О hex-файлах

Двоичное представление команды — командное слово, или код операции (КОП) — компилятор записывает в виде числа в выходной файл с расширением hex, который затем используется программатором для записи в контроллер. Кроме hex-файлов, есть и другие форматы записи готовых программ (самый известный — бинарный), но hex-формат для микроконтроллеров самый распространенный, и мы будем рассматривать только его.

Рассматриваемый формат придуман фирмой Intel (есть и другие "гексы") и отличается тем, что содержит числа в текстовом представлении — в шестнадцатеричной записи. Поэтому в случае чего его можно даже править в обычном текстовом редакторе. Кстати, точно такой же формат применяется для записи констант в EEPROM, если это требуется.

Рассмотрим формат HEX подробнее. На рис. 5.6 представлен файл короткой программы, открытый в обычном Блокноте. На первый взгляд тут сам черт ногу сломит, но на самом деле все достаточно просто, хотя чтение затрудняется тем, что строки не поделены на отдельные байты. Разбираться будет проще, если вы скопируете этот файл под другим именем и расставите в нем пробелы после каждой пары символов. Мы же рассмотрим данный файл как есть.

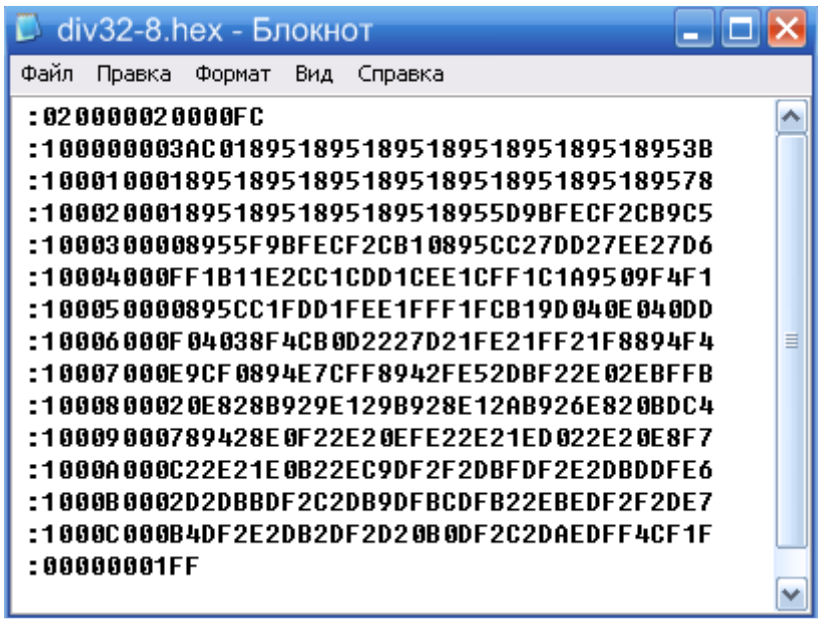

Рис. Файл формата HEX в Блокноте

Основную часть файла занимают информационные строки, содержащие непосредственно КОП. Они состоят из ряда служебных полей и собственно данных. Каждая строка начинается двоеточием и заканчивается парой символов "возврат каретки" — "перевод строки", на экране не отображаемых (ср. с протоколом MODBUS, упомянутым в главе 13). После двоеточия идет число байтов в строке — кроме первой и последней, везде стоит, как видите, число 10 (десятичное 16), т.е. в каждой строке будет ровно 16 информационных байтов (исключая служебные). Затем следуют два байта адреса памяти — куда писать (в первой строке 0000, во второй это будет, естественно, 0010, т. е. предыдущий адрес плюс 16, и т. д.). Наконец, после адреса расположен еще один служебный байт, обозначающий тип данных, который в информационных строках равен 00 (а в первой и последней — 02 и 01, о чем далее). Только после этого начинаются собственно байты данных, которые означают соответствующие КОП, записанные пословно (КОП для AVR, напоминаю, занимают в основном два байта, и память в этих МК также организована пословно), причем так, что младший байт идет первым. Таким образом, запись в первой информационной строке 3AC0 в привычном нам "арабском" порядке, когда самый старший разряд располагается слева, должна выглядеть, как C03A.

В первой строке служебный байт типа данных равен 02, и это означает, что данные в ней представляют сегмент памяти, с которого должна начинаться запись (в данном случае 0000). Заканчивается hex-файл всегда

строкой: 00000001FF — значение типа данных 01 означает конец записи, данных больше не ожидается. А что означает FF?

Самым последним байтом в каждой строке идет контрольная сумма (дополнение до 2, иначе она называется LRC — Longitudinal Redundancy Check) для всех остальных байтов строки, включая служебные. Алгоритм вычисления LRC очень простой — нужно вычесть из числа 256 значения всех байтов строки (не обращая внимание на перенос), и взять младший байт результата. Соответственно, проверка целостности строки еще проще — нужно сложить значения всех байтов (включая контрольную сумму), и младший байт результата должен равняться нулю. Так, в первой строке число информационных байтов всего два, оба равны нулю, плюс (в начале) число информационных байтов, равное 2, плюс служебный байт типа данных, равный также 2, итого контрольная сумма всегда равна 256 - 2 - 0 - 0 - 2 = 252 = $FC. В последней строке одни нули, кроме типа данных, равного 1, — соответственно контрольная сумма равна 256 - 1 = 255 = $FF.

Теперь попробуем немного расшифровать данные. Первое слово в первой информационной строке, как мы выяснили, равно $C03A. Если мы возьмем фирменное описание команд, то обнаружим, что значению старшей тетрады в КОП, равной $C (1100 в двоичной системе), соответствует команда rjmp — как мы далее увидим, практически любая программа начинается с безусловного перехода на метку reset. Теперь очевидно, что остальные биты в этом значении ($03A) представляют абсолютный адрес в программе, где в тексте ее стояла метка reset. Попробуем его найти — для этого вспомним, что адреса отсчитываются по словам, а не по байтам, т. е. число $3A (58) нужно умножить на 2 (получится 116 = $74) и искать в этой области. Разыщем строку с адресом $0070, отсчитаем три пары байтов от начала данных, и найдем там фрагмент "F894", который в нормальной записи будет выглядеть, как $94F8, а это, как легко убедиться по справочнику, есть код команды cli, запрещающей прерывания (которая в начале программы лишняя, т. к. они все равно запрещены, но, видимо, поставлена на всякий случай). Следующая команда будет начинаться с байта $E5, и первая тетрада в ней обозначает код команды ldi (1110 — проверьте!), а пятерка, очевидно, есть фрагмент адреса конца памяти (RAMEND), который в силу довольно сложного формата записи команды получается на самом деле равным $025F (см. значение младшего байта, равное $2F). Это соответствует значению RAMEND, определенному в inc-файлах для МК с 512 байтами встроенного ОЗУ ($025F = 607, т. е. всего адресов 608, из которых 96 ($5F) занимают регистры, итого получается 512 ($0200) незанятых байтовых ячеек, составляющих ОЗУ). Все, как и должно быть — если мы обратим внимание опять на первую-вторую строки с данными, то увидим повторяющийся фрагмент "1895", который, как легко догадаться из материала следующего раздела, должен быть командой reti — если проверите по справочнику, то так оно и окажется.

Как видите, разобраться довольно сложно, но при некотором навыке и наличии под рукой таблицы двоичных кодов команд вполне можно. Именно так работает программа, которая превращает код обратно в текст — дизассемблер (он входит в AVR Studio). Впрочем, в дизассемблированной программе разобраться бывает еще сложнее, чем в самом hex-файле, т. к. там, естественно, нет никаких меток и определений, все в абсолютных числах.

А зачем это может понадобиться на практике? Дело в том, что в памяти программ часто хранят константы — те, что предположительно не будут изменяться в процессе эксплуатации, например, устанавливаемые по умолчанию значения какой-то величины. Но, разумеется, по истечении некоторого времени или при переносе на другое устройство эти константы обязательно захочется изменить. И если у вас текст программы по каким-то причинам отсутствует (например, программа взята из публикации в журнале или скачана с радиолюбительского сайта), а загрузочный hex-файл имеется, то всегда можно "хакнуть" исходный код и немного подправить его под свои нужды.

Опубликовал: askarbin 6-07-2010, 15:47 в рубрике: Матчасть » Практика программирования микроконтроллеров Atmel
http://controllersystems.com/