USER'S MANUAL
for
8-bit Object File Tool
2022.01.26

Lucid Technologies
http://www.lucidtechnologies.info/
Email: info@lucidtechnologies.info

LEGAL STUFF

CONTENTS

1.0 Features

- Can load Intel HEX-record and Motorola S-record files
- Can display data in hexadecimal and ASCII character formats
- Can compare memory images of any two files
- Can change the load (base) address of any loaded file
- Can merge loaded files
- Can save any loaded file in either Intel HEX-record or Motorola S-record format
    - Execution address for S-record files can be changed

2.0 Introduction

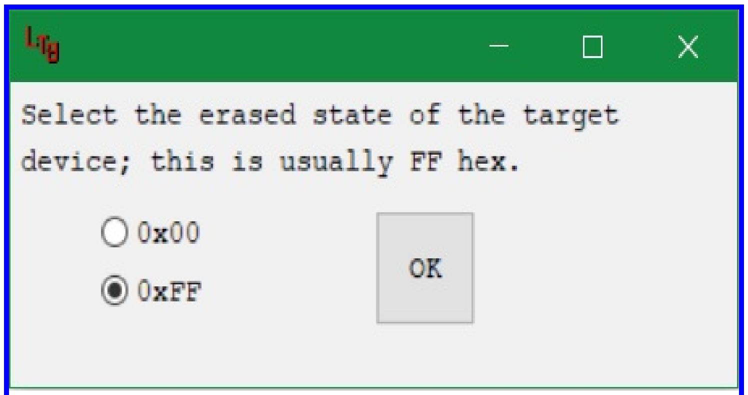
Lucid Technologies' 8-bit Object File Tool is was designed to work with object files intended for traditional 8-bit microcomputers and microcontrollers. It is a handy tool to convert files from Motorola S-record format to Intel HEX-record format and vice versa, to compare files to see if files of different names result in the same memory content, and to observe any ASCII text that may be embedded in the file.

3.0 System Requirements

Lucid Technologies' 8-bit Object File Tool was written for Windows systems. It has been tested on WindowsXP and Windows10. It is distributed as a ZIP archive. Create a new folder under Program Files, such as *C:\Program Files\8-bit tool*, and extract the contents of the ZIP archive to this directory. Click on *8-bit object file tool V300.exe* to run the program. The program files take approximately 3 MB of hard disk space.

4.0 Program Operation

Click on the executable file (*.exe) to open the program. The program maintains complete 64k memory images for both object files. These images are initialized to the erased value of the target EPROM or microcontroller. The first window that opens is for the selection of the erased value. Erased bytes in most devices read as 0xFF but some, such as the MC68701, read as 0x00. You may select either 0x00 or 0xFF. The erased value is assumed to be the same for both File 1 and File 2, otherwise comparing the two files would not make sense.


Select the erased state of the target device; this is usually FF hex.

0x00 [✓ 0xFF]

Figure 1 Erased value.

4.1 Program Windows

After selecting the erased value the program will open two windows: the Display window and the Control window.

The Display Window is a text output window. It is used to show errors that may be encountered while loading an object file, loaded data, or the result of comparing File 1 to File 2.

The Control Window has mouse clickable buttons that control the program. Initially the only active buttons on the Control window are the Load buttons for File 1 and File 2. After successfully loading a file the other buttons for that file will be enabled. Figure 2 shows the Control window after loading File 1. After both files are loaded the Compare and Merge buttons are enabled.

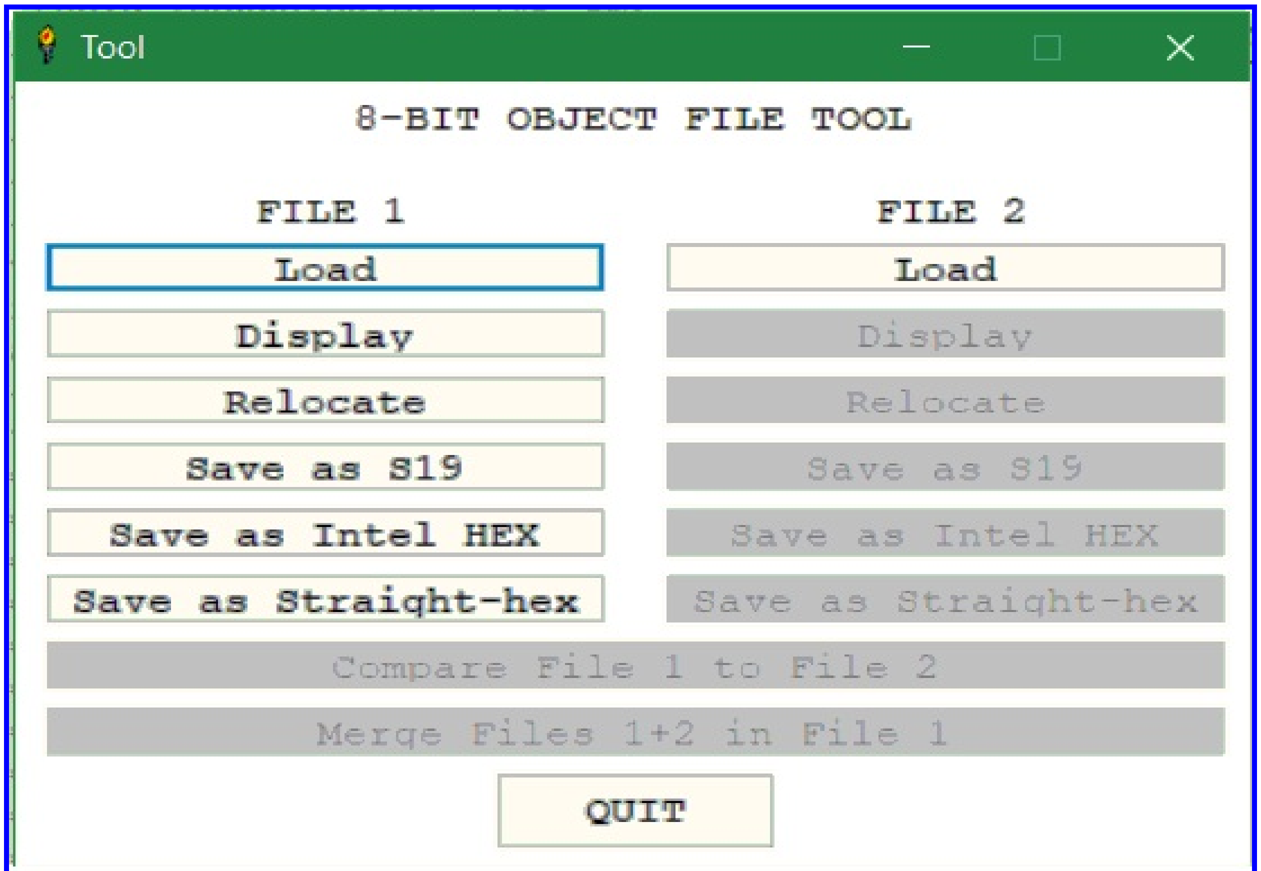


Figure 2 Control Window after loading File 1.

4.2 Load

Clicking on the Load button for either file will bring up a standard file selection window. You can select files whose extensions are HEX, S19 or TXT. The program automatically detects whether the file is of Intel-HEX or Motorola-S19 type, even if it has a TXT extension. The load process checks for several errors. If an error is encountered one of the following messages will be shown in the Display Window:

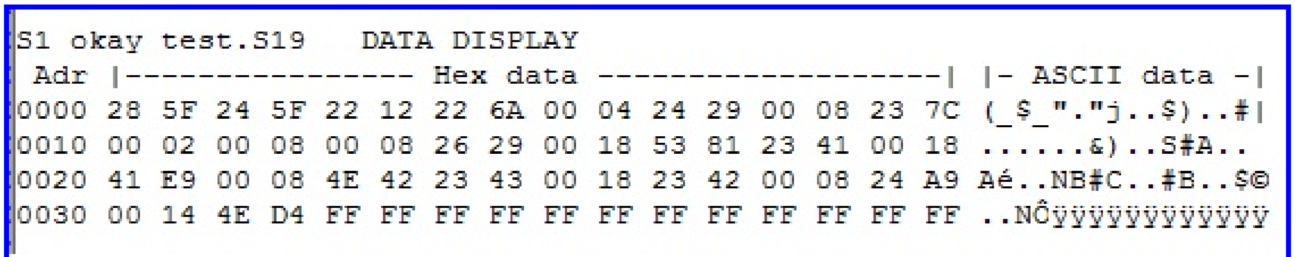
- "Not a valid object file!"
- "Invalid record type in line __ !"
- "End of file record is not last line of file!"
- "Incorrect check sum in line __ !"
- "Incorrect byte count in line __ !"

If the file loads successfully the remainder of the menu buttons for that file will be enabled.

The execution address from the termination record is stored for use in the Save as S19 option. The execution address for HEX-record files is always 0000; the execution address for S-record files may be any address in the 64k address space.

4.3 Display

Clicking on the Display button for either file will display the non-erased memory locations for that file in 16 byte segments. Any 16 byte segment in the 64k memory image with at least one byte that is not erased will be shown in the Display Window. Figure 3 shows the layout of the data display. The first line shows the file name and “DATA DISPLAY”. The second line is the column headers. The third and following lines are the actual address and data. In Figure 3 the third line shows the 16 bytes from address 0000 to 000F in hexadecimal format and then as ASCII characters.



```
S1 okay test.S19   DATA DISPLAY
 Adr |---------------- Hex data -------------------| |- ASCII data -|
0000 28 5F 24 5F 22 12 22 6A 00 04 24 29 00 08 23 7C (_$_"."j..$)..#|
0010 00 02 00 08 00 08 26 29 00 18 53 81 23 41 00 18 ......&)..S#A..
0020 41 E9 00 08 4E 42 23 43 00 18 23 42 00 08 24 A9 Aé..NB#C..#B..$©
0030 00 14 4E D4 FF FF FF FF FF FF FF FF FF FF FF FF ..NÔÿÿÿÿÿÿÿÿÿÿÿÿ
```

Figure 3 Data display for the file “S1 okay test.S19".

4.4 Relocate

Imagine you have a MC6803 8-bit processor with four different 2k boot programs (Boot1-4) stored in a 2764 EPROM. The top two address bits on the 2764 are used to select which 2k segment maps from F800 to FFFF. The four boot programs are all assembled to run at F800 but their locations in the EPROM will start at 0000 (Boot1), 0800 (Boot2), 1000 (Boot3) and 1800 (Boot4). The Relocate option would allow you to move a boot program’s code from a starting (base) address of F800 to any of the proper boundaries. Relocate can change the starting (base) address of a loaded program from any address to any other address, up or down, provided the program is still contained within the 64k address space. See 4.7, the Merge option.

4.5 Save as

Save as Intel HEX button. Clicking on the Save as Intel HEX button for either file will bring up a standard file selection window. Enter a file name, click on Save and the file will be saved in Intel HEX format. Each HEX-record in the saved file will have 16 data bytes. Any 16 byte segment in the 64k memory image with at least one byte that is not erased will be saved as a HEX-record.

Save as S19 button. Unlike the termination HEX-record (type 00) where the address is always zero, the address in a termination S-record (type S9) can have a non-zero address. This address is used for directly executable code, it is the 2-byte address of the instruction to which control is to be passed after the S-record file is loaded. Most assemblers don’t provide an option to set this address in the object file therefore it is handy to have a tool that can do this. After clicking on the Save as S19 button and entering the save file name you will be shown the current execution address for the file and asked if you want to change it, see Figure 4. If you respond Yes, you will be prompted to enter an new execution address as a four character hexadecimal value, see Figure 5.

Valid characters are 0-9, a-f, and A-F.

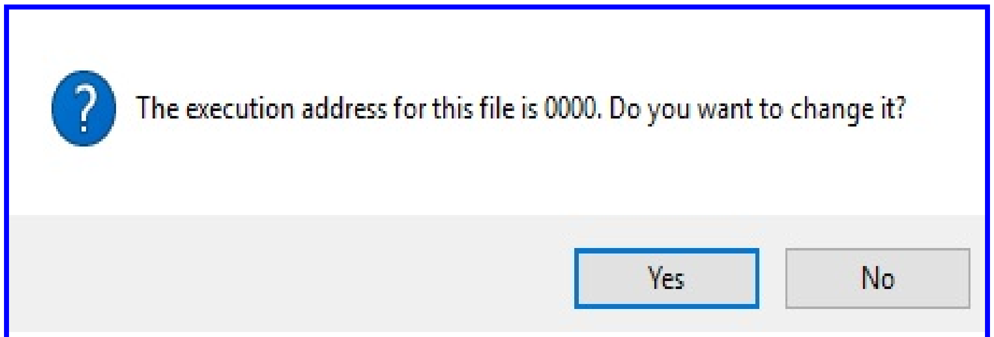

The execution address for this file is 0000. Do you want to change it?
Yes
No

Figure 4 Prompt for option to change S-record file execution address.

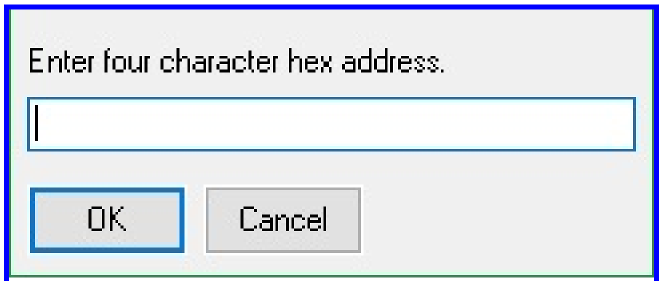
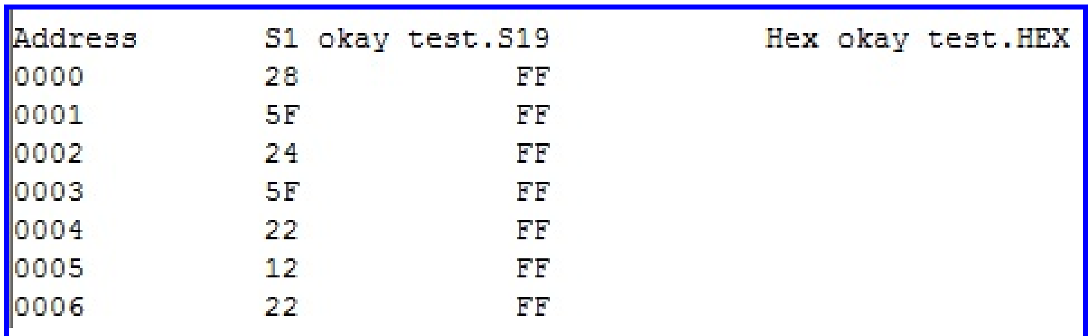


Figure 5 Prompt for new execution address.

Save as Straight-hex button. Clicking on the Save as Straight-hex button for either file will bring up a standard file selection window. Enter a file name, click on Save and the file will be saved in Straight-hex format with a SHX extension. Straight-hex files contain only the program data in ASCII hex format, see Appendix D. Because they do not contain address data, straight-hex files cannot be reloaded by the 8-bit Tool program, but can be used by simple device programmers.

4.6 Compare

The Compare File 1 to File 2 button will be only be enabled when both files are successfully loaded. Clicking on the Compare button will show the first line of Figure 6 in the Display Window. If no differences are found the message “No differences found” will be shown. In the example of Figure 6 we see seven differences at addresses 0000-0006. At address 0000 File 1 (S1 okay test.S19) is 0x28 while File 2 (Hex okay test.HEX) is 0xFF.



```
Address          S1 okay test.S19           Hex okay test.HEX
0000             28        FF
0001             5F        FF
0002             24        FF
0003             5F        FF
0004             22        FF
0005             12        FF
0006             22        FF
```

Figure 6 Compare files example.

4.7 Merge

We will continue the example used in section 4.4, Relocate. To generate a file to program a 2764 with four separate boot programs, first load Boot1 into File 1. Relocate File 1 to 0000. Now load Boot2 into File 2 and relocate it to 0800. Then select the Merge button. File 2 (Boot2) will be stacked on top of Boot1 in File 1. Next you can Load Boot3 into File 2, Relocate it to 1000, and Merge it with Boot1 and Boot 2 in File 1. Load Boot4 into File 2, Relocate it to 1800, and merge it with Boot1-3 in File 1. Finally, save the image for the entire 2674 EPROM from File 1 as either *.HEX or *.S19.

4.8 Quit

The Quit button will end the program.

5.0 Bugs, Suggestions and Donations

The most current version of 8-bit Object File Tool should always be available at www.lucid.technologies.info. If you discover any bugs or have suggestions for improving the program please send them to info@lucidtechnologies.info.

If you find 8-bit Object File Tool useful and would like to send a donation to Lucid Technologies you can do so via PayPal (http://www.lucidtechnologies.info/pay_card.htm).

APPENDIX A

ASCII CHARACTER CODES

The first 32 characters in the ASCII (American Standard Code for Information Interchange) table are unprintable control codes and are used to control peripherals such as printers.

ASCII control characters (character codes 0x00-0x1F)

DEC	HEX	Symbol	Description
0	0	NUL	Null char
1	1	SOH	Start of Header
2	2	STX	Start of Text
3	3	ETX	End of Text
4	4	EOT	End of Transmission
5	5	ENQ	Enquiry
6	6	ACK	Acknowledgment
7	7	BEL	Bell
8	8	BS	Backspace
9	9	HT	Horizontal Tab
10	0A	LF	Line Feed
11	0B	VT	Vertical Tab
12	0C	FF	Form Feed
13	0D	CR	Carriage Return
14	0E	SO	Shift Out
15	0F	SI	Shift In
16	10	DLE	Data Line Escape
17	11	DC1	Device Control 1 (often XON)
18	12	DC2	Device Control 2
19	13	DC3	Device Control 3 (often XOFF)
20	14	DC4	Device Control 4
21	15	NAK	Negative Acknowledgment
22	16	SYN	Synchronous Idle
23	17	ETB	End of Transmit Block
24	18	CAN	Cancel
25	19	EM	End of Medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	File Separator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator

Codes 32-127 decimal, 0x20-0x7F hexadecimal, are called the 7-bit or printable characters. They represent letters, digits, punctuation marks, and a few miscellaneous symbols. You will find almost every character on your keyboard.

DEC	HEX	Symbol	Description
32	20		Space
33	21	!	Exclamation mark
34	22	"	Double quotes
35	23	#	Number
36	24	$	Dollar
37	25	%	Percent
38	26	&	Ampersand
39	27	`	Single quote
40	28	(	Open parenthesis
41	29	)	Close parenthesis
42	2A	*	Asterisk
43	2B	+	Plus
44	2C	,	Comma
45	2D	-	Hyphen
46	2E	.	Period, dot or full stop
47	2F	/	Slash or divide
48	30	0	Zero
49	31	1	One
50	32	2	Two
51	33	3	Three
52	34	4	Four
53	35	5	Five
54	36	6	Six
55	37	7	Seven
56	38	8	Eight
57	39	9	Nine
58	3A	:	Colon
59	3B	;	Semicolon
60	3C	<	Less than
61	3D	=	Equals
62	3E	>	Greater than
63	3F	?	Question mark
64	40	@	At symbol
65	41	A	Uppercase A
66	42	B	Uppercase B
67	43	C	Uppercase C
68	44	D	Uppercase D
69	45	E	Uppercase E
70	46	F	Uppercase F
71	47	G	Uppercase G
72	48	H	Uppercase H
73	49	I	Uppercase I
74	4A	J	Uppercase J
75	4B	K	Uppercase K
76	4C	L	Uppercase L
77	4D	M	Uppercase M

DEC	HEX	Symbol	Description
78	4E	N	Uppercase N
79	4F	O	Uppercase O
80	50	P	Uppercase P
81	51	Q	Uppercase Q
82	52	R	Uppercase R
83	53	S	Uppercase S
84	54	T	Uppercase T
85	55	U	Uppercase U
86	56	V	Uppercase V
87	57	W	Uppercase W
88	58	X	Uppercase X
89	59	Y	Uppercase Y
90	5A	Z	Uppercase Z
91	5B	[	Opening bracket
92	5C	\	Backslash
93	5D	]	Closing bracket
94	5E	^	Caret - circumflex
95	5F	_	Underscore
96	60	`	Grave accent
97	61	a	Lowercase a
98	62	b	Lowercase b
99	63	c	Lowercase c
100	64	d	Lowercase d
101	65	e	Lowercase e
102	66	f	Lowercase f
103	67	g	Lowercase g
104	68	h	Lowercase h
105	69	I	Lowercase I
106	6A	j	Lowercase j
107	6B	k	Lowercase k
108	6C	l	Lowercase l
109	6D	m	Lowercase m
110	6E	n	Lowercase n
111	6F	o	Lowercase o
112	70	p	Lowercase p
113	71	q	Lowercase q
114	72	r	Lowercase r
115	73	s	Lowercase s
116	74	t	Lowercase t
117	75	u	Lowercase u
118	76	v	Lowercase v
119	77	w	Lowercase w
120	78	x	Lowercase x
121	79	y	Lowercase y
122	7A	z	Lowercase z
123	7B	{	Opening brace
124	7C	|	Vertical bar
125	7D	}	Closing brace
126	7E	~	Equivalency sign - tilde
127	7F		Delete

APPENDIX B

INTEL HEX-RECORD FORMAT

INTRODUCTION

Intel's Hex-record format allows program or data files to be encoded in a printable (ASCII) format. This allows viewing of the object file with standard tools and easy file transfer from one computer to another, or between a host and target. An individual Hex-record is a single line in a file composed of many Hex-records.

HEX-RECORD CONTENT

Hex-Records are character strings made of several fields which specify the record type, record length, memory address, data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first ASCII character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 6 fields which comprise a Hex-record are defined as follows:

Field		**Characters**	**Description**
1	Start Code	1	An ASCII colon, ":".
2	Byte Count	2	The number of bytes (character pairs) in the data field. Byte Count is typically 16.
3	Address	4	The 2-byte address at which the data field is to be loaded into memory.
4	Type	2	00 or 01.
5	Data	2(Byte Count)	Each data byte is represented in hexadecimal format by two ASCII characters.
6	Checksum	2	The least significant byte of the two's complement sum of all the bytes in the record except the Start Code and Checksum.

Each record may be terminated with a CR/LF/NULL. Accuracy of transmission is ensured by the byte count and checksum fields. Because the data field is composed of character pairs, each pair being one 8-bit byte, Intel Hex-record files are commonly used with 8-bit processors. Because the address field is only four characters, Intel Hex-record files are restricted to a 64 kilobyte address space.

HEX-RECORD TYPES

For 8-bit processors there are two possible types of Hex-records.

00	A record containing data and the 2-byte address at which the data is to reside.
01	A termination record for a file of Hex-records. Only one termination record is allowed per file and it must be the last line of the file. There is no data field.

HEX-RECORD EXAMPLE

Following is a typical Hex-record module consisting of four data records and a termination record.

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

The first data record is explained as follows:

: Start code.

10 Hex 10 (decimal 16), indicating 16 data character pairs,
16 bytes of binary data, in this record.

01 Four-character 2-byte address field: hex address 0100,
00 indicates location where the following data is to be loaded.

00 Record type indicating a data record.

The next 16 character pairs are the ASCII bytes of the actual program data.

40 Checksum of the first Hex-record.

The termination record is explained as follows:

: Start code.

00 Byte count is zero, no data in termination record.

00 Four-character 2-byte address field, zeros.
00

01 Record type 01 is termination.

FF Checksum of termination record.

APPENDIX C

MOTOROLA S-RECORD FORMAT

INTRODUCTION

Motorola's S-record format for output modules was devised for the purpose of encoding programs or data files in a printable (ASCII) format. This allows viewing of the object file with standard tools and easy file transfer from one computer to another, or between a host and target. An individual S-record is a single line in a file composed of many S-records.

S-RECORD CONTENT

S-Records are character strings made of several fields which specify the record type, record length, memory address, data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first ASCII character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 5 fields which comprise an S-record are defined as follows:

Field		**Characters**	**Description**
1	Type	2	S-record type - S0, S1 or S9.
2	Record length	2	The count of the bytes (character pairs) in the record, excluding the type and record length.
3	Address	4	The 2-byte address at which the data field is to be loaded into memory.
4	Data	0-2n	From 0 to n bytes of executable code, or memory loadable data. The number of Data bytes (n) is typically 16.
5	Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers. Accuracy of transmission is ensured by the record length (byte count) and checksum fields. Because the data field is composed of character pairs, each pair being one 8-bit byte, S19 files are commonly used with 8-bit processors. Because the address field is only four characters, S19 files are restricted to a 64 kilobyte address space.

S-RECORD TYPES

For 8-bit processors there are only three possible types of S-records.

S0	A comment record; ignored by this program.
S1	A record containing data and the 2-byte address at which the data is to reside.
S9	A termination record for a file of S1-records. Only one S9-record is allowed per file and it must be the last line of the file. The address field for directly executable code may

optionally contain the 2-byte address of the instruction to which control is to be passed. For ROM data the S9 address field is usually 0000. There is no data field.

S-RECORD EXAMPLE
The following is a typical S-record file:

```
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The file consists of four S1 records and an S9 termination record. The first S1 data record is explained as follows:

S1 S-record type S1, indicating a data record to be loaded/verified at a 2-byte address.

13 Hex 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow; 2 bytes for address + 16 bytes for data + 1 byte checksum = 19 bytes.

00 Four-character 2-byte address field: hex address 0000,
00 indicates location where the following data is to be loaded.

The next 16 character pairs are the ASCII bytes of the actual program data.

2A Checksum of the first S1-record.

The second and third S1 data records also contain 0x13 character pairs each. The fourth S1 data record contains 7 character pairs.

The S9 termination record is explained as follows:

S9 S-record type S9, indicating a termination record.

03 Hex 03, indicating three character pairs (3 bytes) to follow.

00 Four-character 2-byte address field.
00

FC Checksum of S9-record.

APPENDIX D

STRAIGHT-HEX FORMAT

The straight-hex file format consists of two hex characters for each data byte in the file. The hex characters are separated into lines with a Carriage Return, Line Feed sequence. The file contains no address information nor any checksums. The 8-bit Tool software saves straight-hex files with an SHX suffix.

Here is a straight-hex file example consisting of one line. It contains the data "Hello, World" terminated by a Carriage Return (0D), Line Feed (0A) sequence.

48656C6C6F2C20576F726C640D0A

Broken out for clarity:

```
48  65  6C  6C  6F  2C  20  57  6F  72  6C  64  0D  0A
H   e   l   l   o   ,       W   o   r   l   d   CR  LF
```