|  |
| --- |
| 동원올레브 |
| DSP원격제어프로그램 |
| 프로그램 설계 |

|  |
| --- |
| 이경훈  2016-3-15 |

내용

[프로그램 프로토콜 1](#_Toc30086920)

[메모리 2](#_Toc30086921)

[인텔 HEX 2](#_Toc30086922)

[**형식**[편집] 2](#_Toc30086923)

[**레코드 구조**[편집] 3](#_Toc30086924)

[**줄 종료문자**[편집] 3](#_Toc30086925)

[**레코드 종류**[편집] 3](#_Toc30086926)

[**다른 이름**[편집] 5](#_Toc30086927)

[**파일 예제**[편집] 5](#_Toc30086928)

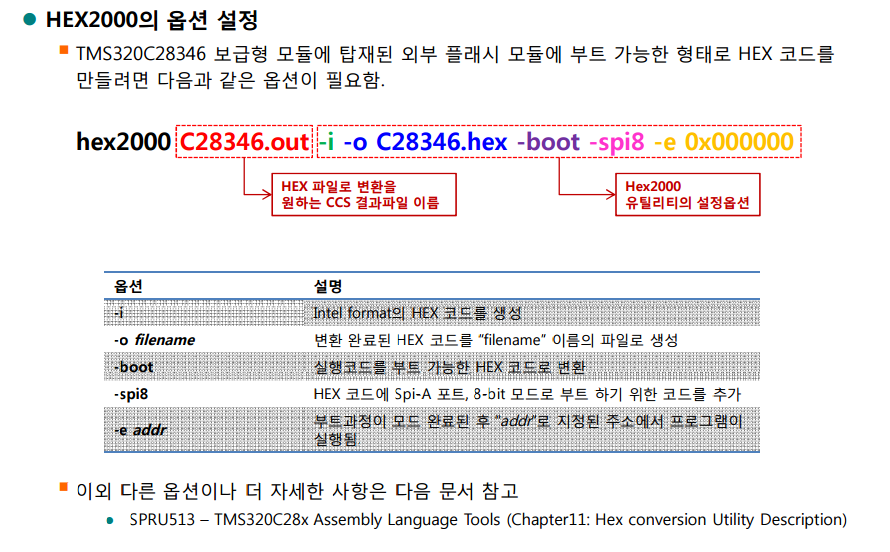
프로그램 프로토콜

먼저 전송을 위하여 원격제어 프로그램의 경우 FALSHA에 설정하였으며 테스트용 APP프로그램은 FLASHK에 설정하였다. 통신 명령은 다음과 같은 명령을 가진다.

|  |  |  |
| --- | --- | --- |
| STX | CMD | ETX |
| [ | ‘S’ : 프로그램 전송  ‘I’ : 정보요구  ‘G’ : 프로그램 실행 | ] |

HEX2000은 기본 설정법은 아래와 같으며 romwidth = 16bit와 인텔HEX포맷으로 설정한다.

Ex)HEX2000 a.out –I –romwidth=16 –o a.hex



인텔 포맷은 다음과 같다

메모리

원격제어 프로그램의 경우 FALSHA에 설정이 되면 테스트용 APP프로그램은 FLASHK에 설정하였으며 HEX포멧은 인텔 포맷으로 설정한다.

인텔 포맷은 다음과 같다

인텔 HEX

위키백과, 우리 모두의 백과사전.

**인텔 HEX**는 [ASCII](https://ko.wikipedia.org/wiki/%EB%AF%B8%EA%B5%AD%EC%A0%95%EB%B3%B4%EA%B5%90%ED%99%98%ED%91%9C%EC%A4%80%EB%B6%80%ED%98%B8) 텍스트 형식으로 이진 정보를 전달하는 [파일 형식](https://ko.wikipedia.org/wiki/%ED%8C%8C%EC%9D%BC_%ED%98%95%EC%8B%9D)이다. 이것은 [마이크로컨트롤러](https://ko.wikipedia.org/wiki/%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%BB%A8%ED%8A%B8%EB%A1%A4%EB%9F%AC), [EPROM](https://ko.wikipedia.org/wiki/EPROM) 그리고 다른 [논리 장치](https://ko.wikipedia.org/wiki/%EC%84%A4%EA%B3%84_%EA%B0%80%EB%8A%A5_%EB%85%BC%EB%A6%AC_%EC%86%8C%EC%9E%90)의 프로그래밍를 위해 흔히 사용된다. 대표적으로 [컴파일러](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%8C%8C%EC%9D%BC%EB%9F%AC)나 [어셈블러](https://ko.wikipedia.org/wiki/%EC%96%B4%EC%85%88%EB%B8%94%EB%A6%AC%EC%96%B4#.EC.96.B4.EC.85.88.EB.B8.94.EB.9F.AC)는 [컴퓨터 프로그램](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%93%A8%ED%84%B0_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8)의 [소스 코드](https://ko.wikipedia.org/wiki/%EC%86%8C%EC%8A%A4_%EC%BD%94%EB%93%9C)를 (예를 들어 [C](https://ko.wikipedia.org/wiki/C_(%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D_%EC%96%B8%EC%96%B4))나 [어셈블리어](https://ko.wikipedia.org/wiki/%EC%96%B4%EC%85%88%EB%B8%94%EB%A6%AC%EC%96%B4)) [기계어](https://ko.wikipedia.org/wiki/%EA%B8%B0%EA%B3%84%EC%96%B4)로 변환하고, 이를 HEX 파일로 출력한다. 그리고 나서 이 HEX 파일은 [소자 프로그래머](https://ko.wikipedia.org/wiki/%EC%86%8C%EC%9E%90_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8)가 읽어서 [ROM](https://ko.wikipedia.org/wiki/%EA%B3%A0%EC%A0%95_%EA%B8%B0%EC%96%B5_%EC%9E%A5%EC%B9%98)를 굽거나, 메모리에 올리고 실행하기 위해 목적 시스템으로 전송된다.[[1]](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%85%94_HEX#cite_note-1)

**형식**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=1)]

인텔 HEX는 [LF](https://ko.wikipedia.org/wiki/%EC%83%88%EC%A4%84_%EB%AC%B8%EC%9E%90)나 [CR](https://ko.wikipedia.org/wiki/%EC%BA%90%EB%A6%AC%EC%A7%80_%EB%A6%AC%ED%84%B4)로 구분된 여러줄의 [ASCII](https://ko.wikipedia.org/wiki/%EB%AF%B8%EA%B5%AD%EC%A0%95%EB%B3%B4%EA%B5%90%ED%99%98%ED%91%9C%EC%A4%80%EB%B6%80%ED%98%B8) 텍스트로 구성된다. 각 줄은 다수의 바이너리 숫자를 [부호화](https://en.wikipedia.org/wiki/Binary-to-text_encoding)한 [16진수](https://ko.wikipedia.org/wiki/%EC%8B%AD%EC%9C%A1%EC%A7%84%EB%B2%95) 문자를 담고 있다. 이 바이너리 숫자가 줄ㄴ에서의 위치, 종류, 줄의 길이에 따라 [메모리 주소](https://ko.wikipedia.org/wiki/%EB%A9%94%EB%AA%A8%EB%A6%AC_%EC%A3%BC%EC%86%8C)나 기타의 값인 데이터를 나타낼 것이다. 각 텍스트 줄을 레코드(*record*)라고 한다.

**레코드 구조**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=2)]

하나의 [레코드](https://en.wikipedia.org/wiki/Record_(computer_science))(텍스트 한줄)는 왼쪽에서 오른쪽으로 순서대로 여섯가지 [필드](https://ko.wikipedia.org/w/index.php?title=Field_(computer_science)&action=edit&redlink=1)(부분)로 나타난다.

1. **시작코드(Start code)**, 한 문자, ASCII 문자 ':'(colon).
2. **바이트 개수(Byte count)**, 16진수 문자 두 자리, 데이터 필드에 있는 바이트(16진수 두 문자의 쌍)의 수를 가리킨다. 최대 바이트 개수는 255 (0xFF)다. 16 (0x10) 과 32 (0x20)가 흔히 쓰인다.
3. **주소(Address)**, 16진수 문자 네 자리, 데이터의 16비트 시작 주소의 옵셋 값을 표현한다. 데이터의 물리 주소는 앞서 지정된 기준 주소에 이 옵셋을 더해서 구한다. 이런 식으로 16비트 주소의 한계인 64 킬로바이트를 넘는 주소를 지정 한다. 기준 주소의 기본값은 0이고, 여러 종류의 레코드로 값을 바꿀 수 있다. 기준 주소와 주소는 항상 [빅 엔디언](https://ko.wikipedia.org/wiki/%EC%97%94%EB%94%94%EC%96%B8)으로 표현된다.
4. **레코드 종류(Record type)** (아래 [레코드 종류](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%85%94_HEX#.EB.A0.88.EC.BD.94.EB.93.9C_.EC.A2.85.EB.A5.98) 참조), 16진수 문자 두 자리, *00*에서 *05*, 데이터 필드의 종류를 정의한다.
5. **데이터(Data)**, 2*n* 개의 16진수 문자로 표현된 *n* 바이트의 데이터 열이다. 어떤 레코드는 이 필드가 빠져 있다(*n*이 0). 데이터 바이트들의 의미나 해석은 응용프로그래밍에 달렸다.
6. **체크섬(**[**Checksum**](https://ko.wikipedia.org/w/index.php?title=Checksum&action=edit&redlink=1)**)**, 16진수 문자 두 자리, 레코드에 오류가 없음을 입증하는데 이용될 수 있는 계산된 값이다.

**색 일러두기**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=3)]

이 문서에서 Intel HEX 레코드의 필드들은 눈에 잘 띄도록 다음처럼 색칠된다.

     시작코드      바이트 개수      주소      레코드 종류      데이터      체크섬

**체크섬 계산**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=4)]

레코드의 [체크섬](https://ko.wikipedia.org/wiki/%EC%B2%B4%ED%81%AC%EC%84%AC) 바이트는 레코드에서 체크섬에 앞서 있는 모든 복호화된 바이트 합의 [최하위 바이트](https://ko.wikipedia.org/wiki/%EC%B5%9C%ED%95%98%EC%9C%84_%EB%B9%84%ED%8A%B8)(LSB)의 [2의 보수](https://ko.wikipedia.org/wiki/2%EC%9D%98_%EB%B3%B4%EC%88%98)이다. 이것은 복호화된 값(파일에 있는 16진수 문자가 아니라 그것을 두 개씩 짝지어 바이트로 해석한 값)들을 모두 더하고, 더한 값의 LSB(256으로 나눌 때 나머지)를 뽑아서 그것의 2의 보수(모든 비트를 [뒤집고](https://ko.wikipedia.org/wiki/1%EC%9D%98_%EB%B3%B4%EC%88%98) 1를 더한다)를 구한 것이다.

예를 들어, :0300300002337A1E 이런 레코드에서 길이부터 체크썸 전까지를 합하면 03 + 00 + 30 + 00 + 02 + 33 + 7A = E2이다. E2의 [2의 보수](https://ko.wikipedia.org/wiki/2%EC%9D%98_%EB%B3%B4%EC%88%98)는 1E이다.

레코드의 유효함은 체크섬을 계산하고, 계산한 체크섬과 레코드에 있는 체크섬이 같다는 것을 확인해서 검사될 수 있다; 체크섬이 다르다는 것은 오류가 있음을 가리키는 것이다. 레코드의 체크섬이 데이터 체크섬의 음수라서 체크섬을 포함하는 전체 복호화된 바이트를 더하면 LSB는 0이 된다.

**줄 종료문자**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=5)]

인텔 HEX 레코드는 각 레코드가 한 줄의 텍스트에 날 수 있도록 하나 이상의 ASCII 줄 끝 문자로 구분된다. 이것은 시가적인 가독성을 높여주고, 레코드 사이에 끼워져 있어 기계의 [구문 분석](https://ko.wikipedia.org/wiki/%EA%B5%AC%EB%AC%B8_%EB%B6%84%EC%84%9D) 효율을 향상시키는데 이용될 수 있다.

HEX 레코드를 만들어내는 프로그램은 전형적으로 사용하는 [운영 체제](https://ko.wikipedia.org/wiki/%EC%9A%B4%EC%98%81_%EC%B2%B4%EC%A0%9C)의 관례에 따른 줄 끝 문자를 사용한다. 예를 들어, 리눅스 프로그램은 하나의 LF([줄 바꿈](https://ko.wikipedia.org/wiki/%EC%83%88%EC%A4%84_%EB%AC%B8%EC%9E%90), 16진수 값 0A) 문자만 사용하고, 윈도우즈 프로그램은 CR ([CR](https://ko.wikipedia.org/wiki/%EC%BA%90%EB%A6%AC%EC%A7%80_%EB%A6%AC%ED%84%B4), 16진수 값 0D)과 LF를 이어서 쓴다.

**레코드 종류**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=6)]

현재 여섯가지 종류가 있다.:

|  |  |  |  |
| --- | --- | --- | --- |
| **Hex code** | **Record type** | **Description** | **Example** |
| 00 | 데이터 | 16비트 시작 주소와 데이터를 가진다. 데이터의 바이트 수가 주어진다. 오른쪽 예제는 0B 11개의 데이터 바이트를 가진다. (61, 64, 64, 72, 65, 73, 73, 20, 67, 61, 70) 이들 데이터가 0010 주소에서부터 연속으로 놓인다. | :0B0010006164647265737320676170A7 |
| 01 | 파일의 끝 | 파일의 마지막 줄에 꼭 있어야 한다. 데이터 필드는 없다. 따라서 바이트 갯수는 00이고, 주소 필드도 보통0000이다. | :00000001FF |
| 02 | 확장된 세그먼트 주소 | 이것은 데이터 필드는 16비트 세그먼트 기준 주소다. 따라서 바이트 갯수는 02다. 80x86 리얼 모드 주소와의 호환을 위한 것이다. 주소 필드는 무시된다. 보통 0000이다. 세그먼트 주소 02 레코드의 값이 16 곱해져서 이어지는 데이터 레코더의 주소에 더해져야 데이터의 물리 시작 주소가 된다. 이것은 1M까지의 주소 공간을 지원한다. | :020000021200EA |
| 03 | 시작 세그먼트 주소 | 80x86 프로세서에서, CS:IP 레지스터의 초기값을 주어진다. 주소 필드는 보통 0000이고, 바이트 갯수는 04다. 첫 두 바이트가 [CS](https://ko.wikipedia.org/w/index.php?title=Code_segment&action=edit&redlink=1) 값이고, 다음 둘이 [IP](https://ko.wikipedia.org/w/index.php?title=Instruction_pointer&action=edit&redlink=1) 값이다. | :0400000300003800C1 |
| 04 | 확장된 선형 주소 | 4기가바이트까지 가능한 32비트 주소체계를 지원한다. 주소 필드는 보통 0000으로 무시되고 바이트 갯수는 항상 02다. 빅 엔디안의 두 데이터 바이트가 이어지는 00 레코드의 절대 주소의 사위 16비트가 이다. 04종류의 레코드가 없는 00 레코드의 주소 상위 16비트는 기본값 0이다. 00 종류의 레코드의 절대 주소는 최근의 04 레코드 값이 상위 16비트가 되고, 00 레코드의 주소 값이 하위 16비트가 된다. | :02000004FFFFFC |
| 05 | 시작 선형 주소 | 주소 필드는 0000으로 사용하지 않고, 바이트 개수는 04다. 4바이트 데이터는 80386과 더 높은 CPU의 EIP 레지스터에 기록할 32비트 값을 나타낸다. | :04000005000000CD2A |

**다른 이름**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=7)]

때때로 레코드 종류의 일부만 채용한 HEX 파일 형식을 지칭하기 위해 특별한 이름이 사용된다. 예를 들어:

* **I8HEX** 파일은 00과 01 레코드만 사용한다. (16비트 주소지정)
* **I16HEX** 파일은 00에서 03까지의 레코드만 사용한다. (20비트 주소지정)
* **I32HEX** 파일은 00, 01, 04 그리고 05 레코드만 사용한다. (32비트 주소지정)

**파일 예제**[[편집](https://ko.wikipedia.org/w/index.php?title=%EC%9D%B8%ED%85%94_HEX&action=edit&section=8)]

이 예제는 4개의 데이터 레코드와 뒤따르는 끝 레코드를 보여준다.

:10010000214601360121470136007EFE09D2190140

:100110002146017E17C20001FF5F16002148011928

:10012000194E79234623965778239EDA3F01B2CAA7

:100130003F0156702B5E712B722B732146013421C7

:00000001FF