LC-2 Instruction Summary

Instruction	Assembler Format	Example	Operation
Addition ADD	ADD dr, sr1, sr2 ADD dr, sr1, imm5	ADD R2, R3, R4 ADD R2, R3, #7	R2 ← R3 + R4 R2 ← R3 + 7
Logical AND AND	AND dr, sr1, sr2 AND dr, sr1, imm5	AND R2, R3, R4 AND R2, R3, #7	R2 ← R3 AND R4 R2 ← R3 AND 7
Conditional Branch	BR label BRn label BRz label BRp label BRnz label BRnp label BRzp label BRnzp label	 BRnz loop BRnzp loop	 Branch if the last result was negative or zero. Unconditional branch
Jump & Jump to Subroutine JMP / JSR	JMP label (L=0) JSR label (L=1)	JMP foo JSR foo	Jump to foo. Jump to foo and put return PC into R7
Jump & Jump to Subroutine Base + Offset JMPR / JSRR	JMPR baseR, offset6 (L=0) JSRR baseR, offset6 (L=1)	JMPR R2, #10 JSRR R2, #10	Jump to R2 + #10 Jump to R2 + #10 and put return PC into R7
Load Direct LD	LD dr, label	LD R4, count	R4 ← mem[count]
Load Indirect LDI	LDI dr, label	LDI R4, pointer	R4 ← mem[mem[pointer]]
Load Base + Offset LDR	LDR dr, baseR, offset6	LDR R4, R2, #10	R4 ← contents of mem[R2+#10]
Load Effective Address LEA	LEA dr, label	LEA R4, foo	R4 ← address of foo
Complement NOT	NOT dr, sr	NOT R4, R2	R4 ← NOT(R2)
Return from Subroutine RET	RET	RET	PC ← R7
Return from Interrupt RTI	RTI	RTI	NZP, PC ← top two values popped off stack
Store Direct ST	ST sr, label	ST R4, count	mem[count] ← R4

Store Indirect STI	STI SR, label	STI R4, pointer	mem[mem[pointer]] ← R4

Store Base + Offset STR	STR sr, baseR, offset6	STR R4, R2, #10	mem[R2+#10] ← R4

Operating System Call TRAP	TRAP x20	GETC	Get a character from keyboard. The character is not echoed onto the screen. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
	TRAP x21	OUT	Write a character in R0[7:0] to the screen
	TRAP x22	PUTS	Write a string pointed to by R0 to the screen.
	TRAP x23	IN	Print a prompt (pointed to by R0) on the screen and read a single character from the keyboard. The character is echoed onto the screen, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
	TRAP x25	HALT	Halt execution

General purpose registers:
The LC-2 has eight 16-bit general purpose registers R0 to R7.

Special memory locations:
xF3FC CRT status register (CRTSR). The ready bit (bit 15) indicates if the video device is ready to receive another character to print on the screen.
xF3FF CRT data register (CRTDR). A character written in the low byte of this register will be displayed on the screen.
xF400 Keyboard status register (KBSR). The ready bit (bit 15) indicates if the keyboard has received a new character.
xF401 Keyboard data register (KBDR). Bits [7:0] contain the last character typed on the keyboard.
xF402 Machine control register (MCR). Bit [15] is the clock enable bit. When cleared, instruction processing stops.

Notations:
baseR – base register
dr – destination register
imm5 – a five-bit immediate value
mem[address] – denotes the contents of memory at the given address
offset6 – a six-bit immediate value used in a Base+Offset instruction
sr – source register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD*	0001				dr			sr1			0	0	0	SR2		
ADD*	0001				dr			sr1			1	imm5				
AND*	0101				dr			sr1			0	0	0	sr2		
AND*	0101				dr			sr1			1	imm5				
BR	0000				n	z	p	pageoffset9								
JMP JSR	0100				0 1	0	0	pageoffset9								
JMPR JSRR	1100				0 1	0	0	baseR			offset6					
LD*	0010				dr			pageoffset9								
LDI*	1010				dr			pageoffset9								
LDR*	0110				dr			baseR			offset6					
LEA*	1110				dr			pageoffset9								
NOT*	1001				dr			sr			1	1	1	1	1	1
RET	1101				0	0	0	0	0	0	0	0	0	0	0	0
RTI	1000				0	0	0	0	0	0	0	0	0	0	0	0
ST	0011				sr			pageoffset9								
STI	1011				sr			pageoffset9								
STR	0111				sr			baseR			offset6					
TRAP	1111				0	0	0	0	trapvect8 x20 = GetC x21 = Out x22 = PutS x23 = In x25 = Halt							

Note: * indicates instructions that modify condition codes.

1. Operate instructions

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	DR			SR1			0	0	0	SR2		

0	0	0	1	DR	SR1	1	imm5

```
if (bit[5] == 0)
    DR = SR1 + SR2
else
    DR = SR1 + sign-extend(imm5)
set cc(DR)
```

Example:

```
        ADD   R2, R3, R4          ; R2 ← R3 + R4
        ADD   R2, R3, #7          ; R2 ← R3 + 7
```

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	DR			SR1			0	0	0	SR2		

0	1	0	1	DR	SR1	1	imm5

```
if (bit[5] == 0)
    DR = SR1 AND SR2
else
    DR = SR1 AND sign-extend(imm5)
set cc(DR)
```

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	DR			SR			1	1	1	1	1	1

There is no **OR** instruction. However, using DeMorgan's law A OR B is:

A OR B = (A′ AND B′)′

2. Data movement instructions

Load and Store

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				DR or SR			operand specifier								

Addressing Modes

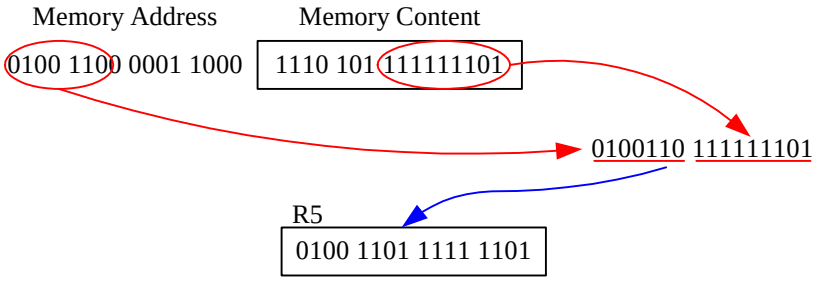
Immediate Mode

The LEA (1110) instruction loads the immediate value formed by concatenating bits [15:9] of the address where the instruction is stored (i.e., PC [15:9]) with bits [8:0] of the instruction. Those 16 bits are loaded into the register specified by bits [11:9].

memory address
x4C18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	1
LEA only				R5			x1FD								

First 7 bits [15:9] of address x4C18 = 0100 1100 0001 1000 is 0100 110.
Concatenate 0100 110 with 1 1111 1101 to give 0100 1101 1111 1101. This value is loaded into R5.



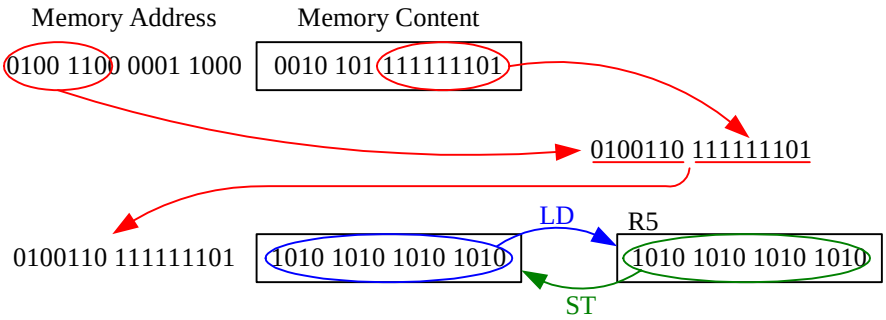
Direct Mode

LD (0010) and ST (0011) specify the *direct mode*. It loads or stores the value that is found in the memory address that is formed by concatenating bits [15:9] of the address where the instruction is stored (i.e., PC [15:9]) with bits [8:0] of the instruction (i.e., the 9 bits page offset).

memory address
x4C18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1
LD				R5			x1FD								

First 7 bits [15:9] of address x4C18 = 0100 1100 0001 1000 is 0100 110.
Concatenate 0100 110 with 1 1111 1101 to give 0100 1101 1111 1101. This value forms the address of a memory location. Load the contents stored in that memory location into R5.



Can access only memory locations that are on the same page as the instruction.

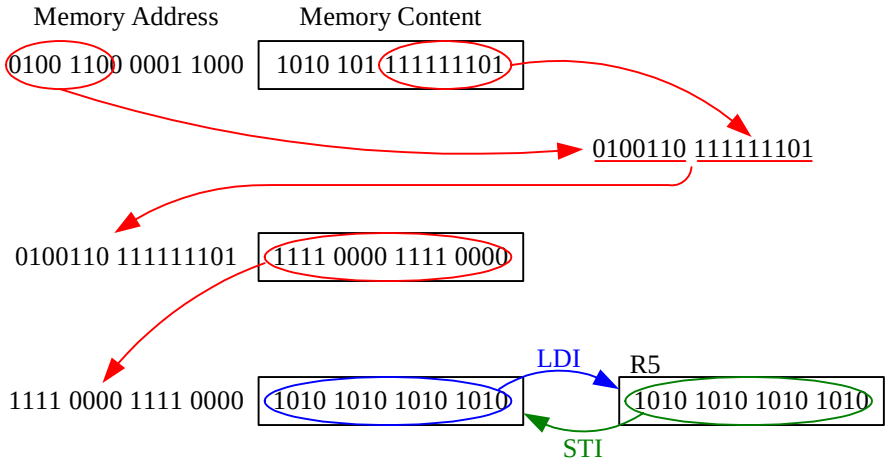
Example
LD R4, count ; R4 ← mem[count]

Indirect Mode

LDI (1010) and ST I(1011) specify the *indirect mode*. An address is first formed like the LD and ST. However, the contents form the address of the operand to be loaded or stored.

memory address
x4C18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1
LDI				R5			x1FD								



Example

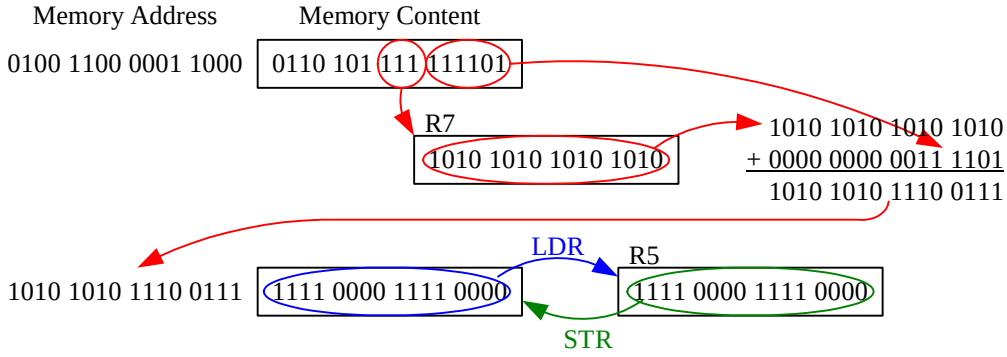
LDI R4, pointer ; R4 ← mem[mem[pointer]]

Base+Offset Mode

LDR (0110) and STR (0111) specify the *base+offset mode*. The address of the operand is obtained by adding the zero-extended six-bit offset to the content of the base register. The six-bit value is taken as a positive value.

memory address
x4C18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	1	1	1	1	1	1	0	1
LDR				R5			BaseR			Offset					



Example

LDR R4, R2, #10 ; R4 ← contents of mem[R2 + #10]

3. Control Instructions

Branch

The branch BR (0000) instruction format is

memory address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x4C18	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1
	BR				N	Z	P	page offset								

When a condition bit [11:9] (N, Z, P) is set, that corresponding condition code is checked. If that corresponding condition is set, then the PC is loaded with the address that is formed by concatenating PC[15:9] with the page offset [8:0].

All instructions that write values into registers set the three condition codes (i.e., the single-bit registers N, Z, P) depending on whether the value written is negative, zero, or positive. These instructions are ADD, AND, NOT, LD, LDI, LDR, and LEA.

BR	Label	BRnz	Label
BRn	Label	BRnp	Label
BRz	Label	BRzp	Label
BRp	Label	BRnzp	Label

The BR and BRnzp instructions are the same. Both provides an unconditional branch.

Jump

The jump instructions JMP (01000) and JMPR (11000) provide an unconditional jump (like the BR). The instruction format is

memory address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x4C18	0	1	0	0	0	0	0									
	JMP				L			page offset								

x4C18	1	1	0	0	0	0	0									
	JMPR				L			BaseR			Offset					

JMP uses the direct addressing mode. The address that it will jump to is obtained by concatenating bits [15:9] of the address where the instruction is stored (i.e., PC [15:9]) with bits [8:0] of the instruction (i.e., the 9 bits page offset).

JMPR uses the base+offset addressing mode. The address that it will jump to is obtained by adding the zero-extended six-bit offset to the content of the base register. The six-bit value is taken as a positive value.

Jump/Return Subroutine

The jump subroutine instructions JSR (01001) and JSRR (11001) call a subroutine located at the specified address. JSR uses direct addressing and JSRR uses base+offset similar to the JMP and JMPR respectively. In addition to the jump, both JSR and JSRR will store the return address (i.e. PC+1) into register R7.

The return from subroutine instruction RET (1101) copies the content of R7 to the PC.

Trap

The Trap (1111) instruction invokes a system routine. When the OS is finished performing the service call, the program counter is set to the address of the instruction following the TRAP instruction and the program continues.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1
TRAP								trap vector							

R7 ← PC;
PC ← mem[ZEXT(trapvector8)

Trap Number	Assembler Name	Description
x20	GETC	Read a single character from the keyboard. The character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x21	OUT	Write a character in R0[7:0] to the console.
x22	PUTS	Write a string pointed to by R0 to the console.
x23	IN	Print a prompt (pointed to by R0) on the screen and read a single character from the keyboard. The character is echoed onto the console, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x25	HALT	Halt execution and print a message on the console.

4. Assembler Directives

Assembler directives are commands for the assembler telling the assembler how to assemble the program. No code is generated for it. All assembler directives start with a period (.).

Directive	Format	Example	Operation
.ORIG	.ORIG value	.ORIG x3000	sets 3000 hex as the starting memory location to store the program.
.END	.END	.END	last line in program
.FILL	label .FILL value	num .FILL #48	allocates one word of memory and store the value 48_{10} in it.
.BLKW	label .BLKW number	array .BLKW 13	reserves a block of 13 words of memory. The label name “array” refers to the first location.
.STRINGZ	label .STRINGZ “a string”	prompt .STRINGZ “Enter a number? ”	allocates memory to store the string and terminated by a 0. The starting location is referred to as “prompt.”