5 Translation Formats

Introduction

When a data file (which contains information to be programmed into a device) is created, it is stored in a specific data translation format. When the data file is transferred to a programmer, it must be set to handle the appropriate translation format. During download, the programmer translates the formatted data and stores it in user memory as a binary image file.

The following translation formats are described in this chapter.

Format	Code	Page
ASCII-BNPF	01 (05*)	5-3
ASCII-BHLF	02 (06*)	5-3
ASCII-B10F	03 (07*)	5-3
Texas Instruments SDSMAC (320	04	5-4
5-level BNPF	08 (09*)	5-5
Formatted Binary	10	5-6
DEC Binary	11	5-7
Spectrum	12 (13*)	5-8
POF	14	5-9
Absolute Binary	16	5-11
LOF	17	5-12
ASCII - Octal Space	30 (35*)	5-14
ASCII - Octal Percent	31 (36*)	5-14
ASCII - Octal		5-14
Apostrophe	32	5-14
ASCII - Octal SMS	37	5-14
ASCII - Hex Space	50 (55*)	5-14
ASCII - Hex Percent	51 (56*)	5-14
ASCII - Hex Apostrophe	52	5-14
ASCII - Hex SMS	57	5-14
ASCII - Hex Comma	53 (58*)	5-14
RCA Cosmac	70	5-16
Fairchild Fairbug	80	5-17
MOS Technology	81	5-18
Motorola EXORcisor	82	5-19
Intel Intellec 8/MDS	83	5-20
Signetics Absolut Object	85	5-20
Tektronix Hexadecimal	86	5-21
Motorola EXORmax	87	5-22
Intel MCS-86 Hex Object	88	5-23
Hewlett-Packard 64000 Absolute	89	5-25
Texas Instruments SDSMAC	90	5-26
JEDEC Format (Full)	91	5-27 and 5-30
JEDEC Format (Kernel)	92	5-27 and 5-37
Tektronix Hexadecimal Extended	94	5-37
Motorola 32 bit (S3 Record)	95	5-39
Hewlett-Packard UNIX Format	96	5-40
Intel OMF 386	97	5-41
Intel OMF 286	98	5-42
Intel Hex-32	99	5-44

* *This alternate code is used to transfer data without the STX start code and the ETX end code.*
** *This alternate code is used to transfer data using the SOH start code instead of the usual STX.*

Instrument Control Codes

The instrument control code, a 1-digit number that signals or controls data transfers, can be used to provide peripherals with flow control beyond that provided by software handshaking. The instrument control code is sent immediately preceding the 2-digit format code in computer remote control. The three instrument control codes and their functions are described below.

Code	Input Function	Output Function
0 Handshake Off	Send X-OFF to stop incoming transmission. Send X-ON to resume transmission.	Data transmission is halted on receipt of an X-OFF character. Transmission resumes on receipt of an X-ON character.
1 Handshake On	Transmit an X-ON character when ready to receive data; transmit X-OFF if the receiver buffer is full; transmit an X-ON if the receiver buffer is empty; transmit an X-OFF after all the data is received.	Transmit a PUNCH-ON character prior to data transmission. Data transmission is halted on receipt of an X-OFF character and resumes on receipt of an X-ON character. A PUNCH-OFF character is sent when the transmission is completed.
2 X-ON/ X-OFF	Send X-OFF to stop the incoming transmission. Send X-ON to resume transmission.	Transmit data only after receiving an X-ON character. Data transmission will be halted upon receipt of an X-OFF character; transmission will resume upon receipt of an X-ON character.

Note: X-ON character is a CTRL-Q, or 11 hex.
X-OFF character is a CTRL-S, or 13 hex.
PUNCH-ON character is a CTRL-R, or 12 hex.
PUNCH-OFF character is a CTRL-T, or 14 hex.

General Notes

Compatibility
When translating data, you may use any remote source that produces formats compatible with the descriptions listed in this section.

Formats with Limited Address Fields
Some formats are not defined for use with address fields greater than 64K. If you transfer a block greater than 64K, the address fields that would be greater than 64K may wrap around and overwrite data transferred in previous data records. Formats 70 through 86, and 90 may exhibit this characteristic.

Hardware Handshaking
If compatible with the host interface, hardware handshaking may be used by connecting the appropriate lines at the serial port interface. Hardware handshake (CTS/DTR) is enabled as the default. If those signals aren't connected, however, the programming electronics communicates using software handshake (XON/XOFF). The programmer always uses software handshake regardless of whether hardware handshake is enabled.

Leader/Trailer

During output of all formats except 89 (HP 64000), a 50-character leader precedes the formatted data and a 50-character trailer follows. This leader/trailer consists of null characters. If null count is set to FF hex, the leader/trailer is skipped. To set the null count, go to More Commands/Configure/Edit/Communication Parameters, or use the **U** command when using CRC.

Note:Formats 10, 11, and 89 do not function properly unless you select NO parity and 8-bit data.

ASCII Binary Format, Codes 01, 02, and 03 (or 05, 06, and 07)

In these formats, bytes are recorded in ASCII codes with binary digits represented by Ns and Ps, Ls and Hs, or 1s and 0s respectively. See Figure 5-1. The ASCII Binary formats do not have addresses.

Figure 5-1 shows sample data bytes coded in each of the three ASCII Binary formats. Incoming bytes are stored sequentially in RAM starting at the first RAM address. Bytes are sandwiched between B and F characters and are separated by spaces.

Figure 5-1. ASCII Binary Format (example)

Data can also be expressed in 4-bit words. The programmer generates the 4-bit format on upload if the data word width is 4 bits. You can insert any other character, such as carriage returns or line feeds, between an F and the next B.

The start code is a nonprintable STX (a CTRL-B, same as a hex 02). The end code is a nonprintable ETX (a CTRL-C, same as a hex 03).

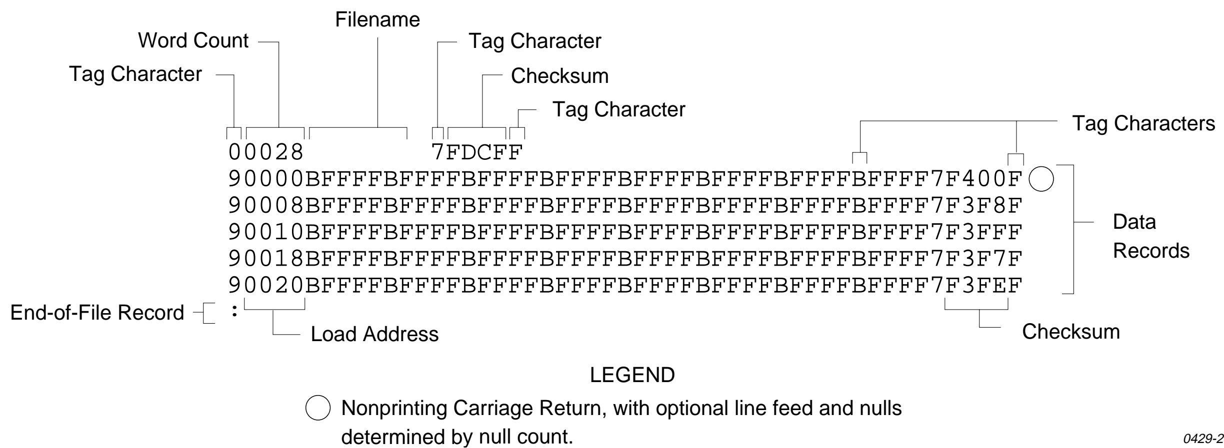
Note: Data without a start or end code may be input to or output from the programmer by use of alternate data translation format codes. These are ASCII-BNPF, 05; ASCII-BHLF, 06; ASCII-B10F, 07.

A single data byte can be aborted if the programmer receives an E character between B and F characters. Data will continue to be stored in sequential RAM addresses. Data is output in 4-byte lines with a space between bytes.

Texas Instruments SDSMAC Format (320), Code 04

Data files in the SDSMAC (320) format consist of a start-of-file record, data records, and an end-of-file record. See Figure 5-2. The format is used for Texas Instruments' 320 line of processors. It is very similar to format 90; the only difference is that the address fields represent 16-bit data words rather than bytes.

Figure 5-2. TI SDSMAC Format (example)



Each record is composed of a series of small fields, each initiated by a tag character. The programmer recognizes and acknowledges the following tag characters:

0 or K — followed by a file header.
7 — followed by a checksum which the programmer acknowledges.
8 — followed by a checksum which the programmer ignores.
9 — followed by a load address which represents a word location.
B — followed by 4 data characters (16-bit word).
F — denotes the end of a data record.
* — followed by 2 data characters.

The start-of-file record begins with a tag character and a 12-character file header. The first four characters are the word count of the 16-bit data words; the remaining file header characters are the name of the file and may be any ASCII characters (in hex notation). Next come interspersed address fields and data fields (each with tag characters). The address fields represent 16-bit words. If any data fields appear before the first address field in the file, the first of those data fields is assigned to address 0000. Address fields may be expressed for any data word, but none are required.

The record ends with a checksum field initiated by the tag character 7 or 8, a 4-character checksum, and the tag character F. The checksum is the two's complement of the sum of the 8-bit ASCII values of the characters, beginning with the first tag character and ending with the checksum tag character (7 or 8).

Data records follow the same format as the start-of-file record but do not contain a file header. The end-of-file record consists of a colon (:) only. The output translator sends a CTRL-S after the colon.

During download or input from disk operations, the destination address for the data is calculated in the following manner:

Memory address =
(load address x 2) — I/O address offset + begin address

During upload or output to disk operations, the load address sent with each data record is calculated in the following manner:

Load address = I/O address offset / 2

The Memory begin address, I/O address offset, and User data size parameters represent bytes and must be even values for this format. The upload record size must also be even for this format (default is 16).

Note: If the data will be programmed into a 16-bit device to be used in a TMS320 processor-based system, the odd/even byte swap switch must be enabled.

5-Level BNPF Format, Codes 08 or 09

Except for the start and end codes, the same character set and specifications are used for the ASCII-BNPF and 5-level BNPF formats.

Data for input to the programmer is punched on 5-hole Telex paper tapes to be read by any paper tape reader that has an adjustable tape guide. The reader reads the tape as it would an 8-level tape, recording the 5 holes that are on the tape as 5 bits of data. The 3 most significant bits are recorded as if they were holes on an 8-level tape. Tape generated from a telex machine using this format can be input directly to a serial paper tape reader interfaced to the programmer. The programmer's software converts the resulting 8-bit codes into valid data for entry in RAM.

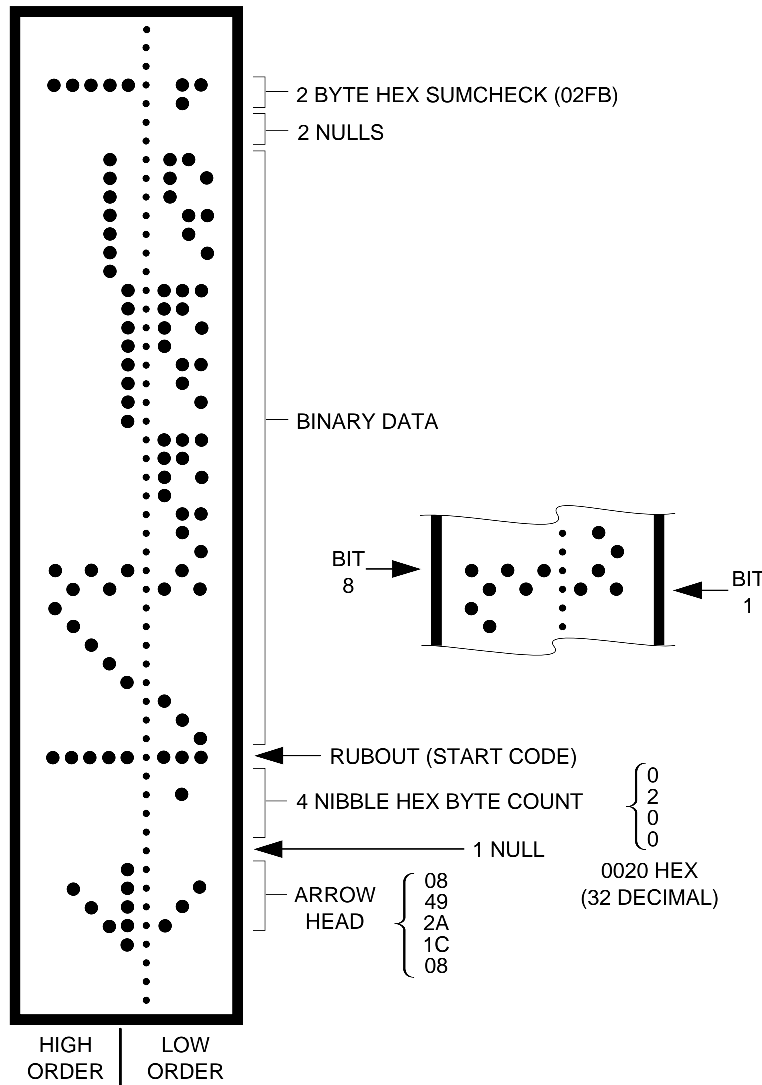
The start code for the format is a left parenthesis, (Figs K on a telex machine), and the end code is a right parenthesis, (Figs L on a telex machine). The 5-level BNPF format does not have addresses.

Note: Data without a start or end code may be input to or output from the programmer by use of the alternate data translation format code, 09. This format accepts an abort character (10 hex) to abort the transmission.

Formatted Binary Format, Code 10

Data transfer in the Formatted Binary format consists of a stream of 8-bit data bytes preceded by a byte count and followed by a sumcheck, as shown in Figure 5-3. The Formatted Binary format does not have addresses.

Figure 5-3. Formatted Binary Format (example)

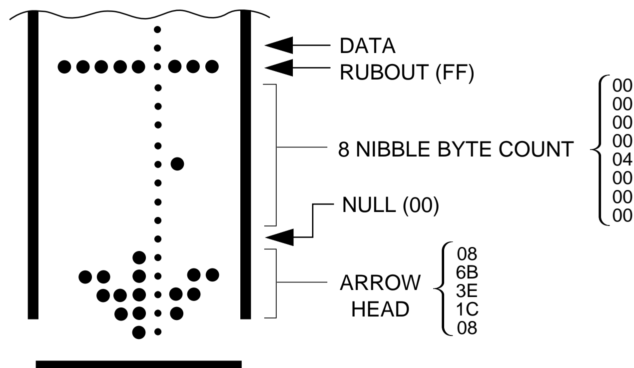


The programmer stores incoming binary data upon receipt of the start character. Data are stored in RAM starting at the first RAM address specified by the Memory Begin Address parameter and ending at the last incoming data byte.

A paper tape generated by a programmer contains a 5-byte, arrow-shaped header followed by a null and a 4-nibble byte count. The start code, an 8-bit rubout, follows the byte count. The end of data is signaled by two nulls and a 2-byte sumcheck of the data field. Refer to Figure 5-4.

If the data output has a byte count GREATER than or equal to 64K, an alternate arrow-shaped header is used. This alternate header (shown below) is followed by an 8-nibble byte count, sandwiched between a null and a rubout. The byte count shown here is 40000H (256K decimal). If the byte count is LESS than 64K, the regular arrowhead is used instead. Data that is input using Formatted Binary format will accept either version of this format.

Figure 5-4. Formatted Binary Format (example)



In addition, a third variation of this binary format is accepted on download. This variation does not have an arrowhead and is accepted only on input. The rubout begins the format and is immediately followed by the data. There is no byte count or sumcheck.

Format 10 does not function properly unless you select NO parity and 8-bit data.

DEC Binary Format, Code 11

Data transmission in the DEC Binary format is a stream of 8-bit data words with no control characters except the start code. The start code is one null preceded by at least one rubout. The DEC Binary format does not have addresses.

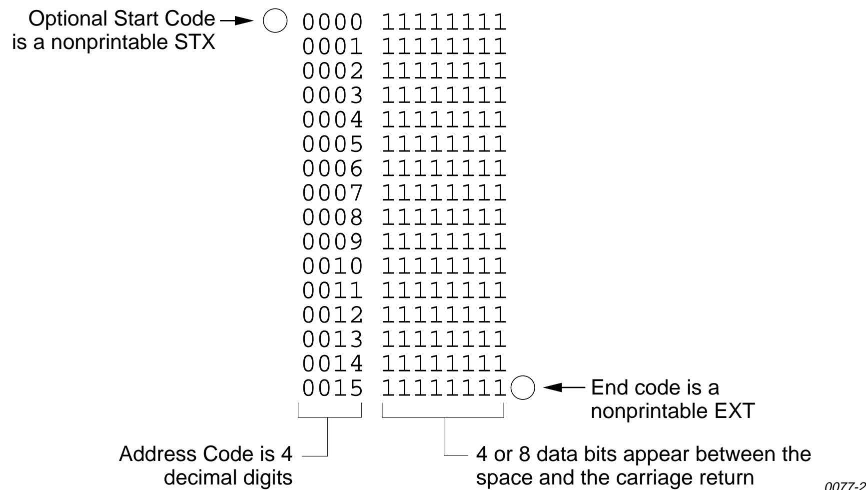
Formats 11 does not function properly unless you select NO parity and 8-bit data.

Spectrum Format, Codes 12 or 13

In this format, bytes are recorded in ASCII codes with binary digits represented by 1s and 0s. During output, each byte is preceded by a decimal address.

Figure 5-5 shows sample data bytes coded in the Spectrum format. Bytes are sandwiched between the space and carriage return characters and are normally separated by line feeds. The start code is a nonprintable STX, CTRL-B (or hex 02), and the end code is a nonprintable ETX, CTRL-C (or hex 03).

Figure 5-5. Spectrum Format (example)



Note: Data without a start or end code may be input to or output from the programmer by use of the alternate data translation format code, 13.

POF (Programmer Object File) Format, Code 14

The POF (Programmer Object File) format provides a highly compact data format to enable translation of high bit count logic devices efficiently. This format currently applies to MAX™ devices, such as the Altera 5032.

The information contained in the file is grouped into "packets." Each packet contains a "tag," identifying what sort of data the package contains plus the data itself. This system of packeting information allows for future definitions as required.

The POF is composed of a header and a list of packets. The packets have variable lengths and structures, but the first six bytes of every packet always adhere to the following structure.

```
struct PACKET_HEAD
{
short tag;/*tag number - type of packet */
long length;/*number of bytes in rest of packet */
}
```

A POF is read by the program examining each packet, and if the tag value is recognized, then the packet is used. If a tag value is not recognized, the packet is ignored.

Any packet except the terminator packet may appear multiple times within a POF. Packets do not need to occur in numerical tag sequence. The POF reader software is responsible for the interpretation and action taken as a result of any redundant data in the file including the detection of error conditions.

The POF format currently uses the following packet types.

Note: In the following packet type descriptions, one of the terms — Used, Skipped, or Read — will appear after the tag and name.

Used: The information in this packet is used by the programmer.
Skipped: This information is not used by the programmer.
Read: This information is read by the programmer but has no direct application.

Creator_ID tag = 1 Used

This packet contains a version ID string from the program which created the POF.

Device_Name tag = 2 Used

This packet contains the ASCII name of the target device to be programmed, for example, PM9129.

Comment_Text tag = 3 Read

This packet contains a text string which may consist of comments related to the POF. This text may be displayed to the operator when the file is read. The string may include multiple lines of text, separated by appropriate new line characters.

Tag_Reserved tag = 4 Skipped

Security_Bit	tag = 5	Used
	This packet declares whether security mode should be enabled on the target device.	
Logical_Address_ and_Data_16	tag = 6	Read
	This packet defines a group of logical addresses in the target device and associates logical data with these addresses. The addresses comprise a linear region in the logical address space, bounded on the low end by the starting address and extending upward by the address count specified in the packet. The starting address and address count are each specified by two-byte fields (16 bits).	
Electrical_Address _and_Data	tag = 7	Used
	This packet defines a group of electrical addresses in the target device and associates data values with those addresses. The data field is ordered in column-row order, beginning with the data for the least column-row address, continuing with increasing row addresses until the first column is filled, then incrementing the column address, etc.	
Terminator	tag = 8	Used
	This packet signals the end of the packet list in the POF. This packet must be the Nth packet, where N is the packet count declared in the POF header. The CRC field is a 16-bit Cyclic Redundancy Check computed on all bytes in the file up to, but not including, the CRC value itself. If this CRC value is zero, the CRC check should be ignored.	
Symbol table	tag = 9	Skipped
Test Vectors	tag = 10	Used
	This packet allows the POF to contain test vectors for post programming testing purposes. Each vector is a character string and uses the 20 character codes for vector bits defined in JEDEC standard 3A, section 7.0.	
Electrical_Address _and_Constant_ data	tag = 12	Skipped
Number of programmable elements	tag = 14	Read
	This packet defines the number of programmable elements in the target device.	
Logical_Address_ and_Data_32	tag = 17	Read

This packet defines a group of logical addresses in the target device and associates logical data with these addresses. The addresses comprise a linear region in the logical address space, bounded on the low end by the starting address and extending upward by the address count specified in the packet.

The starting address and address count are each specified by 4-byte fields (32 bits).

Absolute Binary Format, Code 16

Absolute Binary format is a literal representation of the data to be transferred, and no translation of the data takes place during the transfer. There are no overhead characters added to the data (i.e. no address record, start code, end code, nulls, or checksum). Every byte transferred represents the users data. This format can be used to download unformatted data such as a ".exe" file to the programmer.

Since this format does not have an end of file character, download transfers will terminate after no more data is received and an I/O timeout occurs. This is true for all data formats which don't have an end of file indicator. For this reason, do not use a value of 0 for the I/O timeout parameter on the communication parameters screen, since this will disable the timeout from occurring. A value between 1 and 99 (inclusive) should be used for the I/O timeout parameter when using formats which require the timeout to occur.

LOF Format, Code 17

The Link Object Format (LOF) is an extension of the standard JEDEC data translation format and is used to transfer fuse and test vector data between the programmer and a host computer. LOF is designed to support the Quicklogic QL8x12A family of FPGAs. An LOF data file is stored as an imploded ZIP file, which yields data compression approaching 95%.

Note: The specification for the ZIP data compression algorithm allows for multiple data files to be compressed into one ZIP file. In addition, the ZIP data compression algorithm allows for multiple types of data compression.

The programmer's implementation of UNZIP supports only imploded data files and will extract only the first file in a ZIP file. All remaining files in the ZIP file will be ignored, as will all files not stored in the imploded format.

The LOF format contains both a subset and a superset of the JEDEC format described in this chapter. This section describes only the fields that are extensions of the JEDEC standard or that are unique to the LOF format. See the section explaining the JEDEC format for information on the standard JEDEC fields. See page 5-27 for information on obtaining a copy of the JEDEC Standard 3A.

LOF Field Syntax

The LOF character set consists of all the characters that are permitted with the JEDEC format: all printable ASCII characters and four control characters. The four allowable control characters are STX, ETX, CR (Return), and LF (line feed). Other control characters, such as Esc or Break, should not be used.

Note: This is Data I/O Corporation's implementation of Quicklogic's Link Object Format. Contact Quicklogic for a more in-depth explanation of the format and its syntax.

LOF Fields

The following fields are included in Data I/O's implementation of the LOF format:

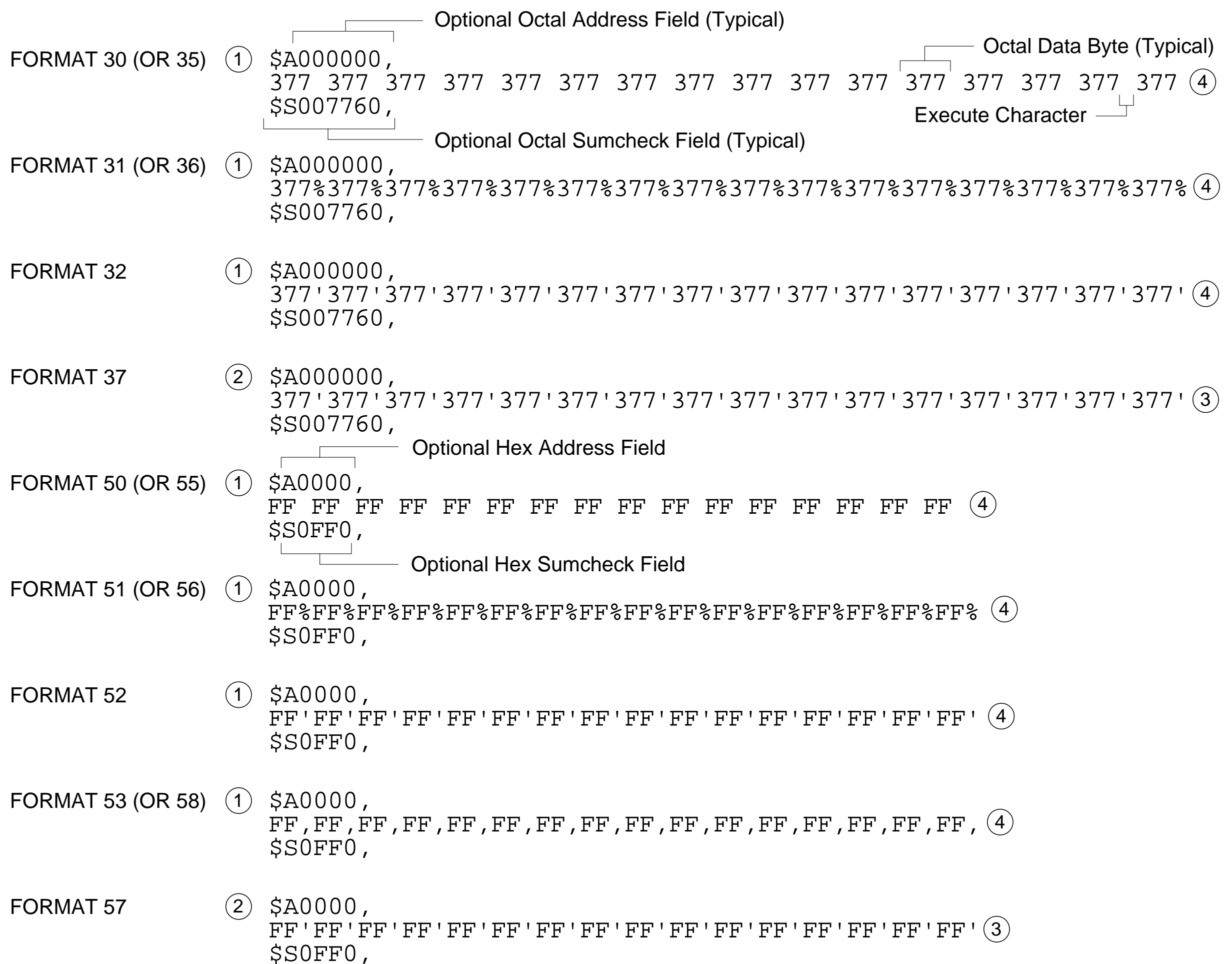
Field		Description
<STX>	*	Start of Data (ASCII Ctrl-B, 0x02 hex)
C	*	Fuse Checksum
K		Fuse data, followed by control words and pulse link cycles
N	*	Notes Field
QB		Number of bits per word
QC		Number of control words at the end of each K field
QF		Number of Fuses in Device (# of K fields)
QM		Number of macro cells in the data file
QP	*	Number of Device Package Pins
QS		Number of Hex-ASCII words in each K field and each control word
QV	*	Maximum Number of Test Vectors
R		Signature Analysis (reserved for future use)
S		SpDE Checksum
T		Signature Analysis (reserved for future use)
V	*	Test Vectors (reserved for future use)
X	*	Default Test Conditions (reserved for future use)
<ETX>	*	End of Data (ASCII Ctrl-C, 0x03 hex)

* *These fields are already defined as part of the JEDEC standard and will not be defined in this section.*

ASCII Octal and Hex Formats, Codes 30-37 and 50-58

Each of these formats has a start and end code, and similar address and checksum specifications. Figure 5-6 illustrates 4 data bytes coded in each of the 9 ASCII Octal and Hexadecimal formats. Data in these formats is organized into sequential bytes separated by the execute character (space, percent, apostrophe, or comma). Characters immediately preceding the execute character are interpreted as data. ASCII Octal and Hex formats can express 8-bit data, by 3 octal, or 2 hexadecimal characters. Line feeds, carriage returns, and other characters may be included in the data stream as long as a data byte directly precedes each execute character.

Figure 5-6. ASCII Octal and Hex Formats (example)



Although each data byte has an address, most are implied. Data bytes are addressed sequentially unless an explicit address is included in the data stream. This address is preceded by a $ and an A, must contain 2 to 8 hex or 3 to 11 octal characters, and must be followed by a comma, except for the ASCII-Hex (Comma) format, which uses a period. The programmer skips to the new address to store the next data byte; succeeding bytes are again stored sequentially.

Each format has an end code, which terminates input operations. However, if a new start code follows within 16 characters of an end code, input will continue uninterrupted. If no characters come within 2 seconds, input operation is terminated.

After receiving the final end code following an input operation, the programmer calculates a sumcheck of all incoming data. Optionally, a sumcheck can also be entered in the input data stream. The programmer compares this sumcheck with its own calculated sumcheck. If they match, the programmer will display the sumcheck; if not, a sumcheck error will be displayed.

> *Note: The sumcheck field consists of either 2-4 hex or 3-6 octal characters, sandwiched between the $ and comma characters. The sumcheck immediately follows an end code. The sumcheck is optional in the input mode but is always included in the output mode. The most significant digit of the sumcheck may be 0 or 1 when expressing 16 bits as 6 octal characters.*

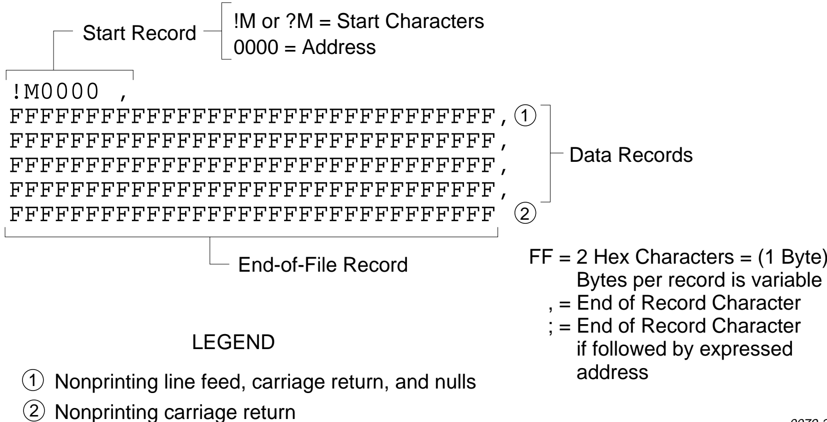
The programmer divides the output data into 8-line blocks. Data transmission is begun with the start code, a nonprintable STX character, or optionally, SOH.* Data blocks follow, each one prefaced by an address for the first data byte in the block. The end of transmission is signaled by the end code, a nonprintable ETX character. Directly following the end code is a sumcheck of the transferred data.

* *ASCII-Octal SMS and ASCII-Hex SMS use SOM (CTRL-R) as a start code and EOM (CTRL-T) as an end code.*

RCA Cosmac Format, Code 70

Data in this format begins with a start record consisting of the start character (!M or ?M), an address field, and a space. See Figure 5-7.

Figure 5-7. RCA Cosmac Format (example)



The start character ?M is sent to the programmer by a development system, followed by the starting address, and a data stream which conforms to the data input format described in the ASCII-Hex and Octal figure. Transmission stops when the specified number of bytes has been transmitted.

Address specification is required for only the first data byte in the transfer. An address must have 1 to 4 hex characters and must be followed by a space. The programmer records the next hexadecimal character after the space as the start of the first data byte. (A carriage return must follow the space if the start code ?M is used.) Succeeding bytes are recorded sequentially.

Each data record is followed by a comma if the next record is not preceded by an address, or by a semicolon if it starts with an address. Records consist of data bytes expressed as 2 hexadecimal characters and followed by either a comma or semicolon, and a carriage return. Any characters received between a comma or semicolon and a carriage return will be ignored by the programmer.

The carriage return character is significant to this format because it can signal either the continuation or the end of data flow; if the carriage return is preceded by a comma or semicolon, more data must follow; the absence of a comma or semicolon before the carriage return indicates the end of transmission.

Output data records are followed by either a comma or a semicolon and a carriage return. The start-of-file records are expressed exactly as for input.

Fairchild Fairbug, Code 80

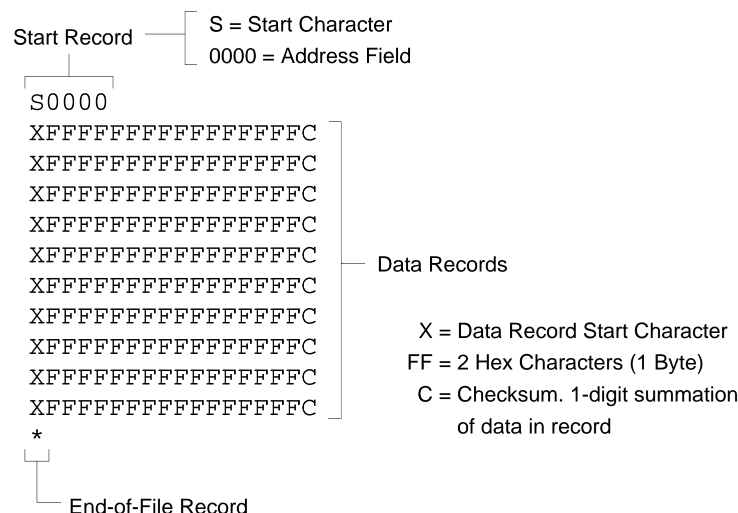
In the Fairbug format, input and output requirements are identical; both have 8-byte records and identical control characters. Figure 5-8 shows a Fairbug data file. A file begins with a 5-character prefix and ends with a 1-character suffix. The start-of-file character is an S, followed by the address of the first data byte. Each data byte is represented by 2 hexadecimal characters. The programmer will ignore all characters received prior to the first S.

Note: Address specification is optional in this format; a record with no address directly follows the previous record.

Each data record begins with an X and always contains 8 data bytes. A 1-digit hexadecimal checksum follows the data in each data record. The checksum represents, in hexadecimal notation, the sum of the binary equivalents of the 16 digits in the record; the half carry from the fourth bit is ignored.

The programmer ignores any character (except for address characters and the asterisk character, which terminates the data transfer) between a checksum and the start character of the next data record. This space can be used for comments.

Figure 5-8. Fairchild Fairbug (example)



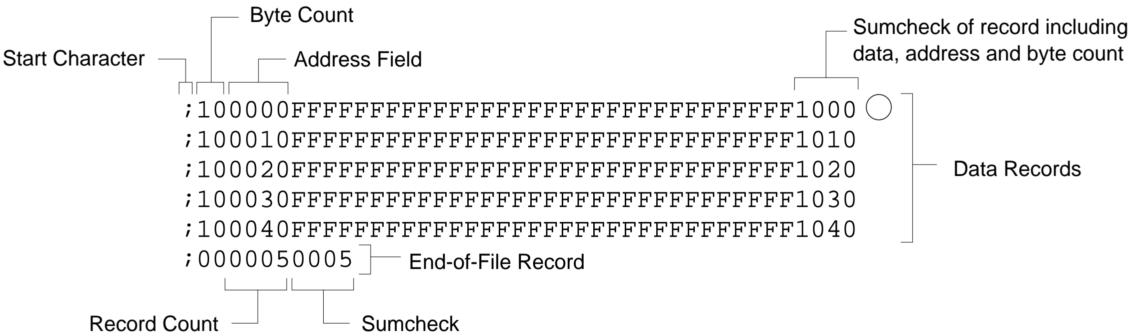
The last record consists of an asterisk only, which indicates the end of file.

MOS Technology Format, Code 81

The data in each record is sandwiched between a 7-character prefix and a 4-character suffix. The number of data bytes in each record must be indicated by the byte count in the prefix. The input file can be divided into records of various lengths.

Figure 5-9 shows a series of valid data records. Each data record begins with a semicolon. The programmer will ignore all characters received prior to the first semicolon. All other characters in a valid record must be valid hexadecimal digits (0-9 and A-F). A 2-digit byte count follows the start character. The byte count, expressed in hexadecimal digits, must equal the number of data bytes in the record. The byte count is greater than zero in the data records, and equals zero (00) in the end-of-file record. The next 4 digits make up the address of the first data byte in the record. Data bytes follow, each represented by 2 hexadecimal digits. The end-of-file record consists of the semicolon start character, followed by a 00 byte count, the record count, and a checksum.

Figure 5-9. MOS Technology Format (example)

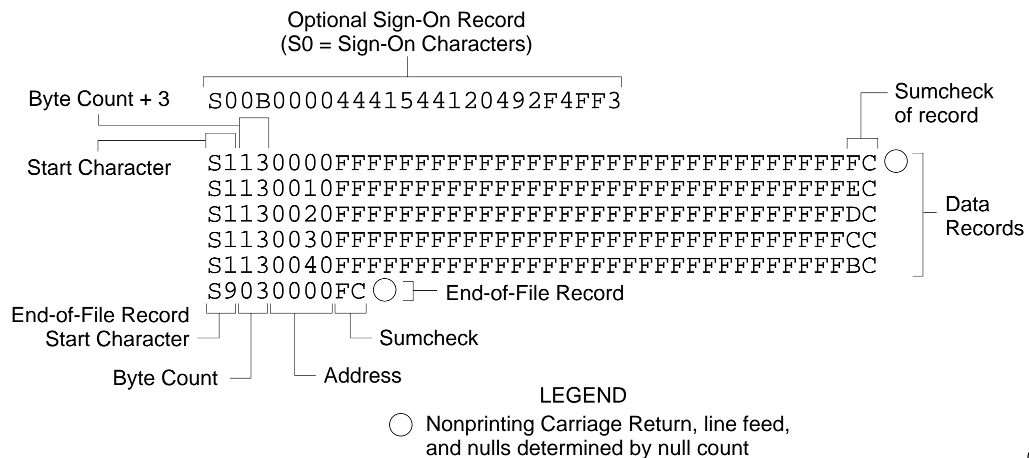


The checksum, which follows each data record, is a 2-byte binary summation of the preceding bytes in the record (including the address and byte count), in hexadecimal notation.

Motorola EXORciser Format, Code 82

Motorola EXORciser data files may begin with an optional sign-on record, which is initiated by the start characters S0. Valid data records start with an 8-character prefix and end with a 2-character suffix. Figure 5-10 shows a series of valid Motorola data records.

Figure 5-10. Motorola EXORciser Format (example)



Each data record begins with the start characters S1. The third and fourth characters represent the byte count, which expresses the number of data, address, and checksum bytes in the record. The address of the first data byte in the record is expressed by the last 4 characters of the prefix. Data bytes follow, each represented by 2 hexadecimal characters. The number of data bytes occurring must be three less than the byte count. The suffix is a 2-character checksum, which equals the one's complement of the binary summation of the byte count, address, and data bytes.

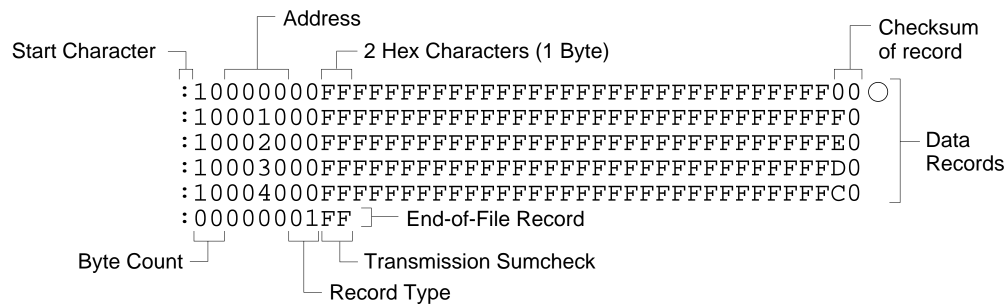
The end-of-file record consists of the start characters S9, the byte count, the address (in hex), and a checksum. The maximum record length is 250 data bytes.

Intel Intellec 8/MDS Format, Code 83

Intel data records begin with a 9-character prefix and end with a 2-character suffix. The byte count must equal the number of data bytes in the record.

Figure 5-11 simulates a series of valid data records. Each record begins with a colon, which is followed by a 2-character byte count. The 4 digits following the byte count give the address of the first data byte. Each data byte is represented by 2 hexadecimal digits; the number of data bytes in each record must equal the byte count. Following the data bytes of each record is the checksum, the two's complement (in binary) of the preceding bytes (including the byte count, address, record type and data bytes), expressed in hex.

Figure 5-11. Intel Intellec 8/MDS Format (example)

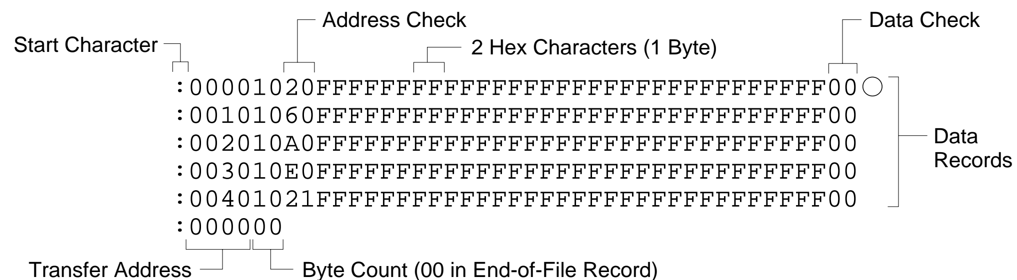


The end-of-file record consists of the colon start character, the byte count (equal to 00), the address, the record type (equal to 01), and the checksum of the record.

Signetics Absolute Object Format, Code 85

Figure 5-12 shows the specifications of Signetics format files. The data in each record are sandwiched between a 9-character prefix and a 2-character suffix.

Figure 5-12. An Example of Signetics Absolute Object Format



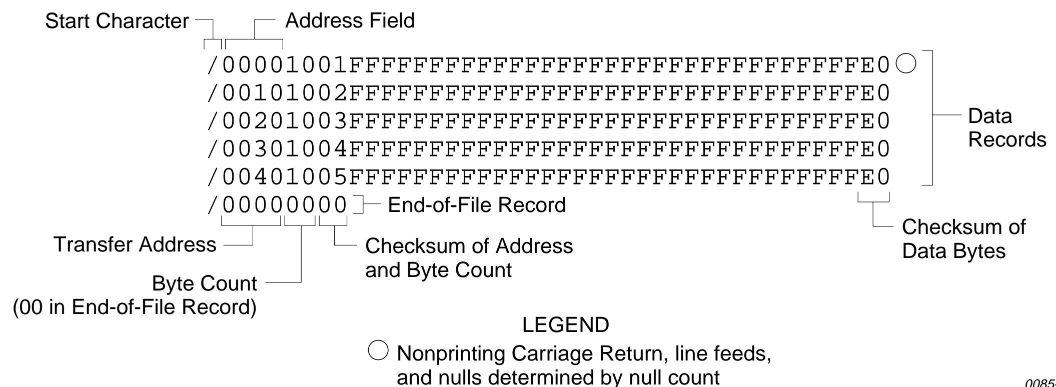
The start character is a colon. This is followed by the address, the byte count, and a 2-digit address check. The address check is calculated by exclusive ORing every byte with the previous one, then rotating left one bit. Data are represented by pairs of hexadecimal characters. The byte count must equal the number of data bytes in the record. The suffix is a 2-character data check, calculated using the same operations described for the address check.

The end-of-file record consists of the colon start character, the address, and the byte count (equal to 00).

Tektronix Hexadecimal Format, Code 86

Figure 5-13 illustrates a valid Tektronix data file. The data in each record are sandwiched between the start character (a slash) and a 2-character checksum. Following the start character, the next 4 characters of the prefix express the address of the first data byte. The address is followed by a byte count, which represents the number of data bytes in the record, and by a checksum of the address and byte count. Data bytes follow, represented by pairs of hexadecimal characters. Succeeding the data bytes is their checksum, an 8-bit sum, modulo 256, of the 4-bit hexadecimal values of the digits making up the data bytes. All records are followed by a carriage return.

Figure 5-13. Tektronix Hex Format (example)



Data are output from the programmer starting at the first RAM address and continuing until the number of bytes in the specified block has been transmitted. The programmer divides output data into records prefaced by a start character and an address field for the first byte in the record.

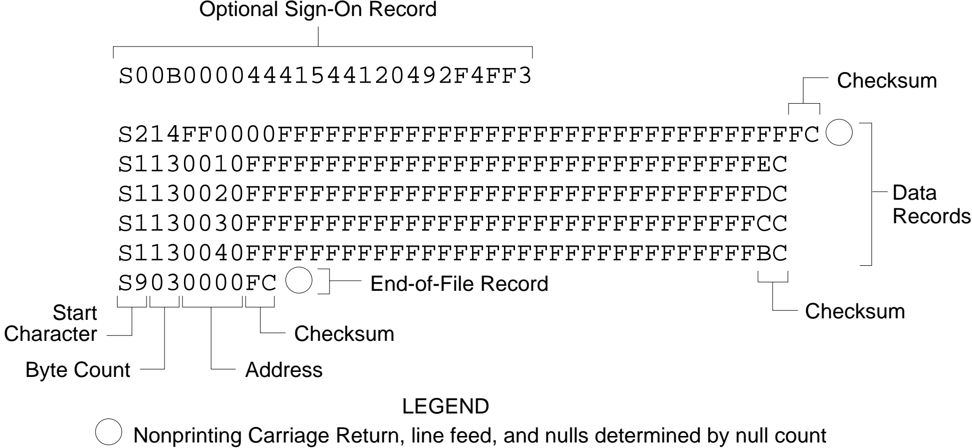
The end-of-file record consists of a start character (slash), followed by the transfer address, the byte count (equal to 00), and the checksum of the transfer address and byte count.

An optional abort record contains 2 start characters (slashes), followed by an arbitrary string of ASCII characters. Any characters between a carriage return and a / are ignored.

Motorola EXORmacs Format, Code 87

Motorola data files may begin with an optional sign-on record, initiated by the start characters S0. Data records start with an 8- or 10-character prefix and end with a 2-character suffix. Figure 5-14 shows a series of Motorola EXORmacs data records.

Figure 5-14. Motorola EXORmacs Format (example)



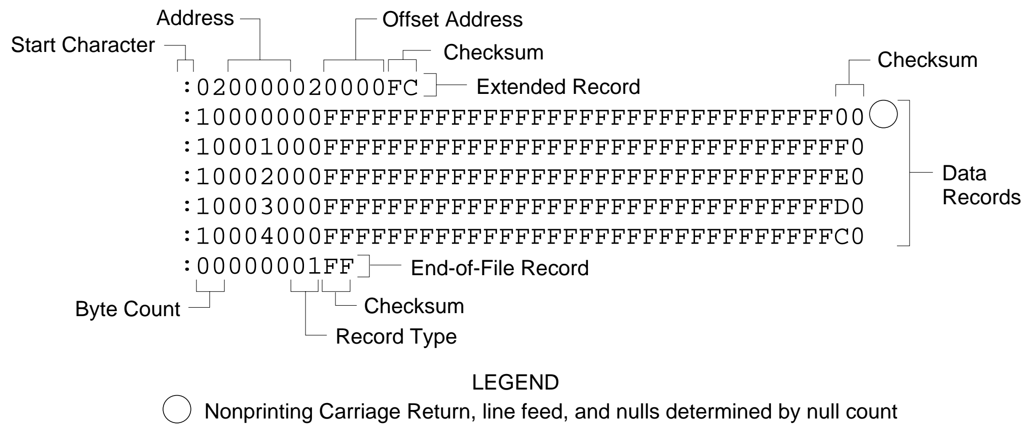
Each data record begins with the start characters S1 or S2: S1 if the following address field has 4 characters, S2 if it has 6 characters. The third and fourth characters represent the byte count, which expresses the number of data, address, and checksum bytes in the record. The address of the first data byte in the record is expressed by the last 4 characters of the prefix (6 characters for addresses above hexadecimal FFFF). Data bytes follow, each represented by 2 hexadecimal characters. The number of data bytes occurring must be 3 or 4 less than the byte count. The suffix is a 2-character checksum, the one's complement (in binary) of the preceding bytes in the record, including the byte count, address, and data bytes.

The end-of-file record begins with an S9 start character. Following the start characters are the byte count, the address, and a checksum. The maximum record length is 250 data bytes.

Intel MCS-86 Hexadecimal Object, Code 88

The Intel 16-bit Hexadecimal Object file record format has a 9-character (4-field) prefix that defines the start of record, byte count, load address, and record type and a 2-character checksum suffix. Figure 5-15 shows a sample record of this format.

Figure 5-15. Intel MCS-86 Hex Object (example)



The four record types are described below.

00 — Data Record

This begins with the colon start character, which is followed by the byte count (in hex notation), the address of the first data byte, and the record type (equal to 00). Following these are the data bytes. The checksum follows the data bytes and is the two's complement (in binary) of the preceding bytes in the record, including the byte count, address, record type, and data bytes.

01 — End Record

This end-of-file record also begins with the colon start character. This is followed by the byte count (equal to 00), the address (equal to 0000), the record type (equal to 01), and the checksum, FF.

02 — Extended Segment Address Record

This is added to the offset to determine the absolute destination address. The address field for this record must contain ASCII zeros (Hex 30s). This record type defines bits 4 to 19 of the segment base address. It can appear randomly anywhere within the object file and affects the absolute memory address of subsequent data records in the file. The following example illustrates how the extended segment address is used to determine a byte address.

Problem:

Find the address for the first data byte for the following file.

: 02 0000 02 1230 BA
: 10 0045 00 55AA FF .....BC

Solution:

Step 1. Find the record address for the byte. The first data byte is 55. Its record address is 0045 from above.

Step 2. Find the offset address. The offset address is 1230 from above.

Step 3. Shift the offset address one place left, then add it to the record address, like this:

```
  1230      Offset address (upper 16 bits)
+  0045     Record address (lower 16 bits)
 ------
  12345     20-bit address
```

The address for the first data byte is 12345.

Note: Always specify the address offset when using this format, even when the offset is zero.

During output translation, the firmware will force the record size to 16 (decimal) if the record size is specified greater than 16. There is no such limitation for record sizes specified less than 16.

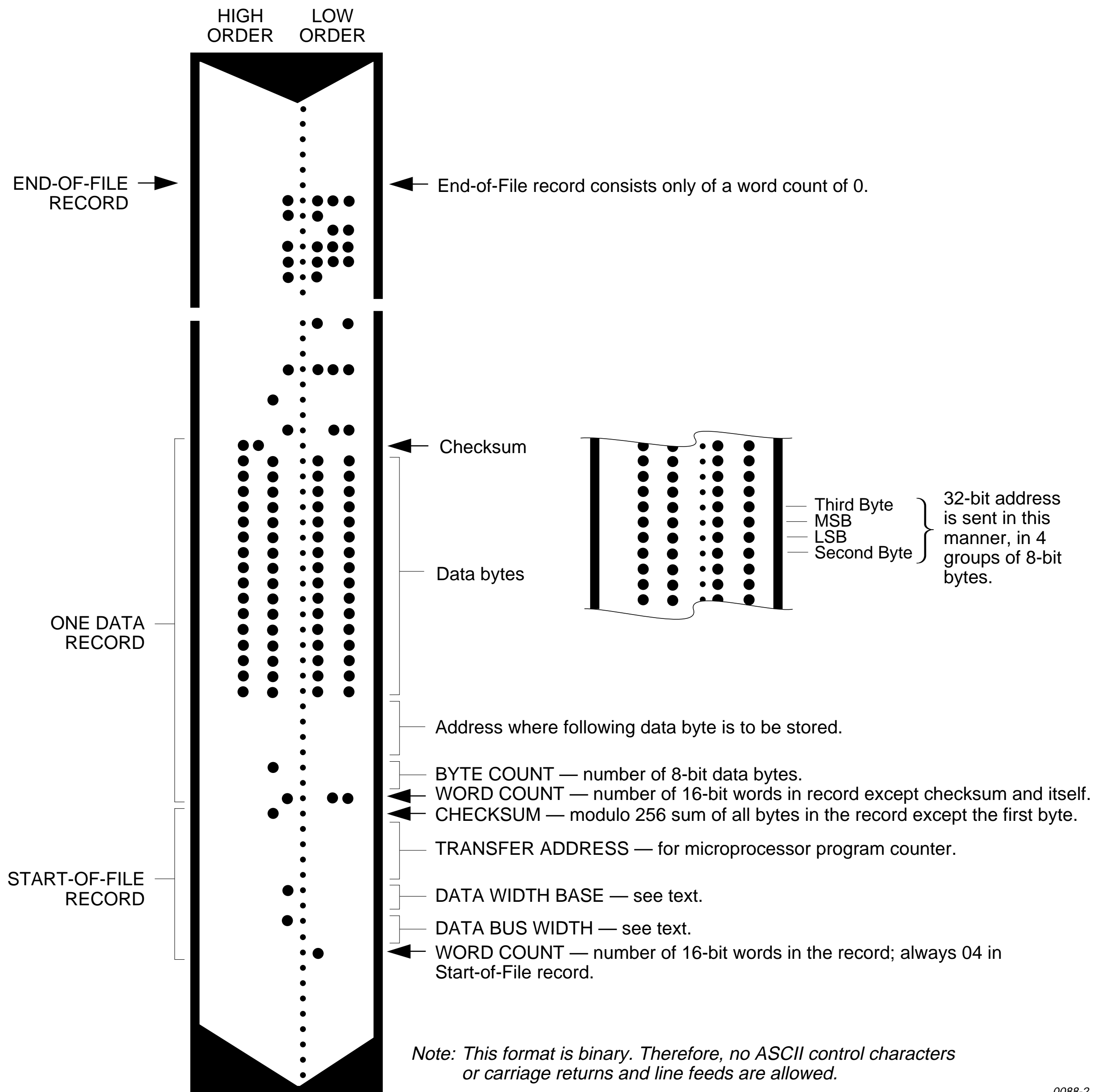
03 — Start Record

This record type is not sent during output by Data I/O translator firmware.

Hewlett-Packard 64000 Absolute Format, Code 89

Hewlett-Packard Absolute is a binary format with control and data-checking characters. See Figure 5-16.

Figure 5-16. HP 64000 Absolute Format (example)



Data files begin with a Start-of-file record, which includes the Data Bus Width, Data Width Base, Transfer Address, and a checksum of the bytes in the record.

The Data Bus Width represents the width of the target system's bus (in bits). The Data Width Base represents the smallest addressable entity used by the target microprocessor.

The Data Bus Width and Data Width Base are not used by the 2900/3900 during download. During upload, the Data Bus Width will be set to the current Data Word Width, and the Data Width Base will be set to 8. The Transfer Address is not used by the 2900/3900.

Data records follow the Start-of-file record. Each begins with 2 byte counts: the first expresses the number of 16-bit bytes in the record not including the checksum and itself; the second expresses the number of 8-bit data bytes in the record. Next comes a 32-bit address, which specifies the storage location of the following data byte. Data bytes follow; after the last data byte is a checksum of every byte in the record except the first byte, which is the word count.

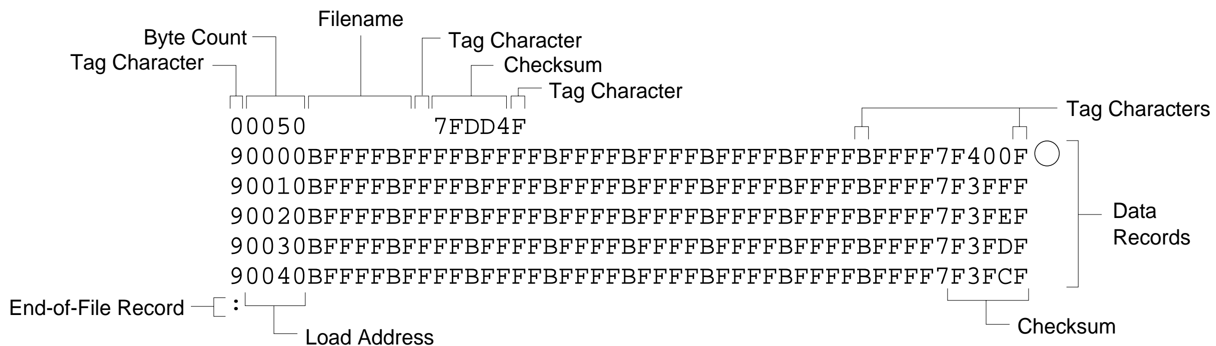
The End-of-file record consists of a one byte word count, which is always zero. Leader and trailer nulls, normally 50 each, are suppressed in this translation format.

Format 89 does not function properly unless you select NO parity and 8-bit data.

Texas Instruments SDSMAC Format, Code 90

Data files in the SDSMAC format consist of a start-of-file record, data records, and an end-of-file record. See Figure 5-17.

Figure 5-17. TI SDSMAC Format (example)



Each record is composed of a series of small fields, each initiated by a tag character. The programmer recognizes and acknowledges the following tag characters:

0 or K — followed by a file header.
7 — followed by a checksum which the programmer acknowledges.
8 — followed by a checksum which the programmer ignores.
9 — followed by a load address.
B — followed by 4 data characters.
F — denotes the end of a data record.
* — followed by 2 data characters.

The start-of-file record begins with a tag character and a 12-character file header. The first four characters are the byte count of the data bytes; the remaining file header characters are the name of the file and may be any ASCII characters (in hex notation). Next come interspersed address fields and data fields (each with tag characters). If any data fields appear before the first address field in the file, the first of those data fields is assigned to address 0000. Address fields may be expressed for any data byte, but none are required.

The record ends with a checksum field initiated by the tag character 7 or 8, a 4-character checksum, and the tag character F. The checksum is the two's complement of the sum of the 8-bit ASCII values of the characters, beginning with the first tag character and ending with the checksum tag character (7 or 8).

Data records follow the same format as the start-of-file record but do not contain a file header. The end-of-file record consists of a colon (:) only. The output translator sends a CTRL-S after the colon.

JEDEC Format, Codes 91 and 92

The JEDEC (Joint Electron Device Engineering Council) format is used to transfer fuse and test vector data between the programmer and a host computer. Code 91 is full format, and includes all the data fields (such as note and test fields) described on the following pages. Code 92 is the Kernel, or shorter, format. The JEDEC Kernel format includes only the minimum information needed for the programming; it does not, for example, include information fields or test vector fields. Prior to transferring a JEDEC file, the appropriate Logic device must be selected.

JEDEC's legal character set consists of all the printable ASCII characters and four control characters. The four allowable control characters are STX, ETX, CR (RETURN), and LF (line feed). Other control characters, such as ESC or BREAK, should not be used.

Note: This is Data I/O Corporation's implementation of JEDEC Standard 3A. For a copy of the strict standard, write to:

Electronic Industries Association
Engineering Department
2001 Eye Street NW
Washington, D.C. 20006

BNF Rules and Standard Definitions

The Backus-Naur Form (BNF) is used in the description here to define the syntax of the JEDEC format. BNF is a shorthand notation that follows these rules:

:: = denotes “is defined as.”

Characters enclosed by single quotes are literals (required).

Angle brackets enclose identifiers.

Square brackets enclose optional items.

Braces {} enclose a repeated item. The item may appear zero or more times.

Vertical bars indicate a choice between items.

Repeat counts are given by a :n suffix. For example, a 6-digit number would be defined as:

<number< :: = <digit>:6

For example, in words, the definition of a person's name reads:

The full name consists of an optional title followed by a first name, a middle name, and a last name. The person may not have a middle name or may have several middle names. The titles consist of: Mr., Mrs., Ms., Miss, and Dr.

The BNF definition for a person's name is:

<full name> :: = [<title>] <f. name> {<m.name>} <l. name>

<title> :: = 'Mr.' | 'Mrs.' | 'Ms.' | 'Miss' | 'Dr.'

The following standard definitions are used throughout the rest of this document:

<digit> :: = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<hex-digit> :: = <digit> | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'

<binary-digit> :: = '0' | '1'

<number> :: = <digit> {<digit>}

<del> :: = <space> | <carriage return>

<delimiter> :: = <del> {<del>}

<printable character> :: = <ASCII 20 hex ... 7E hex>

<control character> :: = <ASCII 00 hex ... 1F hex> | <ASCII 7F hex>

<STX> :: = <ASCII 02 hex>

<ETX> :: = <ASCII 03 hex>

<carriage return> :: = <ASCII 0D hex>

<line feed> :: = <ASCII 0A hex>

<space> :: = <ASCII 20 hex> | ' '

<valid character> :: = <printable character> | <carriage return> | <line feed>

```
<field character> :: = <ASCII 20 hex ... 29 hex> | <ASCII 2B hex ... 7E hex> | <carriage return> | <line feed>
```

The Design Specification Field

<design spec> ::= {<field character>}`*'

The first field sent in a JEDEC transmission is the design specification. Both the full and kernel JEDEC formats accept the design specification field. This field is mandatory and does not have an identifier (such as an asterisk) signaling its beginning. The design specification field consists of general device information. It could, for example, consist of the following information: your name, your company's name, the date, the device name and manufacturer, design revision level, etc. This field is terminated by an asterisk character. Examine the sample transmission shown on the next page of this description — the first three lines of the file comprise the design specification field. The programmer ignores the contents of this field for downloads and places "Data I/O" in this field for upload operations.

Note: You do not need to send any information in this field if you do not wish to; a blank field, consisting of the terminating asterisk, is a valid design specification field.

The Transmission Checksum Field

<xmit checksum> ::= <hex digit>:4

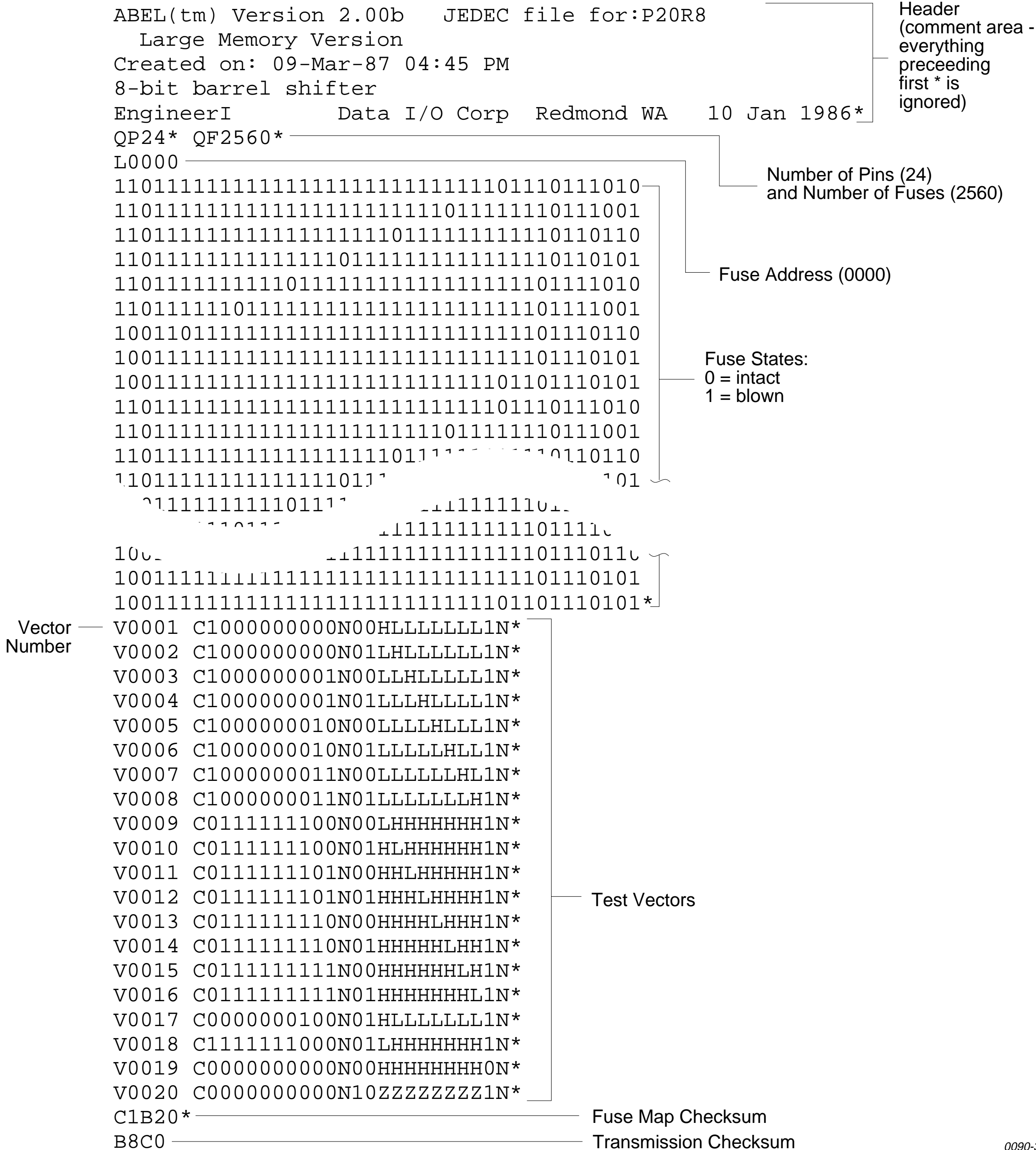
The transmission checksum is the last value sent in a JEDEC transmission. The full JEDEC format requires the transmission checksum. The checksum is a 16-bit value, sent as a 4-digit hex number, and is the sum of all the ASCII characters transmitted between (and including) the STX and ETX. The parity bit is excluded in the calculation of the transmission checksum.

Some computer systems do not allow you to control what characters are sent, especially at the end of a line. You should set up the equipment so that it will accept a dummy value of 0000 as a valid checksum. This zero checksum is a way of disabling the transmission checksum, while still keeping within the JEDEC format rules.

JEDEC Full Format, Code 91

The full JEDEC format consists of a start-of-text character (STX), various fields, an end-of-text character (ETX), and a transmission checksum. A sample JEDEC transmission sent in the full format is shown in Figure 5-18. Each of the fields is described on the following pages.

Figure 5-18. JEDEC Full Format (example)



JEDEC Field Syntax

<field> ::= [<delimiter>]<field identifier>{<field character>}`*'

<field identifier>::= 'A' | 'C' | 'D' | 'F' | 'G' | 'K' | 'L' | 'N' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'V' | 'X'

<reserved identifier>::= 'B' | 'E' | 'H' | 'I' | 'J' | 'M' | 'O' | 'U' | 'W' | 'Y' | 'Z'

Following the design specification field in a JEDEC transmission can be any number of information fields. Each of the JEDEC fields begins with a character that identifies what type of field it is. Fields are terminated by using an asterisk character. Multiple character identifiers can be used to create sub-fields (i.e., A1, A$, or AB3). Although they are not required, you may use carriage returns (CR) and line feeds (LF) to improve readability of the data.

Field Identifiers

Field identifiers which are currently used in JEDEC transmissions are shown above on the “field identifiers” line. The “reserved identifier” line indicates characters not currently used (reserved for future use as field identifiers). JEDEC field identifiers are defined as follows:

A	Access time	N	Note field
B	*	O	*
C	Checksum field	P	Pin sequence
D	Device type	Q	Value field
E	*	R	Resulting vector field
F	Default fuse state field	S	Starting vector
G	Security fuse field	T	Test cycles
H	*	U	*
I	*	V	Test vector field
J	*	W	*
K	Fuse list field (hex format)	X	default test condition
L	Fuse list field	Y	*
M	*	Z	*

* *Reserved for future use*

Device Field (D)

Device selection by this field is not supported by the programmer. It has been replaced by the QF and QP fields and manual selection of devices.

Fuse Information Fields (L, K, F, C)

<fuse information> :: = [<default state>] <fuse list> {<fuse list>} [<fuse checksum>]

<fuse list> : = 'L' <number> <delimiter> {<binary-digit> [<delimiter>]} ' * '

<fuse list> :: = 'K' <number> <delimiter> {<hex-digit> [<delimiter>]} '*'

<default state> :: = 'F' <binary-digit> ' * '

<fuse checksum> :: = 'C' <hex-digit>:4 ' * '

Each fuse of a device is assigned a decimal number and has two possible states: zero, specifying a low-resistance link, or one, specifying a high resistance link. The state of each fuse in the device is given by three fields: the fuse list (L field or K field), the default state (F field), and the fuse checksum (C field).

Fuse states are explicitly defined by either the L field or the K field. The character L begins the L field and is followed by the decimal number of the first fuse for which this field defines a state. The first fuse number is followed by a list of binary values indicating the fuse states.

The information in the K field is the same as that of the L field except that the information is represented by hex characters instead of binary values. This allows more compact representation of the fusemap data. The character K begins the K field and is followed by the decimal number of the first fuse. The fuse data follow the fuse number and are represented by hex characters. Each bit of each hex character represents the state of one fuse, so each hex character represents four fuses. The most significant bit of the first hex character following the fuse number corresponds to the state of that fuse number. The next most significant bit corresponds to the state of the next fuse number, etc. The least significant bit of the first hex character corresponds to the state of the fuse at the location specified by the fuse number plus three.

The K field supports download operations only. The K field is not part of the JEDEC standard, but is supported by Data I/O for fast data transfer. The L and K fields can be any length desired, and any number of L or K fields can be specified. If the state of a fuse is specified more than once, the last state specified replaces all previous ones for that fuse. The F field defines the states of fuses that are not explicitly defined in the L or K fields. If no F field is specified, all fuse states must be defined by L or K fields.

The C field, the fuse information checksum field, is used to detect transmitting and receiving errors. The field contains a 16-bit sum (modulus 65535) computed by adding 8-bit words containing the fuse states for the entire device. The 8-bit words are formed as shown in the following figure. Unused bits in the final 8-bit word are set to zero before the checksum is calculated.

Word 00	msb							lsb
Fuse No.	7	6	5	4	3	2	1	0
Word 01	msb							lsb
Fuse No.	15	14	13	12	11	10	9	8
Word 62	msb							lsb
Fuse No.	503	-	-	-	499	498	497	496

Following is an example of full specification of the L, C, and F fields:

F0*L0 01010101* L0008 01010111* L1000 0101*C019E*

Following is an alternate way of defining the same fuse states using the K field:

F0*K0 55* K0008 57* K1000 5* C019E*

Another example, where F and C are not specified:

L0200   01101010101010101011

010111010110100010010010010*

The Security Fuse Field (G)

<security fuse>::=`G'<binary-digit>`*'

The JEDEC G field is used to enable the security fuse of some logic devices. To enable the fuse, send a 1 in the G field:

G1*

The Note Field (N)

<note>::=`N'<field characters>`*'

The note field is used in JEDEC transmission to insert notes or comments. The programmer will ignore this field; it will not be interpreted as data. An example of a note field would be:

N Test Preload*

The Value Fields

(QF, QP, and QV)

JEDEC value fields define values or limits for the data file, such as number of fuses. The QF subfield defines the number of fuses in the device. All of the value fields must occur before any device programming or testing fields appear in the data file. Files with ONLY testing fields do not require the QF field and fields with ONLY programming data do not require the QP and QV fields.

The QF subfield tells the programmer how much memory to reserve for fuse data, the number of fuses to set to the default condition, and the number of fuses to include in the fuse checksum. The QP subfield defines the number of pins or test conditions in the test vector, and the QV subfield defines the maximum number of test vectors.

The P Field

The P field remaps the device pinout and is used with the V (test vector) field. An asterisk terminates the field. The syntax of the field is as follows:

<pin<N>list>::='P'<pin number>:N'*'

<pin<N>number>::=<delimiter><number>

The following example shows a P field, V field, and the resulting application:

P 1 2 3 4 5 6 14 15 16 17 7 8 9 10 11 12 13 18 19 20 *

V0001 111000HLHHNNNNNNNNNN*

V0002 100000HHHLNNNNNNNNNN*

The result of applying the above P and V fields is that vector 1 will apply 111000 to pins 1 through 6, and HLHH to pins 14 through 17. Pins 7 through 13 and 18 through 20 will not be tested.

JEDEC U and E Fields

As of Version 2.5, the programmer supports the optional JEDEC U (user data) and E (electrical data) fields. The U and E fields are described below.

Note: Implementation of the JEDEC U and E fields is not part of the JEDEC-3C (JESD3-C) standard.

User Data (U Field)

The **U** field allows user data fuses that do not affect the logical or electrical functionality of the device to be specified in JEDEC files. For instance, the U field can be used to specify the User Data Signature fuse available in some types of PLD devices because this fuse contains information only (it has no logical or electrical functionality).

Note: To have the JEDEC U field processed correctly, you must select the device before downloading the JEDEC file.

The following guidelines apply to the U field:

- The U field must be included for devices with U fuses.
- Each U-field cell must be explicitly provided if the U field is present.
- The F (default fuse state) field does not affect U fuses.
- There can only be one U field in a JEDEC file.
- The U field fuses must be listed in the order they appear in the device.
- The U field must be listed after the L field and E fields (if used), and before the V (test vector) field (if used).
- The U field is specified using binary numbers, since the full number of U-field cells is otherwise unknown.
- The number of cells specified in the U field is not included in the QF (number of fuses) field.
- The U-field cells are not included in the C (fuse checksum) field.
- The U field reads left to right to be consistent with the L (fuse list) and E fields.

The syntax for the U field is as follows:

```
<User Data Fuse List>::'U'<binary-digit(s)>'*'
```

The character U begins the U field and is followed by one binary digit for each U fuse. Each binary digit indicates one of two possible states (zero, specifying a low-resistance link, or one, specifying a high-resistance link) for each fuse.

For example,

```
QF24*
L0000
101011000000000000000000*
E10100111*
C011A*
U10110110*
```

Electrical Data (E field)

The **E** field allows special feature fuses that do not affect the logic function of the device to be specified in JEDEC files.

The following guidelines apply to the E field:

- The E-field cell must be explicitly provided if the E field is present.
- The F (default fuse state) field does not affect E fuses.
- There can only be one E field in a JEDEC file.
- The E field fuses must be listed in the order they appear in the device.
- The E field must be listed before the C (checksum) field. If the U field is used, the E field must come before the U (user data) field.
- The E field is specified using binary numbers, since the full number of E-field cells is otherwise unknown.
- The number of cells specified in the E field is not included in the QF (number of fuses) field. The E-field cells are included in the C (fuse checksum) field. The E field reads left to right for the purpose of checksum calculation.

The syntax for the E field is as follows:

```
<Electrical Data Fuse List>::'E'<binary digit(s)>'*'
```

The character E begins the E field and is followed by one binary digit for each E fuse. Each binary digit indicates one of two possible states (zero, specifying a low-resistance link, or one, specifying a high-resistance link) for each fuse. For example,

```
QF24*
L0000
101011000000000000000000*
E10100111*
C011A*
U10110110*
```

Test Field (V field)

<function test> :: = [<pin list>] <test vector> {<test vector>}

<pin<N>number> :: = <delimiter> <number>

N :: = number of pins on device

<test vector> :: = 'V' <number> <delimiter> < test condition> :N ' * '

<test condition> :: = <digit> 'B' | 'C' | 'D' | 'F' | 'H' | 'K' | 'L' | 'N' | 'P' | 'U' | 'X' | 'Z'

<reserved condition> :: = 'A' | 'E' | 'G' | 'I' | 'J' | 'M' | 'O' | 'Q' | 'R' | 'S' | 'T' | 'V' | 'W' | 'Y' | 'Z'

Functional test information is specified by test vectors containing test conditions for each device pin. Each test vector contains *n* test conditions where *n* is the number of pins on the device. The following table lists the conditions that can be specified for device pins.

When using structured test vectors to check your logic design, do NOT use 101 or 010 transitions as tests for clock pins: use C, K, U, or D instead.

Test Conditions

0	Drive input low
1	Drive input high
2-9	Drive input to supervoltage #2-9
B	Buried register preload (not supported)
C	Drive input low, high, low
D	Drive input low, fast slew
F	Float input or output
H	Test output high
K	Drive input high, low, high
L	Verifies that the specified output pin is low
N	Power pins and outputs not tested
P	Preload registers
U	Drive input high, fast slew
X	Output not tested, input default level
Z	Test input or output for high impedance

Note: C, K, U, and D are clocking functions that allow for setup time.

The C, K, U, and D driving signals are presented after the other inputs are stable. The L, H, and Z tests are performed after all inputs have stabilized, including C, K, U, and D.

Test vectors are numbered by following the V character with a number. The vectors are applied in numerical order. If the same numbered vector is specified more than one time, the data in the last vector replaces any data contained in previous vectors with that number.

The following example uses the V field to specify functional test information for a device:

```
V0001 C01010101NHLLLHHLHLN *
V0002 C01011111NHLLHLLLHLN *
V0003 C10010111NZZZZZZZZZN *
V0004 C01010100NFLHHLFFLLN *
```

JEDEC Kernel Mode, Code 92

<kernel>::=<STX><design spec><min. fuse information><ETX><xmit checksum>

<design spec>::={<field character>}`*'

<min. fuse information>::=<fuse list>{<fuse list>}

You may use the JEDEC kernel format if you wish to send only the minimum data necessary to program the logic device, for example, if you do not want to send any test vectors. If you specify format code 92, the programmer will ignore everything except the design specification field and the fuse information field. The following fields will be ignored if format 92 is specified: C, F, G, Q, V, and X. Also, the security fuse will be set to zero and the transmission checksum will be ignored.

Figure 5-19 shows an example of a kernel JEDEC transmission.

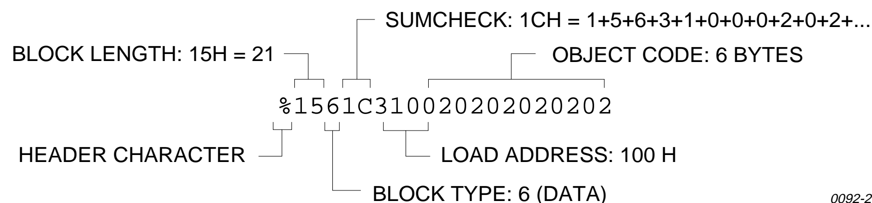
Figure 5-19. JEDEC Kernel Mode Format (example)

```
<STX>
Acme Logic Design  Jane Engineer   Feb. 29 1983
Widget Decode 756-AB-3456 Rev C Device Mullard 12AX7*

L0000 1111111011 1111111111  1111000000 0000000000
      0000000000 0000000000  0000000000 0000000000
      0000000000 0000000101  1111111111 1111111111
      0000000000 0000000000  0000111101 1111111111
      1111111111 1111110111  1111111111 1111111111*

L0200 1110101111 1111110000  0000000000 0000000000
      1111111111 1111011011  1111111111 1111111110
      0111111111 1111111111  1111111110 1111111111
      1111111111 1111101111  1111111111 1111101111
      0000000000 0000000000  0000*

<EXT>0000                                          0091-2
```

Extended Tektronix Hexadecimal Format, Code 94

The Extended Tektronix Hexadecimal format has three types of records: data, symbol, and termination records. The data record contains the object code. Information about a program section is contained in the symbol record (the programmer ignores symbol records), and the termination record signifies the end of a module. The data record (see sample below) contains a header field, a load address, and the object code. Figure 5-20 lists the information contained in the header field.

Figure 5-20. An Example of Tektronix Extended Format



Item	No. of ASCII Characters	Description
%	1	Signifies that the record is the Extended Tek Hex format.
Block length	2	Number of characters in the record, minus the %.
Block type	1	6 = data record 3 = symbol record (ignored by the programmer) 8 = termination record
Checksum	2	A 2-digit hex sum, modulo 256, of all the values in the record except the % and the checksum.

Character Values for Checksum Computation

The number of fields in the file will vary, depending on whether a data or a termination block is sent. Both data and termination blocks have a 6-character header and a 2-to-17 character address.

Character(s)	Value (decimal)	Character(s)	Value (decimal)
0 . . 9	0 . . 9	. (period)	38
A . . Z	10 . . 35	_(underline)	39
$	36	a . . z	40 . . 65
%	37		

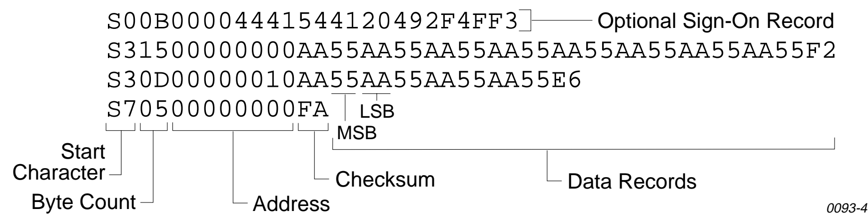
The load address determines where the object code will be located. This is a variable length number that may contain up to 17 characters. The first number determines the address length, with a zero signifying a length of 16. The remaining characters of the data record contain the object code, 2 characters per byte.

When you copy data to the port or to RAM, set the high-order address if the low-order is not at the default value.

Motorola 32-Bit Format, Code 95

The Motorola 32-bit format closely resembles the Motorola EXORmacs format, the main difference being the addition of the S3 and S7 start characters. The S3 character is used to begin a record containing a 4-byte address. The S7 character is a termination record for a block of S3 records. The address field for an S7 record may optionally contain the 4-byte instruction address that identifies where control is to be passed and is ignored by the programmer. Figure 5-21 shows a sample of the Motorola 32-bit format.

Figure 5-21. Motorola S3 Format (example)



Motorola data files may begin with an optional sign-on record, initiated by the start characters S0 or S5. Data records start with an 8- or 10-character prefix and end with a 2-character suffix.

Each data record begins with the start characters S1, S2, or S3: S1 if the following address field has 4 characters, S2 if it has 6 characters, S3 if it has 8 characters. The third and fourth characters represent the byte count, which expresses the number of data, address, and checksum bytes in the record. The address of the first data byte in the record is expressed by the last 4 characters of the prefix (6 characters for addresses above hexadecimal FFFF and 8 characters for addresses above hexadecimal FFFFFF). Data bytes follow, each represented by 2 hexadecimal characters. The number of data bytes occurring must be 3, 4, or 5 less than the byte count. The suffix is a 2-character checksum, the one's complement (in binary) of the preceding bytes in the record, including the byte count, address, and data bytes.

The end-of-file record begins with an S8 or S9 start character. Following the start characters are the byte count, the address, and a checksum. The maximum record length is 250 data bytes.

Hewlett-Packard UNIX Format, Code 96

This format divides the data file into data records; each with a maximum size of 250 bytes not including header information. An ID header is added to the beginning of the first record. Each subsequent record has its own header section. The section at the beginning of the file contains the following elements: the header 8004, filename, byte count for the processor information record, and the processor information record.

The header 8004 identifies the type of file being transferred. The first byte of this header (80) indicates that this file is binary and the 04 indicates the type of file (absolute).

The ID header is followed by a 16-byte filename (not used by the programmer).

Next is the byte count, which indicates the size (minus one) of the Processor Information Record that follows. The Processor Information Record is divided into the following data words: Data Bus Width, Data Width Base, Transfer Address LS (least significant), and Transfer Address MS (most significant).

The Data Bus Width represents the width of the target system's bus (in bits). The Data Width Base represents the smallest addressable entity used by the target microprocessor.

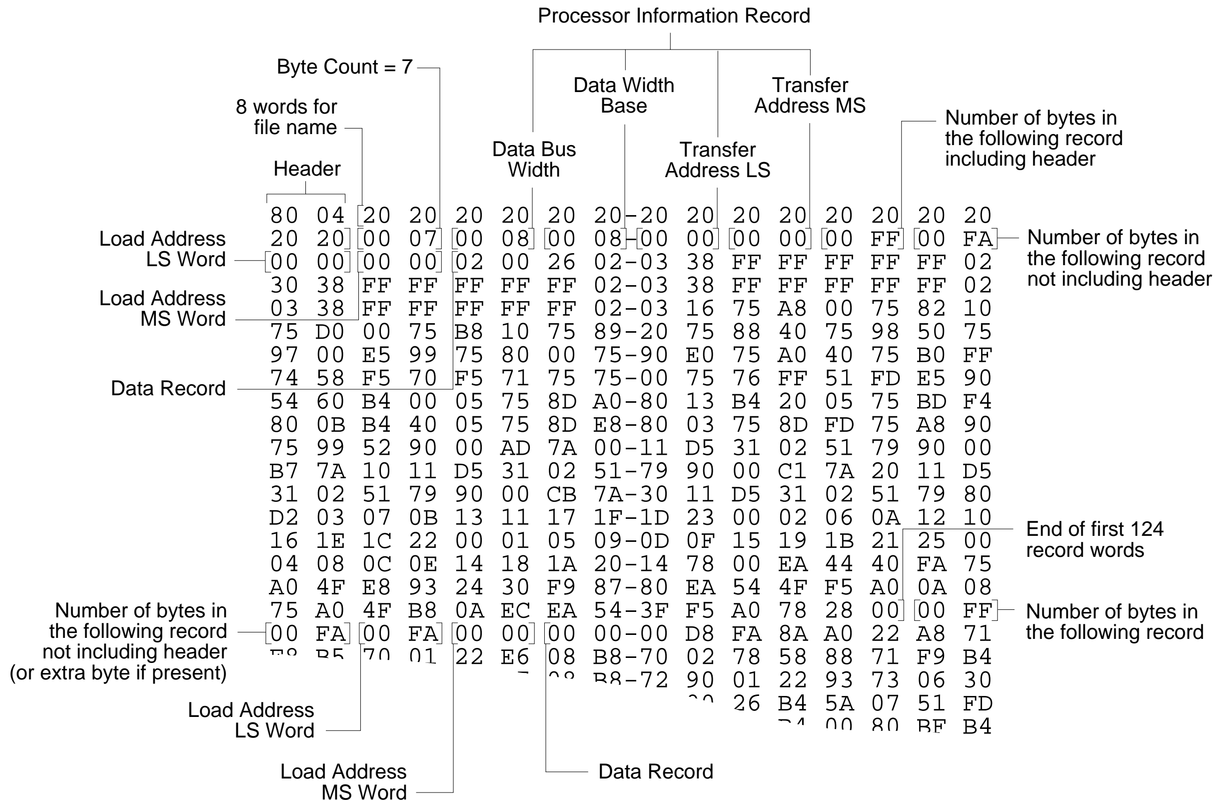
The Data Bus Width and Data Width Base are not used by the programmer during download. During upload, the Data Bus Width will be set to the current Data Word Width, and the Data Width Base will be set to 8. The Transfer Address LS and Transfer Address MS are not used by the programmer.

The data records consist of a header (8 bytes) and the data bytes. The first 2 bytes of the header indicate the size of the data record including the header (minus one). If the number of data bytes in the data record (not including the header) is odd, one extra byte will be added to the data record to ensure that an even number of data bytes exist in the data record. The maximum value for this field is 00FF hex. The next two bytes indicate the number of actual data bytes in the record, not including the header bytes and the extra byte (if present). The maximum value for this field is 00FA hex. The 4 bytes that follow represent the destination address for the data in this record. The rest of the bytes in the record are the data bytes.

This format has no end of file identifier.

The record length during upload is not affected by the upload record size parameter in the Configure/Edit/Communication screen. It is automatically set to transfer records using the maximum size (250 bytes), except for the last record. The size of the last record will be set according to the remaining number of data bytes.

Figure 5-22. Hewlett-Packard 64000 Unix Format



This data translation format was generated by a "dump utility" for illustrative purposes. Actual data files are in binary code and are typically generated by the appropriate development software.

Intel OMF386 Format, Code 97

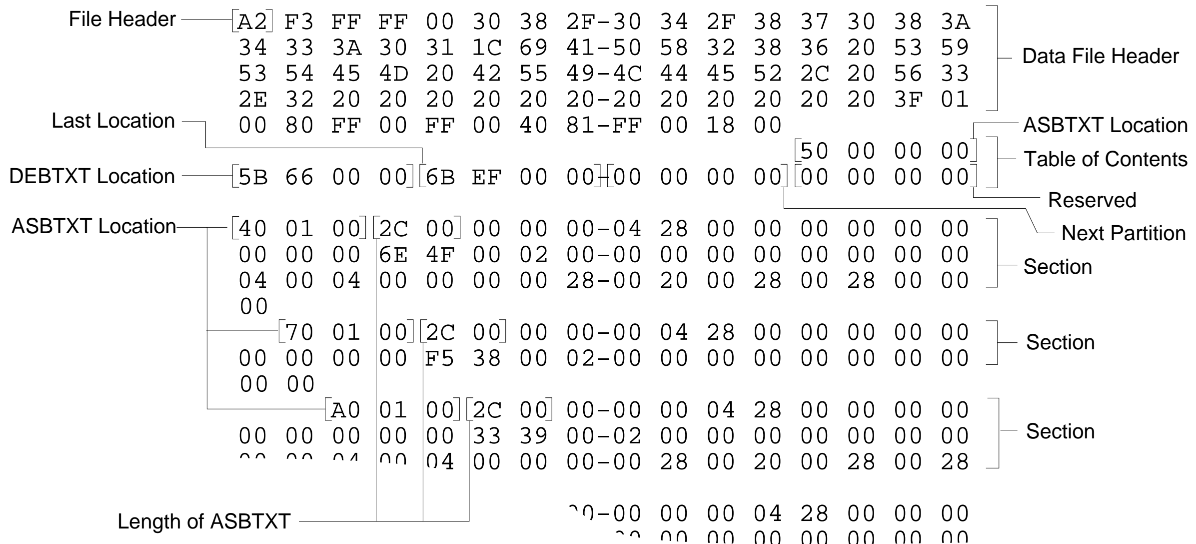
This data translation format is considered by Intel to be proprietary information. Contact your local Intel representative or call (408) 987-8080 for information about the structure of this format.

Intel OMF286 Format, Code 98

The Intel OMF286 format is a dynamically allocatable file format.

This format has three basic parts: the file header, data file module, and a 1-byte checksum. The file header is hexadecimal number (A2) that identifies this file as an Intel OMF 286 format file. See Figure 5-23.

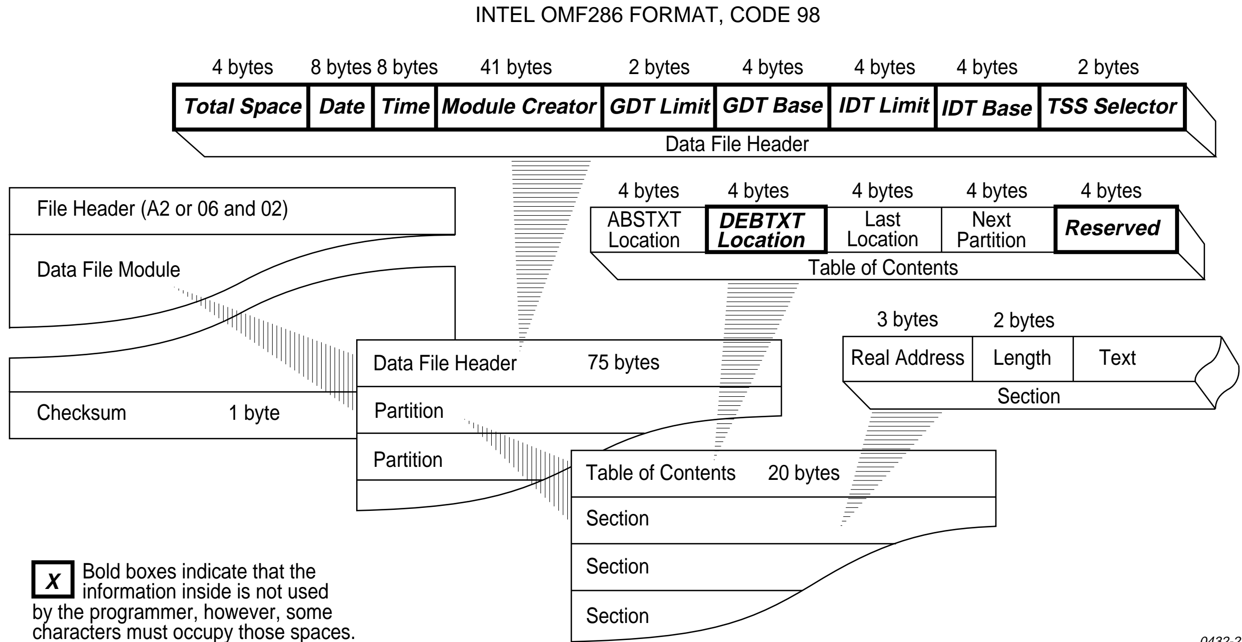
Figure 5-23. Intel OMF286 Format (example)



The first 75 bytes of the data file module is the data file header. The header information is generated and used by the development system and is not used by the programmer, although some characters must fill those bytes. The rest of the data file module consists of one partition.

The partition begins with a 20 byte table of contents. The table of contents specifies the locations of ABSTXT (absolute text), DEBTXT (debug text), the last location of this partition, and the location of the next partition. The OMF286 format consists of only one partition so this field will be zeros. The rest of the partition consists of sections. The actual data is located in the sections. The first 3 bytes in each section specify the real address of the text. The next 2 bytes state the length of the text. The remainder of the section is the text (or data). Following the final section of the final partition is a 1-byte checksum representing the complement of the sum of all the bytes in the file including the header. The sum of the checksum byte and the calculated checksum for the file should equal zero. The programmer ignores this checksum.

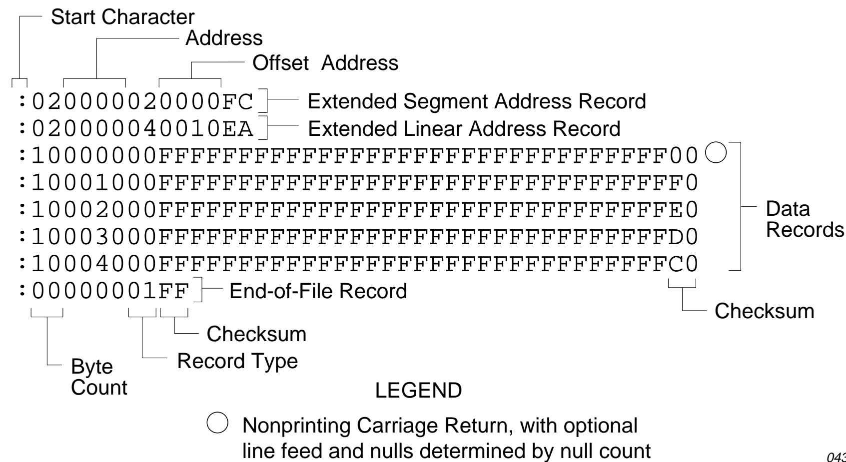
Figure 5-24. Close-up of Intel OMF286 Format



Intel Hex-32, Code 99

The Intel 32-bit Hexadecimal Object file record format has a 9-character (4-field) prefix that defines the start of record, byte count, load address, and record type, and a 2-character checksum suffix. Figure 5-25 illustrates the sample records of this format.

Figure 5-25. Intel Hex-32 Format (example)



The six record types are described below.

00 — Data Record
This record begins with the colon start character, which is followed by the byte count (in hex notation), the address of the first data byte, and the record type (equal to 00). Following these are the data bytes. The checksum follows the data bytes and is the two's complement (in binary) of the preceding bytes in the record, including the byte count, address, record type, and data bytes.

01 — End Record
This end-of-file record also begins with the colon start character and is followed by the byte count (equal to 00), the address (equal to 0000), the record type (equal to 01), and the checksum, FF.

02 — Extended Segment Address Record
This is added to the offset to determine the absolute destination address. The address field for this record must contain ASCII zeros (Hex 30s). This record type defines bits 4 to 19 of the segment base address; it can appear randomly anywhere within the object file and affects the absolute memory address of subsequent data records in the file. The following example illustrates how the extended segment address is used to determine a byte address.

Problem:

Find the address for the first data byte for the following file.

:02 0000 04 0010 EA
:02 0000 02 1230 BA
:10 0045 00 55AA FF ..... BC

Solution:

Step 1. Find the extended linear address offset for the data record (0010 in the example).

Step 2. Find the extended segment address offset for the data record (1230 in the example).

Step 3. Find the address offset for the data from the data record (0045 in the example).

Step 4. Calculate the absolute address for the first byte of the data record as follows:

00100000	Linear address offset, shifted left 16 bits
+ 12300	Segment address offset, shifted left 4 bits
+ 0045	Address offset from data record
00112345	32-bit address for first data byte

The address for the first data byte is 112345.

Note: Always specify the address offset when using this format, even when the offset is zero.

During output translation, the firmware will force the record size to 16 (decimal) if the record size is specified greater than 16. There is no such limitation for record sizes specified less than 16.

03 — Start Segment Address Record

This record, which specifies bits 4-19 of the execution start address for the object file, is not used by the programmer.

04 — Extended Linear Address Record

This record specifies bits 16-31 of the destination address for the data records that follow. It is added to the offset to determine the absolute destination address and can appear randomly anywhere within the object file. The address field for this record must contain ASCII zeros (Hex 30s).

05 — Start Linear Address Record

This record, which specifies bits 16-31 of the execution start address for the object file, is not used by the programmer.

Highest I/O Addresses

The following table shows the highest I/O addresses accepted for each data translation format.

Format Number	Format Name	Highest Address (hex bytes)
01-03	ASCII (BNPF, BHLF, and B10F)	N/A
04	Texas Instruments SDSMAC (320)	1FFFF (FFFF words)
05-07	ASCII (BNPF, BHLF, and B10F)	N/A
11	DEC Binary	N/A
12-13	Spectrum	270F
16	Absolute Binary	N/A
17	LOF	N/A
30-32	ASCII-Octal (Space, Percent, and Apostrophe)	3FFFF (777777 octal)
35-37	ASCII-Octal (Space, Percent, and SMS)	3FFFF (777777 octal)
50-52	ASCII-Hex (Space, Percent, and Apostrophe)	FFFF
55-58	ASCII-Hex (Space, Percent, SMS, and Comma)	FFFF
70	RCA Cosmac	FFFF
80	Fairchild Fairbug	FFFF
81	MOS Technology	FFFF
82	Motorola EXORciser	FFFF
83	Intel Intellec 8/MDS	FFFF
85	Signetics Absolute Object	FFFF
86	Tektronix Hexadecimal	FFFF
87	Motorola EXORmax	FFFFFF
88	Intel MCS-86 Hex Object	FFFFF
89	Hewlett-Packard 64000 Absolute	FFFFFFFF
90	Texas Instruments SDSMAC	FFFF
91, 92	JEDEC Full and Kernel)	N/A
94	Tektronix Hexadecimal Extended	FFFFFFFF
95	Motorola 32 bit (S3 record)	FFFFFFFF
96	Hewlett-Packard UNIX Format	FFFFFFFF
97	Intel OMF 386	FFFFFFFF
98	Intel OMF 286	FFFFFF
99	Intel Hex-32	FFFFFFFF