LCC Instruction Set Summary

Mnemonic	Binary Format	Flags Set	Description
br--	0000 code pcoffset9		on code, pc = pc + pcoffset9
add	0001 dr sr1 000 sr2	nzcv	dr = sr1 + sr2
add	0001 dr sr1 1 imm5	nzcv	dr = sr1 + imm5
ld	0010 dr pcoffset9		dr = mem[pc + pcoffset9]
st	0011 sr pcoffset9		mem[pc + pcoffset9] = sr
bl, call, or jsr	0100 1 pcoffset11		lr= pc; pc = pc + pcoffset11
blr or jsrr	0100 000 baser offset6		lr = pc; pc = baser + offset6
and	0101 dr sr1 000 sr2	nz	dr = sr1 & sr2
and	0101 dr sr1 1 imm5	nz	dr = sr1 & imm5
ldr	0110 dr baser offset6		dr = mem[baser + offset6]
str	0111 sr baser offset6		mem[baser + offset6] = sr
cmp	1000 000 sr1 000 sr2	nzcv	sr1 - sr2 (set flags)
cmp	1000 000 sr1 1 imm5	nzcv	sr1 - imm5 (set flags)
not	1001 dr sr1 000000	nz	dr = ~sr1
push	1010 sr 0000 00000		mem[--sp] = sr
pop	1010 dr 0000 00001		dr = mem[sp++];
srl	1010 sr ct 00010	nzc	sr >> ct (0 inserted on left, c=last out)
sra	1010 sr ct 00011	nzc	sr >> ct (sign bit replicated, c=last out)
sll	1010 sr ct 00100	nzc	sr << ct (0 inserted on right, c=last out)
rol	1010 sr ct 00101	nzc	sr << ct (rotate: bit 15 → bit 0, c=last out)
ror	1010 sr ct 00110	nzc	sr << ct (rotate: bit 0 → bit 15, c=last out)
mul	1010 dr sr 0 00111	nz	dr = dr * sr
div	1010 dr sr 0 01000	nz	dr = dr / sr
rem	1010 dr sr 0 01001	nz	dr = dr % sr
or	1010 dr sr 0 01010	nz	dr = dr | sr (bitwise OR)
xor	1010 dr sr 0 01011	nz	dr = dr ^ sr (bitwise exclusive OR)
mvr	1010 dr sr 0 01100		dr = sr
sext	1010 dr sr 0 01101	nz	dr sign extended (sr specifies field to extend)
sub	1011 dr sr1 000 sr2	nzcv	dr = sr1 - sr2
sub	1011 dr sr1 1 imm5	nzcv	dr = sr1 - imm5
jmp	1100 000 baser offset6		pc = baser + offset6
ret	1100 000 111 offset6		pc = lr + offset6
mvi	1101 dr imm9		dr = imm9
lea	1110 dr pcoffset9		dr = pc + pcoffset9

mov dr, imm9 is a pseudo-instruction translated to the machine instruction corresponding to mvi dr, imm9.
mov dr, sr is a pseudo-instruction translated to the machine instruction corresponding to mvr dr, sr.
dr, sr, sr1, sr2, baser are 3-bit register fields.
ct is a 4-bit shift count field (if omitted in a shift assembly instruction, it defaults to 1).
pcoffset9, pcoffset11, imm5, imm9, offset6 are signed number fields of the indicated length.
If offset6 is omitted in an assembly language instruction, it defaults to 0.

Trap Instructions

Mnemonic	Binary Format				Flags Set	Description
`halt`	1111	000	0	00000000	none	Stop execution, return to OS
`nl`	1111	000	0	00000001	none	Output newline
`dout`	1111	sr	0	00000010	none	Display signed number in `sr`
`udout`	1111	sr	0	00000011	none	Display unsigned number in `sr` in decimal
`hout`	1111	sr	0	00000100	none	Display hex number in `sr` in hex
`aout`	1111	sr	0	00000101	none	Display ASCII character in `sr`
`sout`	1111	sr	0	00000110	none	Display string `sr` points to
`din`	1111	dr	0	00000111	none	Read decimal number from keyboard into `dr`
`hin`	1111	dr	0	00001000	none	Read hex number from keyboard into `dr`
`ain`	1111	dr	0	00001001	none	Read ASCII character from keyboard into `dr`
`sin`	1111	sr	0	00001010	none	Input string into buffer `sr` points to

If `sr` or `dr` is omitted in a trap assembly language instruction, it defaults to `r0` (000).

Debugging Instructions

Mnemonic	Binary Format				Flags Set	Description
`m`	1111	000	0	00001011	none	Display all memory in use
`r`	1111	000	0	00001100	none	Display all registers
`s`	1111	000	0	00001101	none	Display stack
`bp`	1111	000	0	00001110	none	Software breakpoint (activates debugger)

Branch Instruction Codes (same suffixes can be used on the jmp instruction)

Mnemonic	Code	Branch occurs if
`brz` or `bre`	000	`z` = 1 (branch on zero, branch on equal)
`brnz` or `brne`	001	`z` = 0 (branch on nonzero, branch on not equal)
`brn`	010	`n` = 1 (branch on negative)
`brp`	011	`n` = `z` (branch on positive)
`brlt`	100	`n` ≠ `v` (branch on less than in signed comparison)
`brgt`	101	`n` = `v` and `z` = 0 (branch on greater than in signed comparison)
`brc` or `brb`	110	`c` = 1 (branch on carry or branch on below (less than in unsigned comparison)
`br` or `bral`	111	Branch always

Assembler Directives

Directive	Description
`.word <value>`	Create word initialized to `<value>`
`.fill <value>`	Same as `.word`
`.zero <size>`	Create block of `<size>` words initialized to 0
`.space <size>`	Same as `.zero`
`.blkw <size>`	Same as `.zero`
`.string <string>`	Create null-terminated ASCII `<string>`
`.stringz <string>`	Same as `.string`
`.asciz <string>`	Same as `.string`
`.start <label>`	Specify `<label>` as entry point (or use label `_start` on entry point)
`.global <var>`	Specify `<var>` is a global variable
`.globl <var>`	Same as `.globl`
`.extern <var>`	Specify `<var>` is an external variable
`.org <address>`	Reset location counter to higher `<address>`