Intel HEX

From Wikipedia, the free encyclopedia

Not to be confused with Intel hexadecimal notation.

Intel hex

Filename extension	.hex,[1] .h86,[2][3] .hxl,[4] .hxh,[4] .obl,[4] .obh,[4] .mcs,[5] .ihex, .ihe, .ihx, .a43, .a90, .p00[6] to .pff[6]

Intel hexadecimal object file format, **Intel hex format** or **Intellec Hex** is a file format that conveys binary information in ASCII text form.[7] It is commonly used for programming microcontrollers, EPROMs, and other types of programmable logic devices and hardware emulators. In a typical application, a compiler or assembler converts a program's source code (such as in C or assembly language) to machine code and outputs it into a HEX file. Some also use it as a container format holding packets of stream data.[8] Common file extensions used for the resulting files are .HEX[1] or .H86.[2][3] The HEX file is then read by a programmer to write the machine code into a PROM or is transferred to the target system for loading and execution.[9][10]

Contents [hide]

History [edit]

The Intel hex format was originally designed for Intel's Intellec Microcomputer Development Systems (MDS) in 1973 in order to load and execute programs from paper tape. It was also used to specify memory contents to Intel for ROM production.[11] In 1973, Intel's "software group" consisted only of Bill Byerly and Ken Burget, and Gary Kildall as an external consultant.[12][13] Beginning in 1975, the format was utilized by MCS Series II ISIS-II systems, using the file extension HEX.[14] Many PROM and EPROM programming devices accepted this format.

Format [edit]

Intel HEX consists of lines of ASCII text that are separated by line feed or carriage return characters or both. Each text line contains hexadecimal characters that encode multiple binary numbers. The binary numbers may represent data, memory addresses, or other values, depending on their position in the line and the type and length of the line. Each text line is called a *record*.

Record structure [edit]

A record (line of text) consists of six fields (parts) that appear in order from left to right:[9]

1. *Start code*, one character, an ASCII colon ':'. All characters preceding this symbol in a record should be ignored.[11][2][15][16][17][18] In fact, very early versions of the specification even asked for a minimum of 25 NUL characters to precede the first record and follow the last one.[11][19][16][17] However, as this was a little known part of the specification, not all software written copes with this correctly. It allows to store other related information in the same file (and even the same line),[11][18] a facility used by various software development utilites to store symbol tables or additional comments,[20][11][16][21][6] and third-party extensions using other characters as start code like the digit '0' by Keil,[21] '$' by Mostek,[22][23] or '!', '@', '#', '\', '&' and ';' by TDL.[23][24] By convention, '//' is often used for comments. Neither of these extensions may contain any ':' characters as part of the payload.
2. *Byte count*, two hex digits (one hex digit pair), indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 8 (0x08),[6] 16 (0x10)[6] and 32 (0x20) are commonly used byte counts. Not all software copes with counts larger than 16.[5]
3. *Address*, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 kilobyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed as big endian values.
4. *Record type* (see record types below), two hex digits, `00` to `05`, defining the meaning of the data field.
5. *Data*, a sequence of n bytes of data, represented by $2n$ hex digits. Some records omit this field (n equals zero). The meaning and interpretation of data bytes depends on the application. (4-bit data will either have to be stored in the lower or upper half of the bytes, that is, one byte holds only one addressable data item.[11])
6. *Checksum*, two hex digits, a computed value that can be used to verify the record has no errors.

Color legend [edit]

As a visual aid, the fields of Intel HEX records are colored throughout this article as follows:

Start code | Byte count | Address | Record type | Data | Checksum

Checksum calculation [edit]

A record's checksum byte is the two's complement of the least significant byte (LSB) of the sum of all decoded byte values in the record preceding the checksum. It is computed by summing the decoded byte values and extracting the LSB of the sum (i.e., the data checksum), and then calculating the two's complement of the LSB (e.g., by inverting its bits and adding one).

For example, in the case of the record `:0300300002337A1E`, the sum of the decoded byte values is `03` + `00` + `30` + `00` + `02` + `33` + `7A` = `E2`, which has LSB value `E2`. The two's complement of `E2` is `1E`, which is the checksum byte appearing at the end of the record.

The validity of a record can be checked by computing its checksum and verifying that the computed checksum equals the checksum appearing in the record; an error is indicated if the checksums differ. Since the record's checksum byte is the two's complement — and therefore the additive inverse — of the data checksum, this process can be reduced to summing all decoded byte values, including the record's checksum, and verifying that the LSB of the sum is zero. When applied to the preceding example, this method produces the following result: `03` + `00` + `30` + `00` + `02` + `33` + `7A` + `1E` = `100`, which has LSB value `00`.

Text line terminators [edit]

Intel HEX records are usually separated by one or more ASCII line termination characters so that each record appears alone on a text line. This enhances readability by visually delimiting the records and it also provides padding between records that can be used to improve machine parsing efficiency. However, the line termination characters are optional, as the ':' is used to detect the start of a record.[11][2][19][15][16][17][18]

Programs that create HEX records typically use line termination characters that conform to the conventions of their operating systems. For example, Linux programs use a single LF (line feed, hex value `0A`) character to terminate lines, whereas Windows programs use a CR (carriage return, hex value `0D`) followed by a LF.

Record types [edit]

Intel HEX has six standard record types:[9]

Hex code	Record type	Description	Example
`00`	Data	The byte count specifies number of data bytes in the record. The example has `0B` (eleven) data bytes. The 16-bit starting address for the data (in the example at addresses beginning at `0010`) and the data (`61`, `64`, `64`, `72`, `65`, `73`, `73`, `20`, `67`, `61`, `70`).	`:0B0010006164647265737320676170A7`
`01`	End Of File	Must occur exactly once per file in the last record of the file. The byte count is `00`, the address field is typically `0000` and the data field is omitted.	`:00000001FF`
`02`	Extended Segment Address	The byte count is always `02`, the address field (typically `0000`) is ignored and the data field contains a 16-bit segment base address . This is multiplied by 16 and added to each subsequent data record address to form the starting address for the data. This allows addressing up to one megabyte of address space.	`:020000021200EA`
`03`	Start Segment Address	For 80x86 processors, specifies the starting execution address). The byte count is always `04`, the address field is `0000` and the first two data bytes are the CS value, the latter two are the IP value.	`:0400000300003800C1`
`04`	Extended Linear Address	Allows for 32 bit addressing (up to 4 GiB). The byte count is always `02` and the address field is ignored (typically `0000`). The two data bytes (big endian) specify the upper 16 bits of the 32 bit absolute address for all subsequent type `00` records; these upper address bits apply until the next `04` record. The absolute address for a type `00` record is formed by combining the upper 16 address bits of the most recent `04` record with the low 16 address bits of the `00` record. If a type `00` record is not preceded by any type `04` records then its upper 16 address bits default to 0000.	`:02000004FFFFFC`
`05`	Start Linear Address	The byte count is always `04`, the address field is `0000`. The four data bytes represent a 32-bit address value (big-endian). In the case of CPUs that support it, this 32-bit address is the starting execution address).	`:04000005000000CD2A`

Other record types have been used for variants, including `06` by Wayne and Layne,[25] `0A`, `0B`, `0C`, `0D` and `0E` by the BBC/Micro:bit Educational Foundation,[26] and `81`, `82`, `83`, `84`, `85`, `86`, `87` and `88` by Digital Research.[3][15]

Named formats [edit]

The original 4-bit/8-bit *Intellec Hex Paper Tape Format* and *Intellec Hex Computer Punched Card Format* in 1973/1974 supported only one record type `00`.[27][28][20] This was expanded around 1975 to also support record type `01`.[11] It could include an optional header containing a symbol table for symbolic debugging,[20][21][6] all characters in a record preceding the colon are ignored.[11][2]

Around 1978, Intel introduced the new record types `02` and `03` (to add support for the segmented address space of the then-new 8086/8088 processors) in their *Extended Intellec Hex Format*.

Special names are sometimes used to denote the formats of HEX files that employ specific subsets of record types. For example:

- **I8HEX** files use only record types `00` and `01`
- **I16HEX** files use only record types `00` through `03`[7]
- **I32HEX** files use only record types `00`, `01`, `04`, and `05`

File example [edit]

This example shows a file that has four data records followed by an end-of-file record:

```
:10010000214601360121470136007EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Start code | Byte count | Address | Record type | Data | Checksum

Variants [edit]

Besides Intel's own extension, several third-parties have also defined variants and extensions of the Intel hex format, including Digital Research (as in the so called "Digital Research hex format"[3][15]), Zilog, Mostek,[22][23] TDL,[23][24] Texas Instruments, Microchip,[29] c't, Wayne and Layne,[25] and BBC/Micro:bit Educational Foundation (with its "Universal Hex Format"[26]). These can have information on program entry points and register contents, a swapped byte order in the data fields, fill values for unused areas, fuse bits, and other differences.

The Digital Research hex format for 8086 processors supports segment information by adding record types to distinguish between code, data, stack, and extra segments.[2][3][15]

Most assemblers for CP/M-80 (and also XASM09 for the Motorola 6809) don't use record type 01h to indicate the end of a file, but use a zero-length data type 00h entry instead. This eases the concatenation of multiple hex files.[30][31][1]

Texas Instruments defines a variant where addresses are based on the bit-width of a processor's registers, not bytes.

Microchip defines variants INTHX8S[32] (INHX8L,[1] INHX8H[1]), INHX8M,[32][1][33] INHX16[32] (INHX16M[1]) and INHX32[34] for their PIC microcontrollers.

Alfred Arnold's cross-macro-assembler AS,[1] Werner Hennig-Roleff's 8051-emulator SIM51,[21] and Matthias R. Paul's cross-converter BINTEL are also known to define extensions to the Intel hex format.

See also [edit]

- Binary-to-text encoding, a survey and comparison of encoding algorithms
- MOS Technology file format
- Motorola S-record hex format
- Tektronix hex format

References [edit]

1. ^ a b c d e f g h Arnold, Alfred (2020) [1996, 1989]. "6.3. P2HEX". *Macro Assembler AS - User's Manual*. V1.42. Translated by Arnold, Alfred; Hilse, Stefan; Kanthak, Stephan; Sellke, Oliver; De Tomasi, Vittorio. Archived from the original on 2020-02-28. Retrieved 2020-02-28. "[…] For the PIC microcontrollers, the switch -m <0..3> allows to generate the three different variants of the Intel Hex format. Format 0 is INHX8M which contains all bytes in a Lo-Hi-Order. Addresses become double as large because the PICs have a word-oriented address space that increments addresses only by one per word. […] With Format 1 (INHX16M), bytes are stored in their natural order. This is the format Microchip uses for its own programming devices. Format 2 (INHX8L) resp. 3 (INHX8H) split words into their lower resp. upper bytes. […] Unfortunately, one finds different statements about the last line of an Intel-Hex file in literature. Therefore, P2HEX knows three different variants that may be selected […] :00000001FF […] :00000001 […] :0000000000 […] By default, variant 0 is used which seems to be the most common one. […] If the target file name does not have an extension, an extension of HEX is supposed. […]"
2. ^ a b c d e f "3.1. Intel 8086 Hex File Format". *CP/M-86 Operating System - System Guide* (PDF) (2nd printing, 1st ed.). Pacific Grove, California, USA: Digital Research. June 1981. pp. 15–16. Archived (PDF) from the original on 2020-02-28. Retrieved 2020-02-28. p. 16: "[…] The following are output from ASM-86 only: 81 same as 00, data belongs to code segment […] 82 same as 00, data belongs to data segment […] 83 same as 00, data belongs to stack segment […] 84 same as 00, data belongs to extra segment […] 85 paragraph address for absolute code segment […] 86 paragraph address for absolute data segment […] 87 paragraph address for absolute stack segment […] 88 paragraph address for absolute extra segment […] All characters preceding the colon for each record are ignored. […]" (17 pages)
3. ^ a b c d e "Appendix C. ASM-86 Hexadecimal Output Format". *CP/M-86 - Operating System - Programmer's Guide* (PDF) (3 ed.). Pacific Grove, California, USA: Digital Research. January 1983 [1981]. pp. 97–100. Archived (PDF) from the original on 2020-02-27. Retrieved 2020-02-27. pp. 97–99: "[…] The Intel format is identical to the format defined by Intel for the 8086. The Digital Research format is nearly identical to the Intel format, but adds segment information to hexadecimal records. Output of either format can be input to GENCMD, but the Digital Research format automatically provides segment identification. A segment is the smallest unit of a program that can be relocated. […] It is in the definition of record types 00 and 02 that Digital Research's hexadecimal format differs from Intel's. Intel defines one value each for the data record type and the segment address type. Digital Research identifies each record with the segment that contains it. […] 00H for data belonging to all 8086 segments […] 81H for data belonging to the CODE segment […] 82H for data belonging to the DATA segment […] 83H for data belonging to the STACK segment […] 84H for data belonging to the EXTRA segment […] 02H for all segment address records […] 85H for a CODE absolute segment address […] 86H for a DATA segment address […] 87H for a STACK segment address […] 88H for an EXTRA segment address […]" [1] (1+viii+122+2 pages)
4. ^ a b c d "The Interactive Disassembler - Hexadecimal fileformats". Hex-Rays. 2006. Archived from the original on 2020-03-01. Retrieved 2020-03-01. [2]
5. ^ a b "AR#476 PROMGen - Description of PROM/EEPROM file formats: MCS, EXO, HEX, and others". Xilinx. 2010-03-08. Intel MCS-86 Hexadecimal Object - File Format Code 88. Archived from the original on 2020-03-03. Retrieved 2020-03-03.
6. ^ a b c d e f Roche, Emmanuel (2000-04-01). "The Intel HEX File Format". France: Newsgroup: comp.os.cpm. INTELHEX.WS4. Archived from the original on 2021-12-08. Retrieved 2021-12-08. "[…] the Intel HEX file format can contain much more than the "data bytes". As long as the lines do not start with a colon (":"), they can contain anything that you want. […] I once saw a big HEX file […] It contained, at the beginning, the source code of a PL/M program, followed, at the end, by the resulting HEX file produced by the PL/M compiler. […] I found another HEX file containing several lines of comments, not at the beginning or at the end, but separating several lines of "absolute records". […] it was from an "(Intel) 8008 Simulator". So, at the beginning of its use, it was well known that HEX files could contain explanations. […]"
7. ^ a b "Appendix D. MCS-86 Absolute Object File Formats: Hexadecimal Object File Format". *8086 Family Utilities - User's Guide for 8080/8085-Based Development Systems* (PDF). Revision E (A620/5821 6K DD ed.). Santa Clara, California, USA: Intel Corporation. May 1982 [1980, 1978]. pp. D-8–D-13. Order Number: 9800639-04. Archived (PDF) from the original on 2020-02-29. Retrieved 2020-02-29.
8. ^ "LT Programming Hex File Format Documentation -- In Circuit Programming". Analog Devices, Inc. / Linear Technology. 2021. Archived from the original on 2021-03-07. Retrieved 2021-12-11.
9. ^ a b c *Hexadecimal Object File Format Specification*. Revision A. Intel. 1998 [1988-01-06]. Retrieved 2019-07-23. [3][4][5][6] (11 pages)
10. ^ "General: Intel Hex File Format". ARM Germany GmbH. Archived from the original on 2020-02-27. Retrieved 2017-09-06.
11. ^ a b c d e f g h i "Chapter 6. Microcomputer System Component Data Sheet - EPROMs and ROM: I. PROM and ROM Programming Instructions - B1. Intellec Hex Paper Tape Format / C1. Intellec Hex Computer Punched Card Format". *MCS-80 User's Manual (With Introduction to MCS-85)*. Intel Corporation. October 1977 [1975]. pp. 6-75–6-78. 98-153D. Retrieved 2020-02-27. p. 6-76: "[…] In the Intel Intellec Hex Format, a data field can contain either 8 or 4-bit data. Two ASCII hexadecimal characters must be used to represent both 8 and 4-bit data. In the case of 4-bit data, only one of the characters is meaningful and must be specified on the Intel PROM/ROM Order Form. […] Preceding the first data field and following the last data field there must be a leader/trailer length of at least 25 null characters. Comments (except for a colon) may be placed on the tape leader. […] If the data is 4-bit, then either the high or low-order digit represents the data and the other digit of the pair may be any ASCII hexadecimal digit. […]" [7][8] (NB. This manual also describes a "BPNF Paper Tape Format", a "Non-Intellec Hex Paper Tape Format" and a "PN Computer Punched Card Format".)
12. ^ Kildall, Gary Arlen (January 1980). "The History of CP/M, The Evolution of an Industry: One Person's Viewpoint". *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*. **5** (1): 6–7. #41. Archived from the original on 2016-11-24. Retrieved 2013-06-03. "[…] Programs had been written and tested by Intel's software group, consisting of myself and two other people, and we were ready for the real machine. […]"
13. ^ Kildall, Gary Arlen (2016-08-02) [1993]. Kildall, Scott; Kildall, Kristin (eds.). *Computer Connections: People, Places, and Events in the Evolution of the Personal Computer Industry* (Manuscript, part 1). Kildall Family. Archived (PDF) from the original on 2016-11-17. Retrieved 2016-11-17. (NB. Part 2 not released due to family privacy reasons.)
14. ^ Feichtinger, Herwig (1987). "1.8.5. Lochstreifen-Datenformate: Das Intel-Hex-Format" [1.8.5. Paper tape data formats]. *Arbeitsbuch Mikrocomputer* [*Microcomputer work book*] (in German) (2 ed.). Munich, Germany: Franzis-Verlag GmbH. pp. 240–243 [243]. ISBN 3-7723-8022-0. (NB. The book also describes a BNPF, a Motorola S and a MOS 6502 hex format.)
15. ^ a b c d e "4.3 Intel Hexadecimal File Format". *Concurrent CP/M Operating System - Programmer's Reference Guide* (PDF) (1 ed.). Pacific Grove, California, USA: Digital Research Inc. January 1984. pp. 4-9–4-12. Archived (PDF) from the original on 2021-12-11. Retrieved 2021-12-11. pp. 4-11–4-12: "[…] The following are output from ASM-86 only: 81 same as 00, data belongs to Code Segment […] 82 same as 00, data belongs to Data Segment […] 83 same as 00, data belongs to Stack Segment […] 84 same as 00, data belongs to Extra Segment […] *85 paragraph address for absolute Code Segment […] *86 paragraph address for absolute Data Segment […] *87 paragraph address for absolute Stack Segment […] *88 paragraph address for absolute Extra Segment […] * 85, 86, 87, and 88 are Digital Research Extensions. […] All characters preceding the colon for each record are ignored. […]" (346 pages) (NB. This manual marks only types 85, 86, 87 and 88 as Digital Research extensions, as if types 81, 82, 83, 84 were not.)
16. ^ a b c d e "2.8. Microprocessor Formats, 2.8.1. Input Requirements: Intel Intellec 8/MDS Format. Select Code 83". *Operator Guide To Serial I/O Capabilities of Data I/O Programmers - Translation-Format Package* (PDF). Revision C. Data I/O Corporation. October 1980. p. 2-10. 055-1901. Archived (PDF) from the original on 2020-03-01. Retrieved 2020-03-01. p. 2-10: "[…] Input […] This space can be used for line feed, carriage return or comments. […] Output […] 2) Each line ends with nonprinting line feed, carriage returns and nulls. […]" (1+ii+19 pages)
17. ^ a b c "Intel Intellec 8/MDS Format, Code 83". *Translation File Formats* (PDF). Data I/O Corporation. 1987-09-03. pp. 22, 26–27, 52–53, 54. Archived (PDF) from the original on 2020-03-01. Retrieved 2020-03-01. pp. 22, 26, 52: "[…] Nonprinting Carriage Return, line feed, and nulls determined by null count […]" (56 pages)
18. ^ a b c "Appendix B: Intel Hex and Intel Extended Hex Format - B.1 Common Format". *Fujitsu Semiconductor Controller Manual: FR/F²MC Family Softune Linkage Kit Manual for V3* (PDF). Fujitsu Limited. 2001. pp. 319–525 [320–321]. Archived (PDF) from the original on 2021-12-12. Retrieved 2021-12-12. p. 321: "[…] (g) Generally, a control code (such as CR and LF) is added. Data in this field is skipped until the start character ":" of (a) appears. Since the (a), (b), (c), (d), and (f) fields always exist, the minimum length of a record is 11 bytes long and the maximum length is 521 bytes long. […]" (4+xx+350 pages)
19. ^ a b "1.6.4 PIP". *CP/M Operating System Manual* (First printing ed.). Pacific Grove, California, USA: Digital Research. July 1982 [1976]. pp. 17–23. Retrieved 2021-12-12. pp. 19–21: "[…] PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex-formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the reread, a single carriage return is typed at the console, and PIP will attempt another read. If the tape position cannot be properly read, the user continues the read (by typing a return following the error message), and enters the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is an RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard the read operation is terminated normally. […] `PIP PUN:=NUL:,X.ASM,EOF:,NUL:` […] Send 40 nulls to the punch device; copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters. […] `H` […] HEX data transfer: all data are checked for proper Intel hex file format. Nonessential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur. […] `I` […] Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter). […] `PIP PUN:=X.HEX[i],Y.ZOT[h]` […] First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; continue the transfer of data by reading Y.ZOT, which contains HEX records, including any ":00" records it contains. […]" [9] (6+250 pages)
20. ^ a b c "Appendix A. Hexidecimal Object Tape". *MCS-8 A Guide to PL/M programming*. Rev 1 (printed September 1974 ed.). March 1974 [September 1973]. (NB. Shows an example also including a header with symbol names to be processed by Intel ISIS's HEXOBJ command as well as by INTERP/8 or INTERP/80 for symbolic debugging. This optional header is not documented as part of Intel hex or BNPF formats but in Intel's PL/M and assembler programming manuals producing such symbol tables.)
21. ^ a b c d Hennig-Roleff, Werner (1993-02-01) [1988]. "HEX.DOC: Intel-HEX-Format". *SIM51*. 1.04 (in German). Archived from the original on 2017-08-11. Retrieved 2021-12-08. "[…] Beim Absolut-Hex Konvertierprogramm von Keil können optional […] Symbol-Informationen in den Hex-File aufgenommen werden. Die Symbol-Informationen stehen dabei am Anfang des Files, vor dem ersten ':'. Die Symbol-Informationen sind allerdings nicht sehr aussagekräftig, da nicht unterschieden wird zwischen Modul-Name, CODE, XDATA, DATA, IDATA, BIT, NUMBER. Für jeden Symboleintrag werden nur ASCII-Zeichen verwendet. Pro Zeile ist 1 Symbol angeschrieben und zwar in der Form: "0 SymbolName Wert" […]" [10][11] (NB. This is an older version of SIM51, the software and documentation was maintained up to 1996. The HEX.DOC file also describes the EASM51 hex format used by the elektor assembler.)
22. ^ a b Formaniak, Peter G.; Leitch, David (July 1977). "A Proposed Microprocessor Software Standard". *BYTE - the small systems journal*. Technical Forum. Peterborough, New Hampshire, USA: Byte Publications, Inc. **2** (7): 34, 62–63. ark:/13960/t32245485. Retrieved 2021-12-06. (3 pages) (NB. Describes an extension of the Intel hex format by Mostek.)
23. ^ a b c d Ogdin, Carol Anne; Colvin, Neil; Pittman, Tom; Tubb, Philip (November 1977). "Relocatable Object Code Formats". *BYTE - the small systems journal*. Technical Forum. Peterborough, New Hampshire, USA: Byte Publications, Inc. **2** (11): 198–205. ark:/13960/t59c88b4h, ark:/13960/t3kw76j24. Retrieved 2021-12-06. (8 pages) (NB. Besides others describes an incompatible extension of the Intel hex format used by Technical Design Labs (TDL).)
24. ^ a b Kreidl, Günter (June 1981). "Relocator: Das TDL-Format". Hardware. *nascom journal - Zeitschrift für Anwender des NASCOM 1 oder NASCOM 2* (in German). Germersheim, Germany: Verlag NASCOM Journal, MK-Systemtechnik. **2** (6): 12–14 [12]. Archived from the original on 2021-12-01. Retrieved 2021-12-11. (20 pages) (NB. Shows a variant of the TDL format, which itself is a variant of the Intel hex format.)
25. ^ Beckler, Matthew L. (2016-07-25) [2016-07-19]. "Blinky Grid - serial optical bit stream". *Discourse*. Minneapolis, Minnesota, USA: Wayne and Layne, LLC. Archived from the original on 2021-12-11. Retrieved 2021-12-11.
26. ^ a b "micro:bit Universal Hex Format Specification - Specification for the micro:bit Universal Hex Format". *micro:bit*. 0.4.0. Micro:bit Educational Foundation. 2021-01-26 [2020]. Archived from the original on 2021-08-14. Retrieved 2021-12-08. [12][13] (NB. This represents kind of a fat hex file format.)
27. ^ *Intellec 8 Microcomputer System Operator's Manual*. Intel Corporation. November 1973.
28. ^ "Appendix D. Hexadecimal Program Tape Format". *Intellec 8/MOD 80 Operators Manual*. Intel. June 1974. 98-003A. "[…] Frames 7,8: Record Type […] Two ASCII characters. Currently (1974), all records are type 0. This field is reserved for future expansion […]" [14]
29. ^ "PIC Microcontrollers: PIC Hex File Format". *Kanda Electronics Blog*. Canolafan, Llanafan, Aberystwyth, Wales, UK: Embedded Results Ltd. 2012-04-26. Archived from the original on 2021-08-16. Retrieved 2021-12-11.
30. ^ Zschocke, Jörg (November 1987). "Nicht nur Entwicklungshilfe - Down-Loading für Einplatinencomputer am Beispiel des EPAC-09: Intel-Hex-Format". *c't - magazin für computertechnik* (in German). Vol. 1987 no. 11. Verlag Heinz Heise GmbH & Co. KG. pp. 198, 200, 202–203, [200]. ISSN 0724-8679. "[…] Den Vorspann beschließt ein Byte, dessen Wert den Typ des Blockes angibt: 0 = Datenblock, 1 = Endblock. Auf diese Unterscheidung kann jedoch verzichtet werden, wenn sich ein Endblock auch durch eine Blocklänge gleich Null eindeutig kennzeichnen läßt. (So verfahren die meisten Assembler unter CP/M, auch der XASM09; das Typbyte ist dann immer Null). […]" [15] (NB. XASM09 is a Motorola 6809 assembler.)
31. ^ Prior, James E. (1989-02-24). "Re: Intel hex (*.HEX) format questions". Newsgroup: comp.os.cpm. Retrieved 2020-02-27.
32. ^ a b c "PIC16C5X Programming Specification 5.0 - PIC16C5X Hex Data Formats: 5.1. 8-Bit Split Intellec Hex Format (INHX8S) / 5.2. 8-Bit Merged Intellec Hex Format (INHX8M) / 5.3. 16-Bit Hex Format / 5.4. 8-Bit Word Format / 5.5. 16-Bit Word Format". *Microchip Databook* (1994 ed.). Microchip Technology Inc. April 1994. pp. 3-10–3-11, 9-10, 9-15, 9-17, 9-21, 9-23, 9-27. DS00018G. Retrieved 2020-02-28. "[…] Assemblers for the PIC16C5X can produce PIC16C5X object files in various formats. A PIC16C5X programmer must be able to accept and send data in at least one of following formats. The 8-bit merged (INHX8M) format is preferred. […] format […] INHX8S […] produces two 8-bit Hex files. One file will contain the address / data pairs for the high order 8-bits and the other file will contain the low order 8-bits. File extensions for the object code will be '.obl' and '.obh' for low and high order files […] format […] INHX8M […] produces one 8-bit Hex file with a low byte / high byte combination. Since each address can only contain 8 bits in this format, all addresses will be doubled. File extensions for the object code will be '.obj' […] format […] INHX16 […] produces one 16-bit Hex file. File extension for the object code will be '.obj'. […]" [16][17]
33. ^ Beard, Brian (2016) [2010]. "Microchip INHX8M HEX-record Format". *Lucid Technologies*. Archived from the original on 2020-02-28. Retrieved 2020-02-28.
34. ^ Beard, Brian (2016) [2013]. "Microchip INHX32 HEX-record Format". *Lucid Technologies*. Archived from the original on 2020-02-28. Retrieved 2020-02-28.

Further reading [edit]

- "How Do I Interpret Motorola S & Intel HEX Formatted Data? Intel Hex-32, Code 99". *Home > Hardware > … > In-circuit Test Systems > Automated Test Equipment [Discontinued] > Details*. Keysight Technologies. Archived from the original on 2020-03-01. Retrieved 2020-03-01.
- Bergmans, San (2019-06-02) [2001]. "Intel HEX Format". *SB-Projects*. Archived from the original on 2020-03-01. Retrieved 2020-03-01.
- Beard, Brian (2016) [2007]. "Intel HEX-record Format". *Lucid Technologies*. Archived from the original on 2020-02-28. Retrieved 2020-02-28.
- Anderson, Thomas N. (February 1998). "Intel Hex Word Address Object Format". *The Telemark Assembler (TASM) User's Manual* (PDF). 3.1. Issaquah, Washington, USA: Squak Valley Software. pp. 25–26. Archived (PDF) from the original on 2021-12-11. Retrieved 2021-12-11. "Intel Hex Word Address Object Format […] This format is identical to the Intel Hex Object Format except that the address for each line of object code is divided by two thus converting it to a word address (16 bit word). All other fields are identical. Here is an example: […] :180800000102030405060708090A0B0C0D0E0F101112131415161718AC […] :02080C00191AA3 […] :00000001FF […]" (32 pages)

External links [edit]

- binex - a converter between Intel HEX and binary for Windows.
- SRecord, a converter between Intel HEX and binary for Linux (usage), C++ source code.
- kk_ihex, open source C library for reading and writing Intel HEX
- libgis, open source C library that converts Intel HEX, Motorola S-Record, Atmel Generic files.
- bincopy is a Python package for manipulating Intel HEX files.

Categories: Binary-to-text encoding formats | Embedded systems | Computer file formats