Appendix A

The LC-3b ISA

A.1 Overview

The Instruction Set Architecture (ISA) of the LC-3b is defined as follows:

- **Memory address space** 16 bits, corresponding to 2^{16} locations, each containing one byte (eight bits). Addresses are numbered from 0 (i.e, x0000) to 65,535 (i.e., xFFFF). Addresses are used to identify memory locations and memory-mapped I/O device registers. Certain regions of memory are reserved for special uses, as described in Figure A.1.

- **Memory addressability** Each memory location contains one byte (eight bits) of information.

- **Bit numbering** Bits of all quantities are numbered, from right to left, starting with bit 0. The left-most bit of the contents of a memory location is bit 7.

- **Memory alignment** Two word-aligned memory locations are required to store one 16-bit word. Two memory locations are word-aligned if their addresses differ only in bit [0]. For example, locations x0006 and x0007 are word-aligned; locations x0007 and x0008 are not.

- **Memory addressing** The address of a byte of information stored in memory is the address of the location containing that byte. The address of a word of information stored in memory is the lower address of the pair of word-aligned memory locations containing the word. For example, a word stored in locations x0006 and x0007 has the address x0006. Since a word can be stored in memory only in two word-aligned locations, the address of a word of memory is always even.

- **Endian-ness** A word stored in memory at word address X has bits [7:0] stored in location X and bits [15:8] stored in location X+1. Since the less significant byte of the word is stored in location X and the more significant byte is stored in location X+1 (that is, the less significant byte "first"), the ordering is called little endian (for the little end first).

- **Instructions**   Instructions are 16 bits wide. Bits [15:12] specify the opcode (operation to be performed), bits [11:0] provide further information that is needed to execute the instruction. Instructions always occupy two word-aligned locations in the byte-addressable LC-3b memory. The specific operation of each LC-3b instruction is described in Section A.3.

- **Program counter**   A 16-bit register containing the word address of the next instruction to be processed.

- **General purpose registers**   Eight 16-bit registers, numbered from 000 to 111.

- **Condition codes**   Three one-bit registers: N (negative), Z (zero) and P (positive). Load instructions (LDB, LDW, and LEA) and operate instructions (ADD, AND, XOR, and SHF) each load a result into one of the eight general purpose registers. The condition codes are set, based on whether that result, taken as a 16-bit 2's complement integer, is negative (N = 1, Z,P = 0), zero (Z = 1, N,P = 0), or positive (P = 1, N,Z = 0). All other LC-3b instructions leave the condition codes unchanged.

- **Memory mapped I/O**   Input and Output are handled by load/store (LDW/STW) instructions using memory addresses to designate each I/O device register. Addresses xFE00 through xFFFF have been allocated to represent the addresses of I/O devices. See Figure A.1. Also, Table A.3 lists each of the relevant device registers that have been identified for the LC-3b thus far, along with their corresponding assigned addresses from the memory address space.

- **Interrupt processing**   I/O devices have the capability of interrupting the processor. Section A.4 describes the mechanism.

- **Processor Status Register**   A 16-bit register, containing status information about the current process that is executing. Four bits of the PSR have been defined thus far. PSR[15] specifies the privilege level of the executing process. PSR[2:0] contain the condition codes (PSR[2] is N, PSR[1] is Z, PSR[0] is P).

- **Privilege Mode**   PSR[15] = 0 is supervisor mode, PSR[15] = 1 is user mode. Interrupt initiation involves changing the privilege mode to supervisor mode. Interrupt service routines execute in supervisor mode.

- **Supervisor stack**   A region of memory in supervisor space accessible via the supervisor stack pointer (SSP). When PSR[15]=0, the stack pointer (R6) is SSP.

- **User stack**   A region of memory in user space accessible via the user stack pointer (USP). When PSR[15] = 1, the stack pointer (R6) is USP.
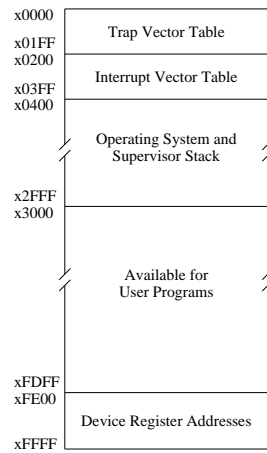
Figure A.1: Memory Map of the LC-3b

A.2 Notation

The notation in Table A.1 will be helpful in understanding the descriptions of the LC-3b instructions (Section A.3).

Notation	Meaning
xNumber	The number in hexadecimal notation.
#Number	The number in decimal notation.
A[l:r]	The *field* delimited by bit[l] on the left and bit[r] on the right, of the datum A. For example, if PC contains 0011001100111111, then PC[15:9] is 0011001. PC[2:2] is 1. If l and r are the same bit number, the notation is usually abbreviated PC[2].
amount4	A four-bit field, bits [3:0], of a shift instruction, designating the number of bits (absolute value) to shift. Range: 0..15.
BaseR	Base Register; one of R0..R7, used in conjunction with a six-bit offset to compute Base+offset addresses.
boffset6	A six-bit value, bits [5:0] of an instruction, used with the LDB and STB opcodes to compute the address of a memory operand. Bits [5:0] are taken as a six-bit signed 2's complement integer, sign-extended to 16 bits, and added to the base register to form the address. Range: -32..31.
DR	Destination Register; one of R0..R7, which specifies which register the result of an instruction should be written to.
imm5	A five-bit immediate value; bits [4:0] of an instruction when used as a literal (immediate) value. Taken as a 5-bit, 2's complement integer, it is sign-extended to 16 bits before it is used. Range: -16..15.
LABEL	An assembly language construct that identifies a location symbolically (i.e., by means of a name, rather than its 16-bit address).
LSHF(A, b)	Shift A to the left by b bits. The vacated bit positions are filled with zeros. The bits of A that are left-shifted out are dropped. For example, if A = 1111 1111 1111 1111 and b = 5, then LSHF(A, b) = 1111 1111 1110 0000.

Table A.1: Notational Conventions

Notation	Meaning
MEM[address]	Denotes the word starting at the given memory address. The byte at mem[address] forms bits[7:0] of the result and the byte mem[address+1] forms bits[15:8] of the result. In all cases the two addresses must be word-aligned.
mem[address]	Denotes the 8-bit contents of memory at the given address.
offset6	A six-bit value; bits[5:0] of an instruction; used with the LDW and STW opcodes to compute the address of a memory operand. Bits[5:0] are taken as a six-bit signed 2's complement integer, sign-extended to 16 bits, shifted one bit to the left, and then added to the base register to form the address. Range: -32..31.
PC	Program Counter; 16-bit register which contains the memory address of the *next* instruction to be fetched. For example, during execution of the instruction at address A, the PC contains address A+2, indicating that the next instruction is contained in locations A+2 and A+3. The PC must always be word aligned.
PCoffset9	A nine-bit value; bits [8:0] of an instruction; used with the BR and LEA opcodes to compute an address. Treated as a nine-bit 2's complement integer, it is sign-extended to 16 bits, shifted one bit to the left, and then added to the incremented PC. Range -256..255.
PCoffset11	An 11-bit value; bits[10:0] of an instruction; used with the JSR opcode to compute the target address of a subroutine call. Bits[10:0] are taken as an 11-bit 2's complement integer, sign-extended to 16 bits shifted one bit to the left, and then added to the incremented PC to form the target address. Range -1024..1023.
PSR	Processor Status Register; 16-bit register which contains status information of the process that is running. PSR[15] = privilege mode. PSR[2:0] contain the condition codes. PSR[2] = N, PSR[1] = Z, PSR[0] = P.
RSHF(A, b, s)	Shift A to the right by b bits. The vacated bit positions are filled by the bit indicated by s. The bits of A that are right-shifted out are dropped. For example, if A = 1111 1111 1111 1111 and b = 7 and s = 0, then RSHF(A, b, s) = 0000 0001 1111 1111.
setcc()	Indicates that condition codes N, Z, and P are set based on the value of the result written to DR. If the value is negative, N = 1, Z = 0, P = 0. If the value is zero, N = 0, Z = 1, P = 0. If the value is positive, N = 0, Z = 0, P = 1.
SEXT(A)	Sign-extend A. The most significant bit of A is replicated as many times as necessary to extend A to 16 bits. For example, if A = 110000, then SEXT(A) = 1111 1111 1111 0000.
SP	The current stack pointer. R6 is the current stack pointer. There are two stacks, one for each privilege mode. SP is SSP if PSR[15] = 0; SP is USP if PSR[15] = 1.
SR, SR1, SR2	Source Register; one of R0..R7 which specifies the register from which a source operand is obtained.
SSP	The supervisor stack pointer.
trapvect8	An eight bit value; bits [7:0] of an instruction; used with the TRAP opcode to determine the starting address of a trap service routine. Bits [7:0] are taken as an unsigned integer, zero-extended to 16 bits, and shifted left one bit. This is the address of the memory location containing the starting address of the corresponding service routine. Range 0..255.
USP	The user stack pointer.
ZEXT(A)	Zero-extend A. Zeroes are appended to the left-most bit of A to extend it to 16 bits. For example, if A = 110000, then ZEXT(A) = 0000 0000 0011 0000.

Table A.1: Notational Conventions (continued)

A.3 The Instruction Set

The LC-3b supports a rich, but lean, instruction set. Each 16-bit instruction consists of an opcode (bits[15:12]) plus 12 additional bits to specify the other information which is needed to carry out the work of that instruction. Figure A.2 summarizes the 14 different opcodes in the LC-3b and the specification of the remaining bits of each instruction. The 15th and 16th 4-bit opcodes are not specified, but are reserved for future use. Figure A.3 shows the entire LC-3b instruction set. On the following pages, the instructions will be described in greater detail. For each instruction, we show the assembly language representation, the format of the 16-bit instruction, the operation of the instruction, an English-language description of its operation, and one or more examples of the instruction. Where relevant, additional notes about the instruction are also provided.

	15 14 13 12	11 10 9	8 7 6	5	4 3 2 1 0
ADD[+]	0001	DR	SR1	A	op.spec
AND[+]	0101	DR	SR1	A	op.spec
BR	0000	n z p	PCoffset9		
JMP	1100	000	BaseR	000000	
JSR(R)	0100	A	operand.specifier		
LDB[+]	0010	DR	BaseR	boffset6	
LDW[+]	0110	DR	BaseR	offset6	
LEA[+]	1110	DR	PCoffset9		
RTI	1000	000000000000			
SHF[+]	1101	DR	SR	A D	amount4
STB	0011	SR	BaseR	boffset6	
STW	0111	SR	BaseR	offset6	
TRAP	1111	0000	trapvect8		
XOR[+]	1001	DR	SR1	A	op.spec
not used	1010				
not used	1011				

Figure A.2: Format of the instructions for the 16 LC-3b opcodes. NOTE: + indicates instructions that modify condition codes.

	15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0
ADD+	0001	DR	SR1	0	00	SR2
ADD+	0001	DR	SR1	1	imm5	
AND+	0101	DR	SR1	0	00	SR2
AND+	0101	DR	SR1	1	imm5	
BR	0000	n z p	PCoffset9			
JMP	1100	000	BaseR	000000		
JSR	0100	1	PCoffset11			
JSRR	0100	0 00	BaseR	000000		
LDB+	0010	DR	BaseR	boffset6		
LDW+	0110	DR	BaseR	offset6		
LEA+	1110	DR	PCoffset9			
NOT+	1001	DR	SR	1	11111	
RET	1100	000	111	000000		
RTI	1000	000000000000				
LSHF+	1101	DR	SR	0	0	amount4
RSHFL+	1101	DR	SR	0	1	amount4
RSHFA+	1101	DR	SR	1	1	amount4
STB	0011	SR	BaseR	boffset6		
STW	0111	SR	BaseR	offset6		
TRAP	1111	0000	trapvect8			
XOR+	1001	DR	SR1	0	00	SR2
XOR+	1001	DR	SR	1	imm5	
not used	1010					
not used	1011					

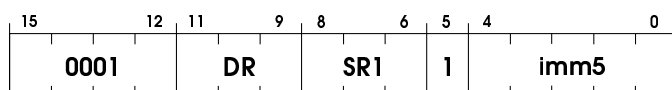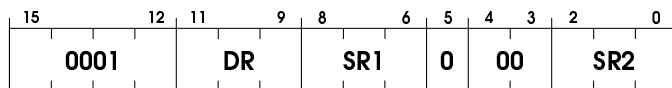Figure A.3: Format of the entire LC-3b Instruction Set. NOTE: + indicates instructions that modify condition codes.

ADD   Addition

Assembler Formats

    ADD   DR, SR1, SR2
    ADD   DR, SR1, imm5

Encodings

15 12	11 9	8 6	5	4 3	2 0
0001	DR	SR1	0	00	SR2

15 12	11 9	8 6	5	4 0
0001	DR	SR1	1	imm5

Operation

    if (bit[5] == 0)
      DR = SR1 + SR2;
    else
      DR = SR1 + SEXT(imm5);
    setcc();

Description

If bit [5] is 0, the second source operand is obtained from SR2. If bit [5] is 1, the second source operand is obtained by sign-extending the imm5 field to 16 bits. In both cases, the second source operand is added to the contents of SR1, and the result stored in DR. The condition codes are set, based on whether the result is negative, zero, or positive.

Examples

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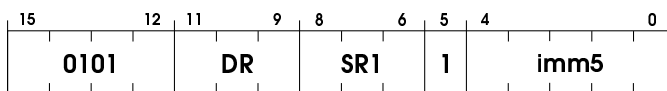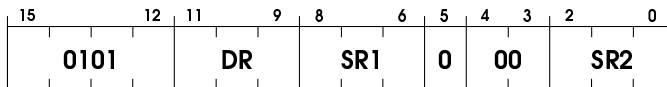   ADD   R2, R3, #7   ; R2 ← R3 + 7

AND   Bitwise logical AND

Assembler Formats

```
AND    DR, SR1, SR2
AND    DR, SR1, imm5
```

Encodings

15	12	11	9	8	6	5	4	3	2	0
0101		DR		SR1		0	00		SR2	

15	12	11	9	8	6	5	4	0
0101		DR		SR1		1	imm5	

Operation

```
if (bit[5] == 0)
   DR = SR1 AND SR2;
else
   DR = SR1 AND SEXT(imm5);
setcc();
```

Description

If bit [5] is 0, the second source operand is obtained from SR2. If bit [5] is 1, the second source operand is obtained by sign-extending the imm5 field to 16 bits. In either case, the second-source operand and the contents of SR1 are bitwise ANDed, and the result stored in DR. The condition codes are set, based on whether the binary value produced, taken as a 2's complement integer, is negative, zero, or positive.

Examples

```
AND    R2, R3, R4      ; R2 ← R3 AND R4
AND    R2, R3, #7      ; R2 ← R3 AND 7
```
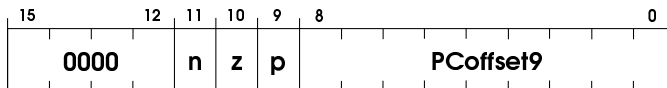
BR   Conditional Branch

Assembler Formats

BRn	LABEL	BRzp	LABEL
BRz	LABEL	BRnp	LABEL
BRp	LABEL	BRnz	LABEL
BR[†]	LABEL	BRnzp	LABEL

Encoding

15	12	11	10	9	8								0
0000		n	z	p				PCoffset9					

Operation

if ((n AND N) OR (z AND Z) OR (p AND P))
    PC = PC[‡] + LSHF(SEXT(PCoffset9), 1);

Description

The condition codes specified by the state of bits [11:9] are tested, as follows: If bit [11] is set, N is tested; if
bit [11] is clear, N is not tested. If bit [10] is set, Z is tested, etc. If any of the condition codes tested is set,
the program branches to the location specified by sign-extending the PCoffset9 field to 16 bits, left-shifting
it one bit, and adding the result to the incremented PC. The PCoffset9 field specifies the number of
instructions, forward or backwards, to branch over.

Examples

BRzp	LOOP	; Branch to LOOP if the last result was zero or positive.
BR[†]	NEXT	; Unconditionally Branch to NEXT.

[†]The assembly language opcode BR is interpreted the same as BRnzp; that is, always branch to the target address.
[‡]This is the incremented PC
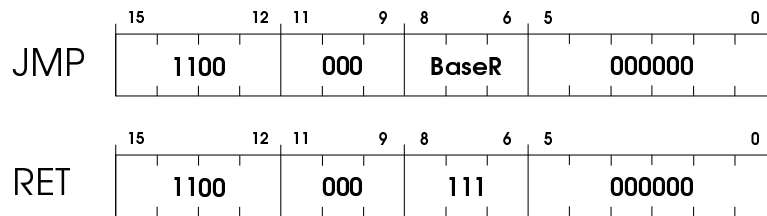
JMP   Jump

RET   Return from Subroutine

Assembler Formats

```
JMP    BaseR
RET
```

Encodings

15	12	11	9	8	6	5	0

JMP: `1100` `000` `BaseR` `000000`

JMP

| 15 | 12 | 11 | 9 | 8 | 6 | 5 | 0 |

RET: `1100` `000` `111` `000000`

RET

Operation

```
PC = BaseR;
```

Description

The program unconditionally jumps to the location specified by the contents of the base register. Bits [8:6] identify the base register.

Examples

```
JMP    R2      ; PC ← R2
RET            ; PC ← R7
```

Notes

The RET instruction is a special case of the JMP instruction. The PC is loaded with the contents of R7, which contains the linkage back to the instruction following the subroutine call instruction.

If the base register contains an odd address, an illegal operand address exception occurs.
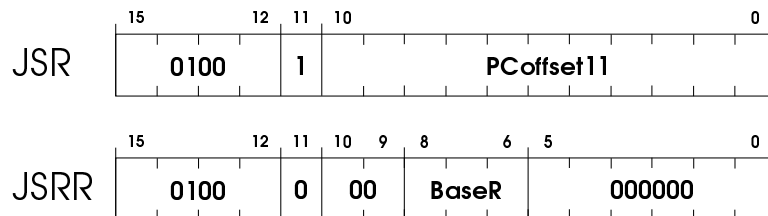
JSR   Jump to Subroutine

JSRR

Assembler Formats

    JSR   LABEL
    JSRR BaseR

Encodings

15	12	11	10						0
JSR	0100	1			PCoffset11				

15	12	11	10 9	8	6 5		0
JSRR	0100	0	00	BaseR		000000	

Operation

R7 = PC†;
if (bit[11] == 0)
  PC = BaseR;
else
  PC = PC† + LSHF(SEXT(PCoffset11), 1);

Description

First, the incremented PC is saved in R7. This is the linkage back to the calling routine. Then, the PC is loaded with the address of the first instruction of the subroutine, causing an unconditional jump to that address. The address of the subroutine is obtained from the base register (if bit[11] is 0), or the address is computed by sign-extending bits [10:0] to 16 bits, left-shifting the result one bit, and then adding this value to the incremented PC (if bit[11] is 1).

Examples

    JSR   QUEUE   ; Put the address of the instruction following JSR into R7; Jump to QUEUE.
    JSRR R3   ; Put the address following JSRR into R7; Jump to the address contained in R3.

Note

If bit[11] is 0, the base register must contain a word address. If the base register contains an odd address, an illegal operand address exception occurs.
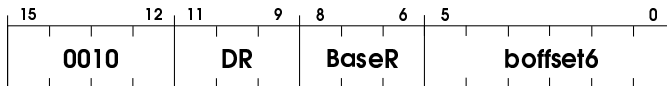
---

†This is the incremented PC.

LDB Load Byte

Assembler Format

LDB   DR, BaseR, boffset6

Encoding

15 12	11 9	8 6	5 0
0010	DR	BaseR	boffset6

Operation

DR = SEXT(mem[BaseR + SEXT(boffset6)]);
setcc();

Description

An address is computed by sign-extending boffset6 to 16 bits and adding the result to the contents of the base register. The 8-bit contents of memory at this address are sign-extended and stored into DR. The condition codes are set, based on whether the 16-bit value loaded into DR is negative, zero, or positive.

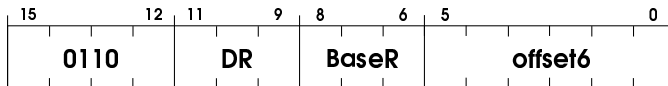## Example

LDB   R4, R2, #10   ; R4 ← SEXT(mem[R2 + 10])

LDW   **Load Word**

Assembler Format

LDW   DR, BaseR, offset6

Encoding

15 12	11 9	8 6	5 0
0110	DR	BaseR	offset6

Operation

DR = MEM[BaseR + LSHF(SEXT(offset6), 1)];
setcc();

Description

A word-aligned address is computed by sign-extending offset6 to 16 bits, left-shifting the result by one bit, and then adding this to the contents of the base register. The word starting at this address is stored into DR. The condition codes are set, based on whether the value loaded is negative, zero, or positive.

Example

LDW   R4, R2, #10   ; R4 ← MEM[R2 + 20]

Note

The base register must contain a word address (i.e., its contents must be even). If the base register contains an odd address, an illegal operand address exception occurs.
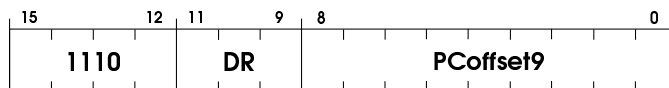
LEA   **Load Effective Address**

Assembler Format

LEA   DR, LABEL

Encoding

15	12	11	9	8	0
1110		DR		PCoffset9	

Operation

DR = PC† + LSHF(SEXT(PCoffset9),1);
setcc();

Description

The register specified by DR is loaded with the address formed by sign-extending the PCoffset9 to 16 bits, left-shifting it one bit and then adding it to the incremented PC.[‡] The condition codes are set, based on whether the value (i.e., address) loaded is negative, zero, or positive.

Example

LEA   R4, TARGET   ; R4 ← address of TARGET

[†]This is the incremented PC.

[‡]The LEA instruction does not read memory to obtain the information to load into DR. The address, itself, is loaded into DR.
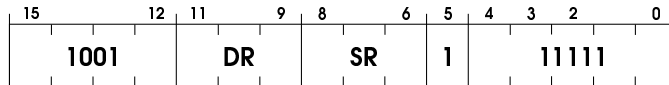
NOT[†]   Bitwise Complement

Assembler Format

NOT   DR, SR

Encoding

15 12	11 9	8 6	5	4 3 2 0
1001	DR	SR	1	11111

Operation

DR = NOT(SR);
setcc();

Description

The contents of SR are bitwise complemented and the result stored in DR. The condition codes are set, based on whether the binary value produced, taken as a 2's complement integer, is negative, zero, or positive.

Example

NOT   R4, R2   ; R4 ← NOT(R2)

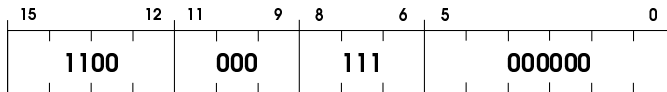[†]The NOT instruction is a specific encoding of the XOR instruction. See also XOR.

RET[†]   **Return from Subroutine**

Assembler Format

RET

Encoding

15	12	11	9	8	6	5	0
1100		000		111		000000	

Operation

PC = R7;

Description

The PC is loaded with the value in R7. This causes a return from a previous JSR instruction.

Example

RET   ; PC ← R7

Note

The contents of R7 must be an even address. If not, an illegal operand address exception occurs.

[†]The RET instruction is a specific encoding of the JMP instruction. See also JMP.
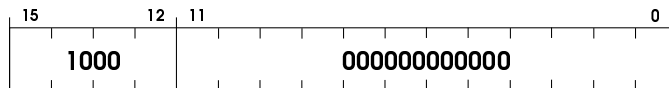
RTI   Return from Interrupt

Assembler Format

RTI

Encoding

15 12	11 0
1000	000000000000

Operation

if (PSR[15] == 1) privilege mode violation
PC = MEM[R6]; R6 is the SSP
R6 = R6 + 2;
TEMP = MEM[R6];
R6 = R6 + 2;
PSR = TEMP; the privilege mode and condition codes of the interrupted process are restored

Description

The top two words are popped off the stack and loaded into PC, PSR.

Example

RTI   ; PC, PSR ← top two values popped off the stack.

Notes

On an external interrupt, the initiating sequence first changes the privilege mode to supervisor (PSR[15]=0).
Then the PSR and PC of the interrupted process are pushed onto the supervisor stack before loading the PC
with the starting address of the interrupt service routine. The PSR that is pushed onto the supervisor stack
contains the privilege level and the condition codes of the interrupted process. The interrupt service routine
runs with supervisor privilege. The last instruction in the service routine is RTI, which returns control to the
interrupted process by popping two values off the supervisor stack, first to restore the PC to the address of
the instruction that was about to be processed when the interrupt was initiated, and second to restore the
PSR to the values they had when the interrupt was initiated. See also Section A.4.

RTI can be executed only if the processor is in supervisor state (i.e., PSR[15] = 0).

SHF   Bit Shift
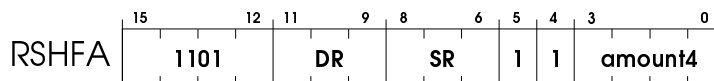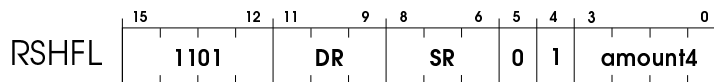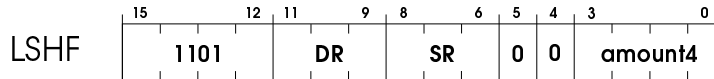
Assembler Formats

```
LSHF     DR, SR, amount4      ; left shift
RSHFL    DR, SR, amount4      ; right shift logical
RSHFA    DR, SR, amount4      ; right shift arithmetic
```

Encodings

LSHF

15 12	11 9	8 6	5	4	3 0
1101	DR	SR	0	0	amount4

RSHFL

15 12	11 9	8 6	5	4	3 0
1101	DR	SR	0	1	amount4

RSHFA

15 12	11 9	8 6	5	4	3 0
1101	DR	SR	1	1	amount4

Operation

```
if (bit[4] == 0)
   DR = LSHF(SR, amount4);
else
   if (bit[5] == 0)
      DR = RSHF(SR, amount4, 0);
   else
      DR = RSHF(SR, amount4, SR[15]);
setcc();
```

Description

Bit [4] determines the direction (left or right) of the shift; bit [5] determines whether a right shift is arithmetic or logical. If bit [4] is 0, the source operand in SR is shifted left by the number of bit positions indicated by amount4. If bit [4] is 1, the source operand is shifted right by amount4 bits. If the operation is a right shift, bit [5] of the instruction determines whether the sign bit of the original source operand is preserved. If bit [5] is 1, the right shift is an arithmetic shift; thus the original SR[15] is shifted into the vacated bit positions. If bit[5] is 0, zeroes are shifted into the vacated bit positions. The result is stored in DR. The condition codes are set, based on whether the result is negative, zero, or positive.

Examples

```
LSHF     R2, R3, #3       ; R2 ← LSHF(R3, #3)
RSHFL    R2, R3, #7       ; R2 ← RSHF(R3, #7, 0)
RSHFA    R2, R3, #7       ; R2 ← RSHF(R3, #7, R3[15])
```
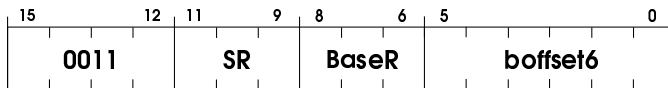
STB   Store Byte

Assembler Format

STB   SR, BaseR, boffset6

Encoding

15 12	11 9	8 6	5 0
0011	SR	BaseR	boffset6

Operation

mem[BaseR + SEXT(boffset6)] = SR[7:0];

Description

The low 8 bits of the register specified by SR are stored into the memory location whose address is obtained by sign-extending boffset6 to 16 bits and adding the result to the contents of the base register.

Example

STB   R4, R2, #10   ; mem[R2 + 10] ← R4
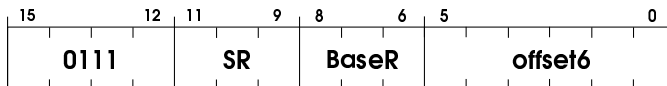
STW

Store Word

Assembler Format

STW   SR, BaseR, offset6

Encoding

15	12	11	9	8	6	5	0
0111		SR		BaseR		offset6	

Operation

MEM[BaseR + LSHF(SEXT(offset6), 1)] = SR;

Description

The contents of SR are stored into the word-aligned memory location whose address is obtained by sign-extending offset6 to 16 bits, left-shifting the result by one bit and adding this to the contents of the base register.

Example

STW   R4, R2, #10   ; MEM[R2 + 20] ← R4

Note

The base register must contain a word address (i.e., its contents must be even). If the base register contains an odd address, an illegal operand address exception occurs.
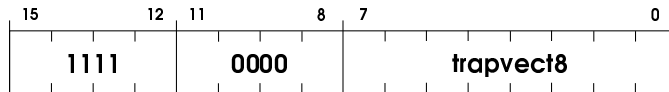
TRAP   Operating System Call

Assembler Format

TRAP   trapvector8

Encoding

15	12	11	8	7	0
1111		0000		trapvect8	

Operation

R7 = PC†;
PC = MEM[LSHF(ZEXT(trapvect8), 1)];

Description

First R7 is loaded with the incremented PC. (This enables a return to the instruction physically following the TRAP instruction in the original program after the service routine has completed execution.) Then trapvect8 is zero-extended to 16 bits, and left shifted one bit, forming the address of the trap vector table entry which contains the starting address of the service routine. The starting address of the service routine is loaded into PC.

Example

TRAP   x23   ; Directs the operating system to execute the **IN** system call. The starting address
                   ; of this system call is contained in memory location x0046.

Note

Memory locations x0000 through x01FF are available to contain starting addresses for system calls specified by their corresponding trapvectors. This region of memory is called the trap vector table. See Table A.2.

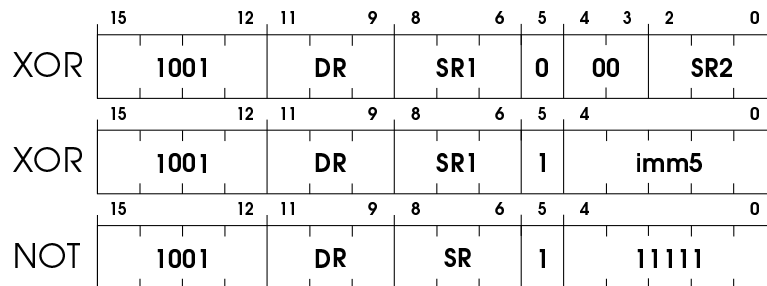†This is the incremented PC.

XOR
NOT

Bitwise Exclusive-OR

Bitwise Complement

Assembler Formats

```
XOR    DR, SR1, SR2
XOR    DR, SR1, imm5
NOT    DR, SR
```

Encodings

15 12	11 9	8 6	5	4 3 2	0
XOR 1001	DR	SR1	0	00	SR2

15 12	11 9	8 6	5	4 0
XOR 1001	DR	SR1	1	imm5

15 12	11 9	8 6	5	4 0
NOT 1001	DR	SR	1	11111

Operation

```
if (bit[5] == 0)
   DR = SR1 XOR SR2;
else
   DR = SR1 XOR SEXT(imm5);
setcc();
```

Description

If bit [5] is 0, the second source operand is obtained from SR2. If bit [5] is 1, the second source operand is obtained by sign-extending the imm5 field to 16 bits. In both cases, the second source operand is XORed with the contents of SR1, and the result stored in DR. The condition codes are set, based on whether the binary value produced, taken as a 2's complement integer, is negative, zero, or positive.

Examples

```
XOR    R3, R1, R2      ; R3 ← R1 XOR R2
XOR    R3, R1, #12     ; R3 ← R1 with bits [3], [2] complemented.
NOT    R3, R2          ; R3 ← NOT(R2)
```

Note

The NOT instruction is a special case of the XOR instruction.

TRAP vector	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 display.
x22	PUTS	Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. Writing terminates with the occurrence of x00 in a memory location.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console display, 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.

Table A.2: TRAP vector table

Address	I/O Register Name	I/O Register Function
xFE00	Keyboard status register	Also known as KBSR. The ready bit (bit [15]) indicates if the keyboard has received a new character.
xFE02	Keyboard data register	Also known as KBDR. Bits [7:0] contain the last character typed on the keyboard.
xFE04	Display status register	Also known as DSR. The ready bit (bit [15]) indicates if the display device is ready to receive another character to print on the screen.
xFE06	Display data register	Also known as DDR. A character written in the low byte of this register will be displayed on the screen.
xFFFE	Machine control register	Also known as MCR. Bit [15] is the clock enable bit. When cleared, instruction processing stops.

Table A.3: Device register assignments

A.4   Interrupt Processing

Events external to the process that is running are able to interrupt the processor. A common example of this is interrupt-driven I/O.

Associated with each external event that can interrupt the processor is an eight-bit interrupt vector (i.e., INTV) that provides an entry point into a 256 entry interrupt vector table. The starting address of the interrupt vector table is x0200. That is, the interrupt vector table occupies memory locations x0200 to x03FF. Each entry in the interrupt vector table contains the starting address of the interrupt service routine that handles the corresponding external event.

In order for an interrupt to occur, the following must be true:

1. The Interrupt Enable bit (IE) associated with the event must be set (i.e., IE=1).

2. The priority of the event must be greater than the priority of the process that is executing.

At this time, the LC-3b ISA specifies only one event that can interrupt the processor: Keyboard input. This is an example of interrupt-driven I/O. To occur, bit [14] in the Keyboard Status

Register must be 1. In the LC-3b, the priority of the keyboard interrupt is greater than the priority of user programs.

If someone strikes a key on the keyboard, the process that is executing is interrupted if IE=1 and the priority of the process executing is less than the priority of the cause of the interrupt. The interrupt service routine is initiated as follows:

1. IE is temporarily disabled for all new interrupts. i.e., no new interrupts are temporarily allowed access to the processor.

2. The privilege mode is set to Supervisor Mode (PSR[15]=0) if it is not already set to supervisor mode.

3. R6 is set to the supervisor stack pointer if that is not already the case.

4. The PSR and PC of the interrupted process are pushed on to the supervisor stack.

5. The interrupting event supplies its eight-bit interrupt vector (INTV). The interrupt vector for the keyboard is x40.

6. The processor left-shifts the interrupt vector one bit, yielding x80, and adds it to the base address of the interrupt vector table (x200), yielding the address of the memory location (x0280), which contains the starting address of the interrupt service routine.

7. The contents of memory location x0280 are read and loaded into the PC.

8. IE is again enabled for all interrupts.

9. The processor begins execution of the interrupt service routine.

The last instruction in an interrupt service routine is RTI. The top two elements of the processor stack are popped and loaded into the PC and PSR, respectively. Processing then continues where the interrupted process left off.