QUESTION

What is the Intel HEX file format?

ANSWER

The Intel HEX file is an ASCII text file with lines of text that follow the Intel HEX file format. Each line in an Intel HEX file contains one HEX record. These records are made up of hexadecimal numbers that represent machine language code and/or constant data. Intel HEX files are often used to transfer the program and data that would be stored in a ROM or EPROM. Most EPROM programmers or emulators can use Intel HEX files.

Record Format

An Intel HEX file is composed of any number of HEX records. Each record is made up of five fields that are arranged in the following format:

```
:llaaaatt[dd...]cc
```

Each group of letters corresponds to a different field, and each letter represents a single hexadecimal digit. Each field is composed of at least two hexadecimal digits-which make up a byte-as described below:

- **:** is the colon that starts every Intel HEX record.
- ***ll*** is the record-length field that represents the number of data bytes (**dd**) in the record.
- ***aaaa*** is the address field that represents the starting address for subsequent data in the record.
- ***tt*** is the field that represents the HEX record type, which may be one of the following:
  00 - data record
  01 - end-of-file record

02 - extended segment address record

04 - extended linear address record

- ***dd*** is a data field that represents one byte of data. A record may have multiple data bytes. The number of data bytes in the record must match the number specified by the **ll** field.
- ***cc*** is the checksum field that represents the checksum of the record. The checksum is calculated by summing the values of all hexadecimal digit pairs in the record modulo 256 and taking the two's complement.

Data Records

The Intel HEX file is made up of any number of data records that are terminated with a carriage return and a linefeed. Data records appear as follows:

```
:10246200464C5549442050524F46494C4500464C33
```

This record is decoded as follows:

```
:10246200464C5549442050524F46494C4500464C33
|||||||||||                               CC->Checksum
|||||||||DD->Data
|||||||TT->Record Type
|||AAAA->Address
|LL->Record Length
:->Colon
```

where:

- **10** is the number of data bytes in the record.
- **2462** is the address where the data are to be located in memory.
- **00** is the record type 00 (a data record).
- **464C...464C** is the data.
- **33** is the checksum of the record.

Extended Linear Address Records (HEX386)

Extended linear address records are also known as 32-bit address records and HEX386 records. These records contain the upper 16 bits (bits 16-31) of the data address. The extended linear address record always has two data bytes and appears as follows:

```
:02000004FFFFFC
```

where:

- **02** is the number of data bytes in the record.
- **0000** is the address field. For the extended linear address record, this field is always 0000.
- **04** is the record type 04 (an extended linear address record).
- **FFFF** is the upper 16 bits of the address.
- **FC** is the checksum of the record and is calculated as 01h + NOT(02h + 00h + 00h + 04h + FFh + FFh).

When an extended linear address record is read, the extended linear address stored in the data field is saved and is applied to subsequent records read from the Intel HEX file. The linear address remains effective until changed by another extended address record.

The absolute-memory address of a data record is obtained by adding the address field in the record to the shifted address data from the extended linear address record. The following example illustrates this process..

```
Address from the data record's address field      2462
Extended linear address record data field     FFFF
                                              --------
Absolute-memory address                       FFFF2462
```

Extended Segment Address Records (HEX86)

Extended segment address records-also known as HEX86 records-contain bits 4-19 of the data address segment. The extended segment address record always has two data bytes and appears as follows:

```
:020000021200EA
```

where:

- **02** is the number of data bytes in the record.
- **0000** is the address field. For the extended segment address record, this field is always 0000.
- **02** is the record type 02 (an extended segment address record).
- **1200** is the segment of the address.
- **EA** is the checksum of the record and is calculated as 01h + NOT(02h + 00h + 00h + 02h + 12h + 00h).

When an extended segment address record is read, the extended segment address stored in the data field is saved and is applied to subsequent records read from the Intel HEX file. The segment address remains effective until changed by another extended address record.

The absolute-memory address of a data record is obtained by adding the address field in the record to the shifted-address data from the extended segment address record. The following example illustrates this process.

```
Address from the data record's address field    2462

Extended segment address record data field     1200

                                           --------

Absolute memory address                       00014462
```

End-of-File (EOF) Records

An Intel HEX file must end with an end-of-file (EOF) record. This record must have the value 01 in the record type field. An EOF record always appears as follows:

```
:00000001FF
```

where:

- **00** is the number of data bytes in the record.
- **0000** is the address where the data are to be located in memory. The address in end-of-file records is meaningless and is ignored. An address of 0000h is typical.
- **01** is the record type 01 (an end-of-file record).

- **FF** is the checksum of the record and is calculated as
  01h + NOT(00h + 00h + 00h + 01h).

Example Intel HEX File

Following is an example of a complete Intel HEX file:

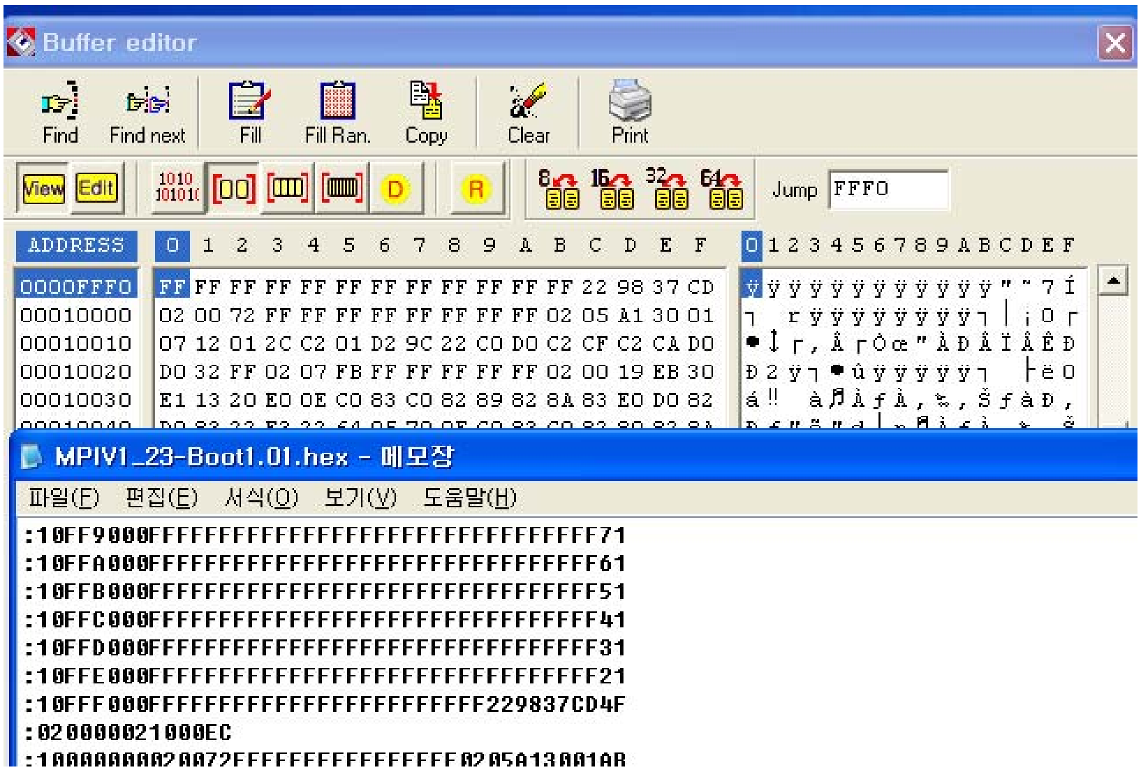
```
:10001300AC12AD13AE10AF1112002F8E0E8F0F2244

:10000300E50B250DF509E50A350CF5081200132259

:03000000020023D8

:0C002300787FE4F6D8FD7581130200031D

:10002F00EFF88DF0A4FFEDC5F0CEA42EFEEC88F016

:04003F00A42EFE22CB

:00000001FF
```

File load illustrated

-EE Tools Maxloader buffer default address is “Byte” mode, click on 8-bit(Byte) when you review the open buffer data.

-For example, the Hex data is below before loading and it is 16-byte mode because (:10) and data for 8-bit address "FFF0" is “FFFFFFFFF....229837CD” with a checksum value 4F.

-After the file is loaded in buffer, check the address “fff0” and see a data stream is loaded in 8-bit address FFF0 correcly.



```
Jump FFF0

ADDRESS   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000FFF0  FF FF FF FF FF FF FF FF FF FF FF FF 22 98 37 CD  ÿÿÿÿÿÿÿÿÿÿÿÿ"˜7Í
00010000  02 00 72 FF FF FF FF FF FF FF FF 02 05 A1 30 01  ┐  rÿÿÿÿÿÿÿÿ┐ │¡0┌
00010010  07 12 01 2C C2 01 D2 9C 22 C0 D0 C2 CF C2 CA D0  •↕┌,Â┌Òœ"ÀÐÂÏÂÊÐ
00010020  D0 32 FF 02 07 FB FF FF FF FF FF 02 00 19 EB 30  Ð2ÿ┐•ûÿÿÿÿÿ┐ ├ë0
00010030  E1 13 20 E0 0E C0 83 C0 82 89 82 8A 83 E0 D0 82  á‼ à♫ÀƒÀ‚‰‚Šƒàð‚
```

MPIV1_23-Boot1.01.hex - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
:10FF9000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF71
:10FFA000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF61
:10FFB000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF51
:10FFC000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF41
:10FFD000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF31
:10FFE000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF21
:10FFF000FFFFFFFFFFFFFFFFFFFFFFFF229837CD4F
:020000021000EC
:10000000020072FFFFFFFFFFFFFFFF0205A13001AB
```

eeTools

4620 Fortran Drive, Ste 102
San Jose, CA 95134
Tel: (408)263-2221
Fax: (408)263-2230
www.eetools.com
info@eetools.com