Chapter 1

The LC-3b ISA

1.1 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 (8 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 1.1.

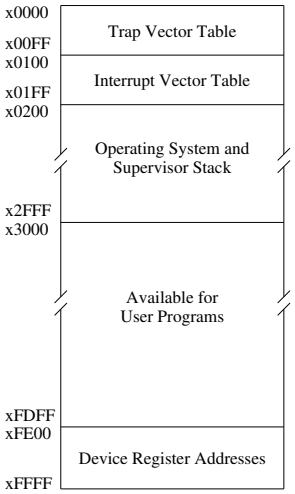


Figure 1.1: Memory Map of the LC-3b

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 15.

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 are always 16-bit-aligned in the byte-addressable LC-3b memory. The specific operation of each LC-3b instruction is described in Section 1.3.

Program counter A 16-bit register containing the 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 (LDR, LDB, LDI, and LEA) and operate instructions (ADD, AND, and NOT) 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 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 1.1. Also, Table 1.3 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 1.4 describes the mechanism.

Priority Level The LC-3b supports eight levels of priority. Priority level 7 (PL7) is the highest; PL0 is the lowest. The priority level of the currently executing process is specified in bits PSR[10:8].

Processor Status Register (PSR) A 16-bit register, containing status information about the currently executing process. Seven bits of the PSR have been defined thus far. PSR[15] specifies the privilege mode of the executing process. PSR[10:8] specifies the priority level of the currently executing process. PSR[2:0] contain the condition codes. PSR[2] is N, PSR[1] is Z, and PSR[0] is P.

Privilege Mode The LC-3b specifies two levels of privilege, Supervisor mode (privileged) and User mode (unprivileged). Interrupt service routines execute in Supervisor mode. The privilege mode is specified by PSR[15]. PSR[15]=0 indicates Supervisor mode; PSR[15]=1 indicates User 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.

1.2 Notation

Notation	Meaning
xNumber	The number in hexadecimal notation.
#Number	The number in decimal notation.
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 off are dropped. For example, if A = 1111 1111 1111 1111 and b = 5, then A << b = 1111 1111 1110 0000.
A >> b,c	Shift A to the right by b bits. The vacated bit positions are filled by the bit indicated by c. The bits of A that are right-shifted off are dropped. If A = 1111 1111 1111 1111, b = 7, c = 0, then A >> b,c = 0000 0001 1111 1111.
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].
BaseR	Base Register; one of R0..R7, used in conjunction with a six-bit offset to compute Base+offset addresses.
DR	Destination Register; one of R0..R7, which specifies which register the result of an instruction should be written to.
imm4	A four-bit immediate value. Taken as a 4-bit unsigned integer.
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).
mem[address]	Denotes the 8-bit (byte) contents of memory at the given address.
memWord[address]	Denotes the 16-bit (word) contents of memory starting at the given 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 address is treated as word-aligned, i.e., bit [0] is treated as 0.
offset6	A six-bit value; bits[5:0] of an instruction; used with the Base+offset addressing mode. Bits[5:0] are taken as a six-bit signed 2's complement integer, sign-extended to 16 bits and then added to the Base Register to form an 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 the next instruction is contained in A+2. The PC is always treated as word-aligned, meaning PC[0] is ignored.
PCoffset9	A nine-bit value; bits[8:0] of an instruction; used with the PC+offset addressing mode. Bits[8:0] are taken as a nine-bit signed 2's complement integer, sign-extended to 16 bits, left-shifted, and then added to the incremented PC to form an address. 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, left-shifted, 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] contains the condition codes. PSR[2] = N, PSR[1] = Z, PSR[0] = P.
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. The SP is always treated as word-aligned, meaning SP[0] is ignored.
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.
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 1.1: Notational Conventions

1.3 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 1.3 summarizes the 16 different opcodes in the LC-3b and the specification of the remaining bits of each instruction. In the following pages, the instructions are 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.

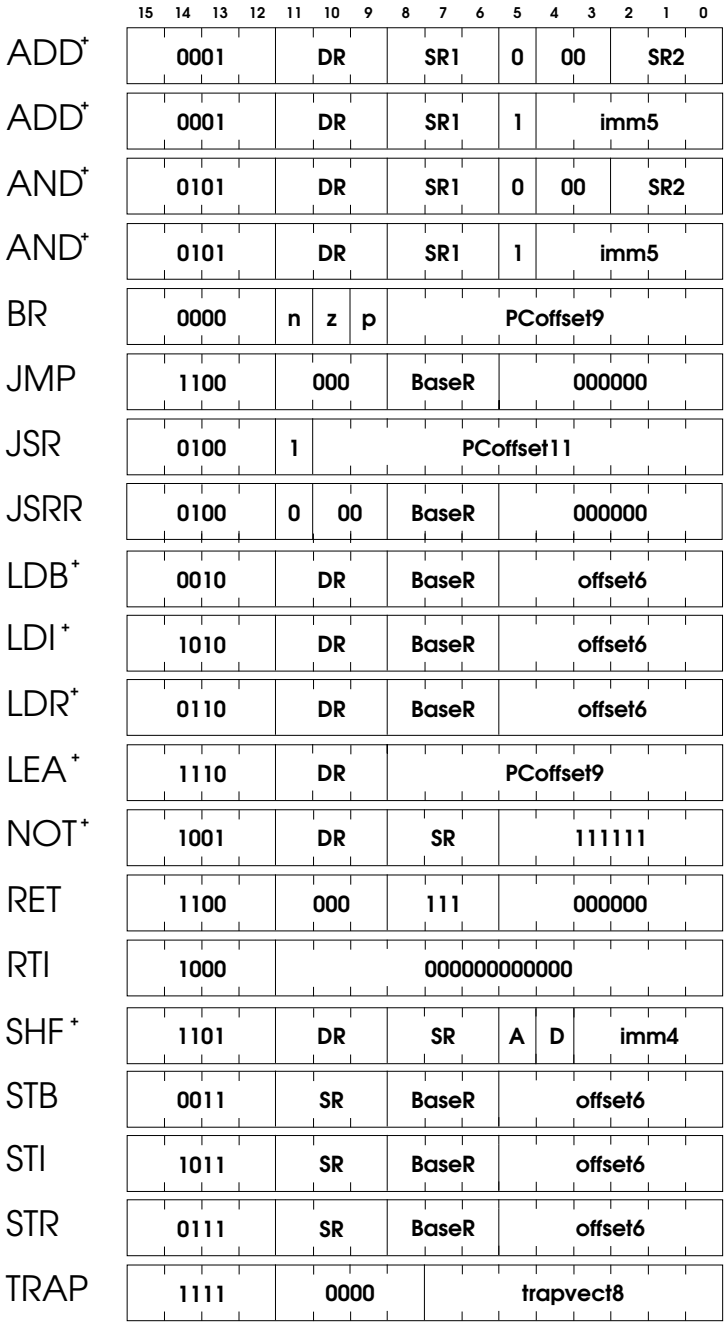


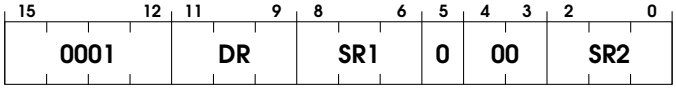
Figure 1.2: LC-3b Instruction Formats. NOTE: + indicates instructions that modify condition codes.

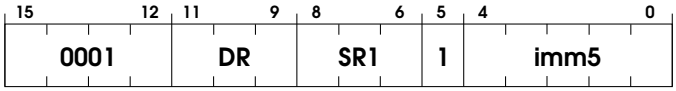
ADD Addition

Assembler Formats

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

Encodings





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

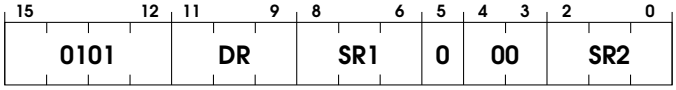
```
ADD    R2, R3, R4      ; R2 ← R3 + R4
ADD    R2, R3, #7      ; R2 ← R3 + 7
```

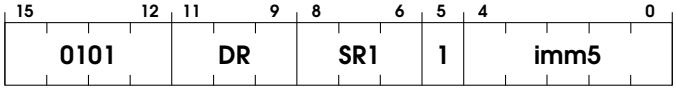
AND — Bitwise logical AND

Assembler Formats

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

Encodings





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

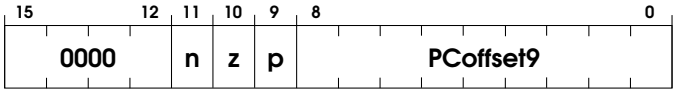
```
AND    R2, R3, R4      ; R2 ← R3 AND R4
AND    R2, R3, #7      ; R2 ← R3 AND 7
```

BR Conditional Branch

Assembler Formats

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

Encoding



Operation

```
if ((n AND N) OR (z AND Z) OR (p AND P))
   PC = PC‡ + (SEXT(PCoffset9) << 1);
```

Description

The condition codes specified by the state of bits [11:9] are tested. 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 adding the sign-extended and left-shifted PCoffset9 field to the incremented PC. In otherwords, the PCoffset9 field specifies the number of instructions, forwards 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

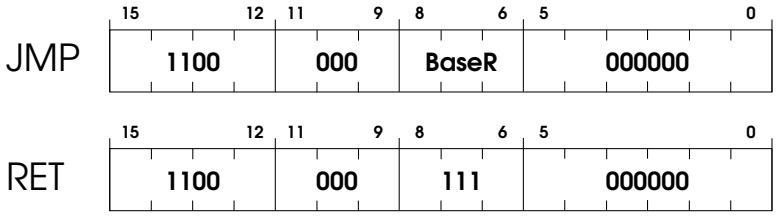
JMP — Jump

RET — Return from Subroutine

Assembler Formats

```
JMP    BaseR
RET
```

Encoding



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. The target of the JMP will be treated as a word-aligned address. PC[0] will always be zero.

Examples

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

Note

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.

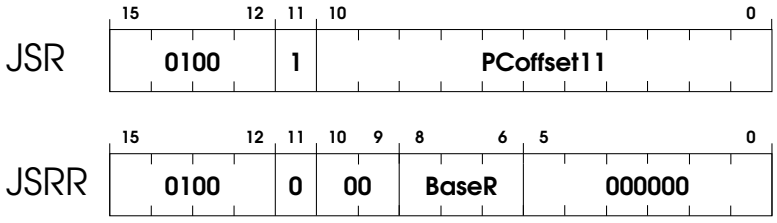
JSR
JSRR

Jump to Subroutine

Assembler Formats

```
JSR    LABEL
JSRR   BaseR
```

Encoding



Operation

```
R7 = PC†;
if (bit[11] == 0)
  PC = BaseR;
else
  PC = PC† + (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 and left-shifting bits [10:0] and 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.
```

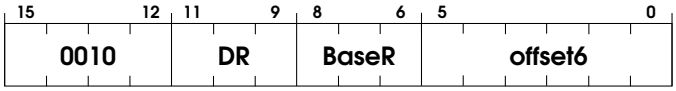
† This is the incremented PC.

LDB Load Byte

Assembler Format

LDB DR, BaseR, offset6

Encoding



Operation

DR = ZEXT(mem[BaseR + SEXT(offset6)]);
setcc();

Description

An address is computed by sign-extending bits [5:0] to 16 bits and adding this value to the contents of the register specified by bits [8:6]. The byte contents of memory at this address are zero-extended to 16 bits and loaded into DR. The condition codes are set, based on whether the value loaded is negative, zero, or positive.

Example

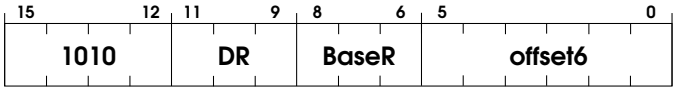
LDB R4, R2, #-5 ; R4 ← byte contents of mem[R2 - 5]

LDI Load Word Indirect

Assembler Format

LDI DR, BaseR, offset6

Encoding



Operation

DR = memWord[memWord[BaseR + (SEXT(offset6) << 1)]];
setcc();

Description

An address is computed by sign-extending bits [5:0] to 16 bits, left-shifting this value by 1 bit, and adding this result to the contents of the register specified by bits [8:6]. The word contents of memory at this address is the address of the 16-bit word data to be loaded into DR. The condition codes are set, based on whether the value loaded is negative, zero, or positive. The memory address specified by Base+offset will be treated as a word-aligned address. In other words, bit [0] of the address will be treated as if it is 0.

Example

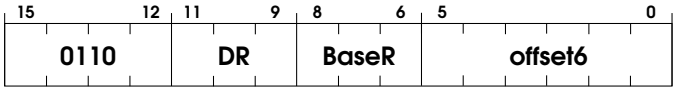
LDI R4, R2, #10 ; R4 ← memWord[memWord[R2 + 20]]

LDR Load Word

Assembler Format

LDR DR, BaseR, offset6

Encoding



Operation

DR = memWord[BaseR + (SEXT(offset6) << 1)];
setcc();

Description

An address is computed by sign-extending bits [5:0] to 16 bits, left-shifting this value by 1 bit, and adding this result to the contents of the register specified by bits [8:6]. The 16-bit word at this address is loaded into DR. The condition codes are set, based on whether the value loaded is negative, zero, or positive. The memory address specified by Base+offset will be treated as a word-aligned address. In other words, bit [0] of the address will be treated as if it is 0.

Example

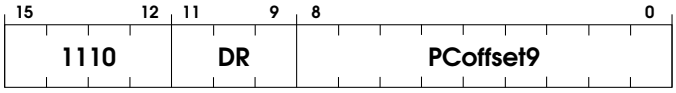
LDR R4, R2, #-5 ; R4 ← memWord[R2 - 10]

LEA — Load Effective Address

Assembler Format

LEA DR, LABEL

Encoding



Operation

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

Description

An address is computed by sign-extending bits [8:0] to 16 bits, left-shifting this value by 1 bit, and adding this result to the incremented PC. This address is loaded into DR.‡ The condition codes are set, based on whether the value loaded is negative, zero, or positive.

Example

```
LEA     R4, TARGET      ; R4 ← address of TARGET.
```

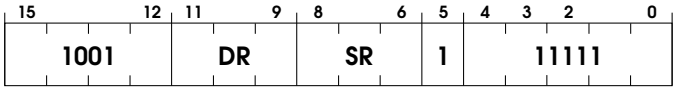
†This is the incremented PC.
‡The LEA instruction does not read memory to obtain the information to load into DR. The address, itself, is loaded into DR.

NOT — Bitwise Complement

Assembler Format

```
NOT     DR, SR
```

Encoding



Operation

```
DR = NOT(SR);
setcc();
```

Description

The bitwise complement of the contents of SR are 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

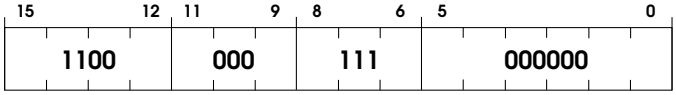
```
NOT     R4, R2      ; R4 ← NOT(R2)
```

RET[†] Return from Subroutine

Assembler Format

RET

Encoding



Operation

PC = R7;

Description

The PC is loaded with the value in R7. This causes a return from a previous JSR, JSRR, or TRAP instruction. A return address in R7 will be treated as a word-aligned address. PC[0] will always be zero.

Example

```
RET          ; PC ← R7
```

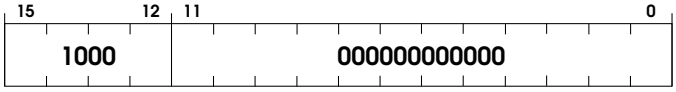
[†]The RET instruction is a specific encoding of the JMP instruction. See also JMP.

RTI Return from Interrupt

Assembler Format

RTI

Encoding



Operation

```
if (PSR[15] == 0) {
 PC = memWord[R6]; // R6 is the SSP
 R6 = R6 + 2;
 TEMP = memWord[R6];
 R6 = R6 + 2;
 PSR = TEMP; // the privilege, condition codes of the interrupted process are restored
 }
else
 Initiate a Privilege Mode Exception;
```

Description

If the processor is running in Supervisor mode, the top two elements on the Supervisor Stack are popped and loaded into PC, PSR. If the processor is running in User mode, a privilege mode violation exception occurs. Section 1.4 describes what happens in this case.

Example

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

Note

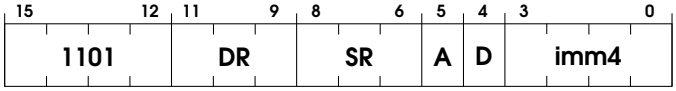
On an external interrupt, the initiating sequence first changes the privilege mode to Supervisor mode (PSR[15]=0). Then the PSR and PC of the interrupted process are pushed onto the Supervisor Stack before loading the PC with the starting address of the interrupt service routine. The interrupt service routine runs with Supervisor privilege. The last instruction in the service routine is RTI, which returns control to the interrupted program 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 1.4.

SHF Bit Shift

Assembler Formats

```
LSHF DR, SR, imm4     ;   left shift
RSHFL DR, SR, imm4    ;   right shift logical
RSHFA DR, SR, imm4    ;   right shift arithmetic
```

Encodings



Operation

```
if (D == 0)
  DR = SR << imm4;
else
  if (A == 0)
    DR = SR >> imm4,0;
  else
    DR = SR >> imm4,SR[15];
setcc();
```

Description

If the D bit (bit [4]) is 0, the source operand in SR is shifted left by the number of bit positions indicated by the imm4 field. If D is 1, the source operand is shifted to the right by imm4 bits. When shifting to the right, the A bit (bit [5]) of the instruction indicates whether the sign bit of the original source operand is preserved. When A is set to 1, the right shift is an arithmetic shift and the original SR[15] is shifted into the vacated bit positions. The result stored in DR. Otherwise the shift is a logical shift and zeroes are shifted in. The condition codes are set, based on whether the result is negative, zero, or positive.

Examples

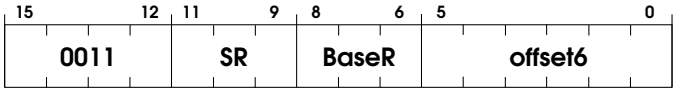
```
LSHF    R2, R3, #3    ;   R2 ← R3 << #3
RSHFL   R2, R3, #7    ;   R2 ← R3 >> #7,0
RSHFA   R2, R3, #7    ;   R2 ← R3 >> #7,R3[15]
```

STB Store Byte

Assembler Format

STB SR, BaseR, offset6

Encoding



Operation

mem[BaseR + SEXT(offset6)] = SR[7:0];

Description

The lower 8 bits of the register specified by SR (SR[7:0]) are stored at the memory location whose address is computed by sign-extending bits [5:0] to 16 bits and adding the result to the contents of the register specified by bits [8:6].

Example

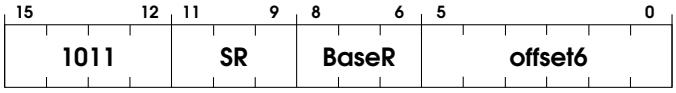
STB R6, R3, #-6 ; mem[R3 - 6] ← R6[7:0]

STI Store Word Indirect

Assembler Format

STI SR, BaseR, offset6

Encoding



Operation

memWord[memWord[BaseR + (SEXT(offset6) << 1)]] = SR;

Description

The contents of the register specified by SR are stored starting at the memory location whose address is computed as follows: Bits [5:0] are sign-extended to 16 bits, then left-shifted by 1 bit, and added to the contents of the register specified by bits [8:6]. The 16-bit word contents of memory at starting at this address is the address of the location to which the data in SR is stored. The memory address specified by Base+offset will be treated as a word-aligned address. In other words, bit [0] of the address will be treated as if it is 0.

Example

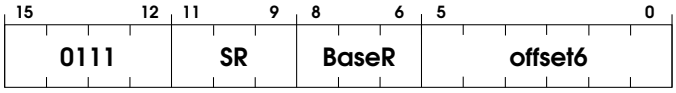
STI R4, R2, #10 ; memWord[memWord[R2 + 20]] ← R4

STR Store Word

Assembler Format

STR SR, BaseR, offset6

Encoding



Operation

memWord[BaseR + (SEXT(offset6) << 1)] = SR;

Description

The contents of the register specified by SR are stored starting at the memory location whose address is computed by sign-extending bits [5:0] to 16 bits, left-shifting this value by 1 bit, and adding the result to the contents of the register specified by bits [8:6]. The memory address specified by Base+offset will be treated as a word-aligned address. In other words, bit [0] of the address will be treated as if it is 0.

Example

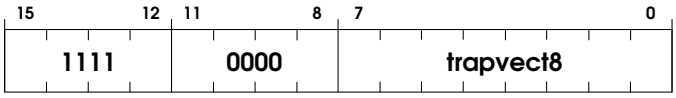
STR R4, R2, #5 ; memWord[R2 + 10] ← R4

TRAP System Call

Assembler Format

TRAP trapvector8

Encoding



Operation

R7 = PC†;
PC = memWord[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 the PC is loaded with the contents of the memory location obtained by zero-extending trapvector8 to 16 bits and left-shifting by 1 bit. This is the starting address of the system call specified by trapvector8.

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 x00FF are available to contain starting addresses for system calls specified by their corresponding trap vectors. This region of memory is called the trap vector table. Thus there are 128 trap vectors. As a result, the most significant bit of the trapvector8 number is ignored. Table 1.2 describes the functions performed by the service routines corresponding to trap vectors x20 to x25.

† This is the incremented PC.

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 x0000 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 monitor, 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 1.2: Trap Service Routines

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 1.3: Device register assignments

1.4 Interrupt Processing

Events external to the program that is running are able to interrupt the processor. A common example of this is interrupt-driven I/O. It is also the case that the processor can be interrupted by exceptional events that occur while the program is running that are caused by the program itself.

Associated with each event that can interrupt the processor is an 8-bit vector that provides an entry point into a 128 entry interrupt vector table (note: on the LC-3b the most significant bit of the 8-bit vector is ignored). The starting address of the interrupt vector table is x0100. That is, the interrupt vector table occupies memory locations x0100 to x01FF. Each entry in the interrupt vector table contains the starting address of the service routine that handles the needs of that event. These service routines execute in Supervisor mode.

Half (128) of these entries, locations x0100 to x017F, provide starting addresses of routines that service events caused by the running program itself. These routines are called Exception service routines because they handle exceptional events, that is, events that prevent the program from executing normally. The other half of the entries, locations x0180 to x01FF, provide starting addresses of routines that service events that are external to the program that is running, such as requests from I/O devices. These routines are called Interrupt service routines.

At this time, an LC-3b computer system provides only one I/O device that can interrupt the processor. That device is the keyboard. It interrupts at priority level PL4, and supplies the interrupt vector x80.

An I/O device can interrupt the processor if it wants service, if its Interrupt Enable (IE) bit is set, and if the priority of its request is greater than the priority of the program that is running.

Assume a program is running at a priority level less than four, and someone strikes a key on the keyboard. If the IE bit of the KBSR is 1, the currently executing program is interrupted at the end of the current instruction cycle. The interrupt service routine is `initiated` as follows:

1. The processor sets the privilege mode to Supervisor mode (PSR[15]=0).
2. R6 is loaded with the Supervisor Stack Pointer (SSP) if it does not already contain the SSP.
3. The PSR and PC of the interrupted process are pushed onto the Supervisor Stack.
4. The keyboard supplies its 8-bit interrupt vector, in this case x80.
5. The processor expands that vector to x0180, the corresponding 16-bit address in the interrupt vector table.
6. The PC is loaded with the contents of memory location x0180, the address of the first instruction in the keyboard interrupt service routine.

The processor then begins execution of the interrupt service routine.

The last instruction executed in an interrupt service routine is RTI. The top two elements of the Supervisor Stack are popped and loaded into the PC and PSR registers. R6 is loaded with the appropriate stack pointer, depending on the new value of PSR[15]. Processing then continues where the interrupted program left off.