

[Home \(/\)](#) / [Technical Support](#)

GENERAL: INTEL HEX FILE FORMAT

Information in this article applies to:

- MDK-ARM All Versions
- C166 All Versions
- C251 All Versions
- C51 All Versions

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
 - 05** - start linear address record (MDK-ARM only)
- **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.

Don't show this message again

[Change Settings](#)
(//company/cookiesettings/)

GENERAL: INTEL HEX FILE FORMAT

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:

```
:02000004FFFFFFC
```

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

Start Linear Address Records (MDK-ARM only)

Start linear address records specify the start address of the application. These records contain the full linear 32 bit address. The start linear address record always has four data bytes and appears as follows:

```
:04000005000000CD2A
```

where:

- **04** is the number of data bytes in the record.
- **0000** is the address field. For the start linear address record, this field is always 0000.
- **05** is the record type 05 (a start linear address record).
- **000000CD** is the 4 byte linear start address of the application.
- **2A** is the checksum of the record and is calculated as $01h + NOT(04h + 00h + 00h + 05h + 00h + 00h + 00h + CDh)$.

The Start Linear Address specifies the address of the __main (pre-main) function but not the address of the startup code which usually calls __main after calling SystemInit(). An odd linear start address specifies that __main is compiled for the Thumb instruction set.

The Start Linear Address Record can appear anywhere in hex file. In most cases this record can be ignored because it does not contain information which is needed to program flash memory.

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:

Privacy Policy Update

Arm's Privacy Policy has been updated. By continuing to use our site, you consent to Arm's Privacy Policy. Please review our Privacy Policy (/company/privacy) to learn more about our collection, use and transfers of your data.

Accept and hide this message

Important information

This site uses cookies to store information on your computer. By continuing to use our site, you consent to our cookies (/company/cookiepolicy).

Don't show this message again

[Change Settings](#)
(/company/cookiesettings/)

GENERAL: INTEL HEX FILE FORMAT

```
: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
:10002F00EFF88DFA4FFEDC5F0CEA42EFEEC88F016
:04003F00A42EFE22CB
:00000001FF
```

SEE ALSO

- GENERAL: Converting HEX, Binary, etc. File Formats (<http://www.keil.com/support/docs/4038.htm>)
- µVision: Creating Intel Hex Files for Arm-Based Devices (<http://www.keil.com/support/docs/1759.htm>)
- ULINK: USING ULINK AS A DEVICE PROGRAMMER (<http://www.keil.com/support/docs/3061.htm>)
- ARMLINK: GENERATING BINARY OUTPUT DURING A BUILD (<http://www.keil.com/support/docs/3213.htm>)
- ARMCC: Arm Compiler Output Formats (<http://www.keil.com/support/docs/3206.htm>)

SEE ALSO FOR 8051

- Refer to the **OH51 User's Guide** (<http://www.keil.com/support/man/docs/oh51/>).
- Refer to the **OHX51 User's Guide** (<http://www.keil.com/support/man/docs/ohx51/>).
- LX51: GENERATING HEX FILES FOR BANKED APPLICATIONS (<http://www.keil.com/support/docs/2349.htm>)
- µVision: Hex Output File for a Device Programmer (<http://www.keil.com/support/docs/251.htm>)

FORUM THREADS

The following Discussion Forum (/forum/) threads may provide information related to this topic.

- Reading .hex file (/forum/docs/thread21455.asp)
- Need some help in programming 8051 with keil C (/forum/docs/thread19017.asp)
- HEX386(H167) or HEX-86? (/forum/docs/thread18224.asp)
- Intel HEX - Start Linear Address Record (/forum/docs/thread17360.asp)
- HEX file in LPC2000 Flash Utility (/forum/docs/thread11717.asp)
- Program size calculation (/forum/docs/thread11542.asp)
- How do you convert a HEX file to ASCII (/forum/docs/thread9548.asp)
- hex format (/forum/docs/thread9891.asp)
- Can LOAD file to a specified memory area ? (/forum/docs/thread9295.asp)
- problem with the size of the hex file obtained from the code banking application (/forum/docs/thread8541.asp)

Last Reviewed: Monday, May 7, 2018

Did this article provide the answer you needed?

- ☒ Yes
- ☐ No
- ☐ Not Sure

Submit

Privacy Policy Update

Arm's Privacy Policy has been updated. By continuing to use our site, you consent to Arm's Privacy Policy. Please review our Privacy Policy (/company/privacy) to learn more about our collection, use and transfers of your data.


Accept and hide this message

Important information

This site uses cookies to store information on your computer. By continuing to use our site, you consent to our cookies (/company/cookiepolicy).

Don't show this message again

[Change Settings](#)
(/company/cookiesettings/)

Products (/product/)		Downloads (/download/)	Support (/support/)	Contact
Development Tools	Hardware & Collateral	MDK-Arm (/demo/eval/arm.htm)	Knowledgebase (/support/knowledgebase.asp)	Distributors (/distis/)
Arm (/Arm/)	ULINK Debug Adaptors (/ulink/)	C51 (/demo/eval/c51.htm)	Discussion Forum (/forum/)	Request a Quote (/product/prices.asp)
C166 (/c166/)	Evaluation Boards (/boards2/)	C166 (/demo/eval/c166.htm)	Product Manuals (/support/man/)	Sales Contacts (/company/contact/)
C51 (/c51/)	Product Brochures (/product/brochures.asp)	C251 (/demo/eval/c251.htm)	Application Notes (/appnotes/)	
C251 (/c251/)	Device Database (/dd2/)	File downloads (/download/file/)		
µVision IDE and Debugger (/uvision/)	Distributors (/distis/)			
Cookie Settings (/company/cookiesettings) Terms of Use (/company/terms) Privacy (/company/privacy) Accessibility (/company/accessibility) Trademarks (https://www.arm.com/company/policies/trademarks) Contact Us (/company/contact/) Feedback (/support/feedback.asp)				
Copyright (/company/terms) © 2005-2019 Arm Limited (/company) (or its affiliates). All rights reserved.				
				

Privacy Policy Update

Arm's Privacy Policy has been updated. By continuing to use our site, you consent to Arm's Privacy Policy. Please review our Privacy Policy (/company/privacy) to learn more about our collection, use and transfers of your data.

Accept and hide this message

Important information

This site uses cookies to store information on your computer. By continuing to use our site, you consent to our cookies (/company/cookiepolicy).

Don't show this message again

Change Settings (/company/cookiesettings/)