PC.

Our Notation Conventions:

```
    16'h123F     a 16-bit constant expressed in hex:
                 16' for bits (or, 8' for eight bits, etc.)
                 h for hex, d for decimal, or b for binary notation.
    SEXTk        k-to-16-bit sign-extension
```

LC3 Assembler's Notation Conventions:

```
    x10           constant expressed in hex
    #16           decimal (can have a minus sign, e.g., #-16)
    .ORIG x0200   1st line of .asm file: memory location to load program
    .END          last line of .asm file: end-of-program
    .FILL x1234   create a 16-bit word containing 16'h1234
    ;-comment
    .BLKW 5       create five 16-bit words containing 16'h0000
    .STRINGZ "string"     create seven 16-bit words, one 8-bit ASCII code
                          per word, last word contains ASCII NUL.
```