Machine Language Specification

Nicholas Prowse

April 2023

The Computer

This document provides the specification for the machine language of the HADLoC Mk I (**H**arvard **A**rchitecture **D**iscrete **Lo**gic **C**omputer). HADLoC is an 8 bit computer constructed primarily out of 7400 series TTL integrated circuits. It contains 32KB of ROM, which is preloaded with the machine code of the program to be executed. The ROM is a AT28C256 EEPROM and must be removed from the computer to be programmed on a separate device. There is also 32KB of RAM for the program to use as memory.

There are 2 data registers; X and Y, and 2 address registers; Low (L) and High (H). Memory addresses are 15 bits, with the lower 8 bits stored in L and the upper 7 bits stored in H. L and H are also used as the destination address for jump instructions. We will also define M to be the memory location in RAM pointed to by L and H. In many situations M is treated just like the data registers, X and Y.

Input can be provided to the computer through 8 data lines. There is also a 9th input called the trigger which is pulsed when there is new data available on the data lines. The input data is stored in an in (I) register. When trigger is pulsed, the data on the input lines is stored in I and an 'in flag' (IF) is set. The computer is able to conditionally jump on this flag, allowing for the computer to know when new data is available. Note that IF is set on the rising edge of the trigger pulse, so if is held high for an extended period of time, the computer will only register one input even if the input is checked multiple times. This ensures that a single button press, which may last for thousands of instruction executions, is detected as a single input. In the HADLoC Mk I the 8 data lines are connected to an 8 way DIP switch and the trigger is connected to a debounced button, however these could be connected to any input device capable of communicating through 8 bits, with a clock line connected to the trigger. When the computer reads I, then IF is reset. This means an external device could read IF and wait until it is reset before writing more data, ensuring that data isn't written too fast for HADLoC.

The computer has a 4×20 character LCD screen controlled by a HD44780 dot-matrix liquid crystal display controller. This controller is controlled by 8 data bits and 1 control bit. The control bit is specified in the instruction, and instructs the display if the data provided is data, or an instruction for the display. The data for the display can be loaded from I, L, X or M.

Notation Conventions

A full list of notation conventions used in this document is shown in the table below

$A \leftarrow B$	The value B is stored into the register A
A_n	The nth bit of A, where $n = 0$ refers to the least significant bit
$A_{m..n}$	Consecutive bits m through n of the value stored in register A
IMMn	An n bit constant, embedded in a machine instruction
$A : B$	Bitwise concatenation of A and B. For example, $0101 : 1001 = 01011001$
Mem$[A]$	The byte located in RAM at address A
$\neg A$	The bitwise logical NOT (one's complement) of A
$A \wedge B$	The bitwise logical AND of A and B
$A \vee B$	The bitwise logical OR of A and B
$A + B$	The arithmetic addition of A and B
$-A$	The negative (two's complement) of A: $-A \equiv \neg A + 1$
$A - B$	The subtraction of B from A, using two's complement addition: $A - B \equiv A + 1 + (\neg B)$

Memory, Registers and Flags

The computer has 2 memory segments, 6 registers and 2 flags.
There are 2 address registers, 2 data registers and a register to store input to the computer

Y	– 8 bit data register – Data can be moved in and out of Y
X	– 8 bit data register – Data can be moved in and out of X – Arithmetic and logic calculations can be performed on X, and the results can be stored in X – Used as the condition in conditional jumps
L	– 8 bit general purpose register – When addresses are required, L is used for the $\underline{L}$ower 8 bits of the address – Data can be moved in and out of L – Arithmetic and logic calculations can be performed on L, and the results can be stored in L – Immediate values embedded in instructions can be loaded into L
H	– 7 bit address register – Data can be moved into, but not out of H – When addresses are required, H is used for the $\underline{H}$igher 7 bits of the address
PC	– The program counter (PC) is a 15 bit register, used to address the instruction currently being executed – PC is incremented once each clock cycle – Addresses stored in L and H can be conditionally moved into PC allowing the program to conditionally branch to another point in the program – Data cannot be moved out of PC
I	– I is an 8 bit $\underline{I}$nput register – When new data from the external input device is available, it is stored in this register and the input flag IF is set – Data can be moved out of I, but cannot be moved into I programmatically – When data is moved out of I, IF is reset

The two memory segments are ROM and RAM

ROM	– ROM is a 32768 byte memory and contains the program to be executed – It is addressed by the 15 bit program counter and cannot be written to – Each clock cycle, the program counter is incremented, and the instruction stored in ROM at the location addressed by PC is executed
RAM	– RAM is a 32768 byte memory, used to store data used by the program – RAM is addressed by L and H. The address is given by $H_{6..0} : L$ – Data can be moved into RAM from L, X, Y and I – Data can be moved out of RAM, into L, X, Y and H – The contents of RAM can directly be used in arithmetic and logic calculations

For the remainder of this document we will define M to the the contents of RAM addressed by L and H

$$M = \text{Mem}[H_{6..0} : L]$$

The two flags are the input flag (IF) and the carry flag (CF)

IF	– IF indicates if there is new data available in I – When data is moved out of I, IF is reset, allowing a program to know if the input data has been read – The computer will accept new data when IF is set, so it is up to external devices to monitor this flag, and not send new data if IF is set – Jumps can be conditionally executed based on IF
CF	– CF indicates in the last arithmetic command resulted in overflow – CF is only changed during arithmetic commands – Due to the internal method of calculating some of the arithmetic functions, CF is sometimes inverted from what would be expected. See the Arithmetic/Logic section for more information – Jumps can be conditionally executed based on CF – H can be conditionally incremented based on CF

The Language

Each instruction is 8 bits long and there are 8 distinct types of instructions, some of which also take arguments. The type of instruction is determined by the number of leading zeroes at the start of the instruction. Below, the table shows the format of the 8 instructions. Here we have used the notation b_n to represent the nth bit of the instruction, where b_0 is the least significant bit.

Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Description
Load byte	1	IMM7							Loads IMM7 into L. The most significant bit of L gets set to 0
Arithmetic/Logic	0	1	x	m	Op Code				Performs an arithmetic/logic calculation, such as addition, bitwise and, etc.
Move	0	0	1	s	Dest		Source		Moves the contents of one register into another register
Jump	0	0	0	1	s	Op Code			Jumps to the address stored in L and H if the given condition is met
Output	0	0	0	0	1	d	Source		Outputs the data stored in the register specified by Source to the display
No Operation	0 0	0 0	0 0	0 0	0 0	1 0	$\times$ 0	$\times$ 1	Does nothing. Use this command to create delays in execution
Carry	0	0	0	0	0	0	1	v	Increments H if the carry flag is equal to v
Halt	0	0	0	0	0	0	0	0	Halts execution of the computer and will only restart after a manual reset

Note: $\times$ indicates the value of that bit doesn't matter.

1 Load Byte

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	IMM7						

Operation:

$L \leftarrow IMM7$

Description:

The load byte command loads a 7 bit immediate value, embedded in the instruction, into L. The most significant bit of L gets set to 0. To load an 8 bit value into L, one must first load the one's complement of the value into L, then invert L using a bitwise logical not instruction.

Examples:

- 10000101 — Loads the value 5 into L
- 10000000 — Loads the value 0 into L
- 10010100 — Loads the value 20 into L

2 Arithmetic/Logic

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	x	m	Op Code			

Operation:

$R \leftarrow f(X, B)$

where R and B are determined by x and m

$$R = \begin{cases} L & \text{if } x = 0 \\ X & \text{if } x = 1 \end{cases}$$

$$B = \begin{cases} L & \text{if } m = 0 \\ M & \text{if } m = 1 \end{cases}$$

and $f(X, B)$ is determined by the Op Code according to the following table

Op Code	$f(X, B)$	Op Code	$f(X, B)$
0000	$\neg X$	0011	$\neg B$
1000	$-X$	1111	$-B$
1100	$X + 1$	1011	$B + 1$
0100	$X - 1$	0111	$B - 1$
1101	$X - B$	0101	$B - X$
1010	$X \wedge B$	1110	$X \vee B$
1001	$X + B$	0001	$\neg X \vee B$
0110	$\neg(X \wedge B)$	0010	$\neg(X \wedge B)$

Description:

Arithmetic/Logic commands (AL commands) perform a calculation with two registers as arguments, and stores the result in a third register. One argument is always X, while the second argument is L if $m = 0$ or M if $m = 1$. The result is stored in L if $x = 0$ or X if $x = 1$. The 4 bit Op Code determines what calculation is done. There are a total of 15 different functions that can be performed, shown in the table above.

If the command is an arithmetic command, then the carry flag will be set if the calculation results in overflow. The jump command can conditionally jump on this flag, allowing programs to detect when calculations have overflown. For simple addition and subtraction, this works as one would expect, but for other arithmetic commands, such as increment/decrement calculations, the conditions that produce overflow are inverted, due to the way these calculations are performed internally in the computer. The full list of conditions required to set the carry flag are shown in the table below, where X and B are treated as unsigned bytes. If these conditions are not met, the carry flag is reset. If the op code being used is not one shown in the table, the carry flag will neither be set or reset.

Op Code	$f(X, B)$	CF is set if
0010	$-X$	$X \neq 0$
1111	$-B$	$B \neq 0$
0011	$X + 1$	$X \neq 255$
1110	$B + 1$	$B \neq 255$
0001	$X - 1$	$X \neq 0$
1101	$B - 1$	$B \neq 0$
0110	$X + B$	$X + B > 255$
0111	$X - B$	$X < B$
0101	$B - X$	$B < X$

Examples:

- 01101010 — Calculate the bitwise and of L and X and store the result in X
- 01001110 — Increment L and store the result in L
- 01010111 — Subtract M from X and store the result in L

3 Move

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	s	Dest		Source	

Operation:

$A \leftarrow B$

where A and B are registers determined by s, Dest and Source according to the following table

s	Dest	A	Source	B
$\times$	00	X	00	X
$\times$	01	L	01	L
$\times$	10	H	10	I
0	11	Y	11	M
1	11	M	11	Y

Note: $\times$ indicates the value of that bit doesn't matter

Description:

The move command moves the contents of one register to another.

Data should only be moved out of I if IF is set, but this isn't enforced. This can be checked by using a conditional jump, with IF as the condition (See the Jump section). The I register can be accessed when IF is not set, and it will contain the last input received, but this should not be relied upon, because new data can arrive at any time. When data is moved out of I using a move command, IF is reset.

Note that data can be moved into H, but not out. So H cannot be used to store data for later use, as it cannot be retrieved.

Examples:

- 00100010 — Moves I into X and resets the input flag
- 00101001 — Moves L into H
- 00101111 — Moves Y into M

4 Jump

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	1	s	Op Code		

Operation:

if C
  then $PC \leftarrow H_{6..0} : L$
  else $PC \leftarrow PC + 1$

where C is a condition determined by s and the Op Code according to

Op Code	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
$s = 0$	False		CF		IF		$CF \vee IF$	
$s = 1$	False	$X > 0$	$X = 0$	$X \geq 0$	$X < 0$	$X \neq 0$	$X \leq 0$	True

Description:

The jump command allows program execution to conditionally jump to any other point in the program. Program execution jumps to the address stored in H and L, where H contains the upper 7 bits and L contains the lower 8 bits of the address.

The s bit in the instruction selects whether the condition is based upon the two flags or on the value of X. Each bit on the op code represents a condition. Setting a given bit to 1 means that condition is checked, and setting it to 0 means that condition is ignored. Multiple bits in the op code can be set to 1, indicating that any one of those conditions being true will result in a jump. If no conditions are set, then a jump will not occur, meaning this is effectively a no operation (nop). However, to maintain consistency, this should not be used. Rather, the No Operation command should be used for nops.

If $s = 0$, then the two flags are used as the conditions. By setting the second bit of the op code to 1, a jump will occur if CF is set. If the third bit of the op code is set then a jump will occur if IF is set. The first bit of op code is not used, so setting it to 0 or 1 has no effect.

If $s = 1$, then the value of X is used as the condition. The first bit of the op code checks if X is negative, then second bit checks if X is equal to 0, and the third bit checks of X is positive. Here, we are treating X as a 2's complement signed byte, so the values 0x01 to 0x7F are positive, while the values 0x80 to 0xFF are negative. These conditions can be combined to create other conditions. For example if the first and third bits are set, then a jump will occur if X is positive or negative, which is equivalent to $X \neq 0$.

An unconditional jump can be created by setting $s = 1$ and using the op code 111. This means a jump will occur if X is positive, negative or equal to 0. Clearly, this is always true, meaning this is an unconditional jump.

Examples:

- 00011110 — Jump if $X \leq 0$
- 00010100 — Jump if IF is set
- 00010010 — Jump if CF is set

5 Output

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	1	d	Source	

Operation:

Writes the data in R to the output, with control bit set to d, where R is determined by Source

Source	R
00	X
01	L
10	I
11	M

Description:

The output command sends the data stored a register to the output. The output is a HD44780U LCD display. If $d = 1$, then the data is sent to the data register of the display, otherwise the data is sent to the instruction register (the d bit is connected to the RS pin of the LCD). The datasheet[1] contains all the information about how to interface with the display. Pages 23-27 are particularly useful as they describe the instructions that can be used.

Examples:

- 00001000 — Outputs the X register to the instruction register of the display
- 00001101 — Outputs the L register to the data register of the display
- 00001111 — Outputs the M register to the data register of the display

6 No Operation

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	1	$\times$	$\times$

OR

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	1

[1]The datasheet for the HD44780U LCD display is available at https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Operation:

Nothing

Description:

The no operation (nop) command does nothing, except the usual incrementing of PC. This can be useful to create delays. The amount of delay that is created is dependant on the clock speed, but a large delay can be created by putting a few nops inside of a loop that is executed many times.

There are 2 alternative bit patterns for a nop and they are both equally valid to use.

Examples:

- 00000100 — Does nothing
- 00000101 — Does nothing
- 00000001 — Does nothing

7 Carry

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	1	v

Operation:

if $v = CF$
  then $H \leftarrow H + 1$

Description:

The carry instruction conditionally increments H, depending on whether CF is set or not. Using the v bit the carry instruction can be set to increment if CF is equal to 0, or if it is equal to 1. The main purpose of the carry instruction is to increment the address. To increment the address, L is incremented, and if this results in overflow, then H must also be incremented. This can be done with the following two instructions

01001110 – Increment L
00000010 – Increment H if the previous instruction overflowed

Note: Since the increment instruction sets CF if there is *no* overflow (see the Arithmetic/Logic section), we use $v = 0$ here.

If we need to increment the address by an amount other than 1, then we use $v = 1$, because the addition instruction sets CF if there *is* an overflow. The example below adds 40 to the address

00100100 – Move L into X
10101000 – Load 40 into L
01000110 – Add L and X and store the result in L
00000011 – Increment H if the previous instruction overflowed

Examples:

- 00000010 — Increment H if $CF = 0$
- 00000011 — Increment H if $CF = 1$

8 Halt

Bit Fields:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0

Operation:

Halts execution

Description:

The halt command has no arguments and only serves a single function. This command will halt the clock which will stop the computer from executing any more instructions. Since no more instructions are executed, this cannot be undone programmatically. The only way to restart the clock is to reset the computer which will restart the program.

Examples:

- 00000000 — Halts execution of the program