AVR. Учебный Курс. Работа с памятью

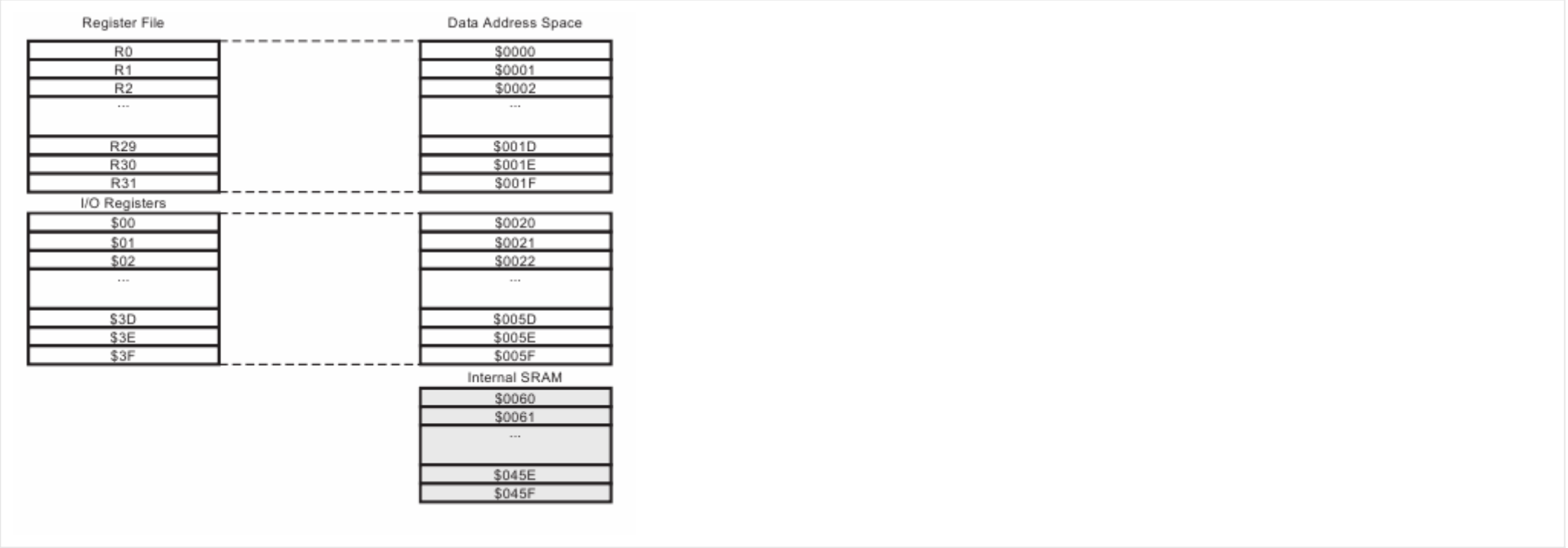
AVR. Учебный курс | 6 Июль 2008 | DI HALT | 199 Comments

Так, с работой ядра на предмет переходов и адресации разобрались. Пора обратить свой взор в другую область — память.

Ее тут два вида (EEPROM не в счет т.к. она вообщет переферия, а о ней потом):

- RAM — оперативка
- ROM — ПЗУ, она же flash, она же память программ

Так как архитектура у нас Гарвардская, то у оперативы своя адресация, а у флеша своя. В даташите можно увидеть структуру адресации ОЗУ.



Сразу обратите внимание на адреса! РОН и регистры периферии, а также ОЗУ находятся в одном адресном пространстве. Т.е. адреса с 0000 по 001F занимают наши регистры, дальше вплоть до адреса 005F идут ячейки ввода-вывода — порты. Через порты происходит конфигурирование всего, что есть на борту контроллера. И только потом, с адреса 0060 идет наше ОЗУ, которое мы можем использовать по назначению.

Причем обратите внимание, что у регистров I/O есть еще своя адресация — адресное пространство регистров ввода-вывода (от 00 до 3F), она указана на левой части рисунка. Блок IO/Register Эта адресация работает ТОЛЬКО в командах OUT и IN Из этого вытекает интересная

особенность.

К регистрам периферии можно обратиться двумя разными способами:

- Через команды IN/OUT по короткому адресу в пространстве адресов ввода-вывода
- Через группу команд LOAD/STORE по полному адресу в пространстве адресов RAM

Пример. Возьмем входной регистр асинхронного приемопередатчика UDR он имеет адрес 0x0C(0x2C) в скобках указан адрес в общем адресном пространстве.

```
1
2
3
4
5
6
7
```

```
        LDI R18,10      ; Загрузили в регистр R18 число 10. Просто так

        OUT UDR,R18     ; Вывели первым способом, компилятор сам
                        ; Подставит вместо UDR значение 0x0C

        STS 0x2C,R18    ; Вывели вторым способом. Через команду Store
                        ; Указав адрес напрямую.
```

Оба метода дают идентичные результаты. НО! Те что работают адресацией в пространстве ввода-вывода (OUT/IN) на два байта короче. Это и понятно — им не нужно хранить двубайтный адрес произвольной ячейки памяти, а короткий адрес пространства ввода—вывода влезает и в двухбайтный код команды.

Правда тут возникает еще один прикол. Дело в том, что с каждым годом появляются все новые и новые камни от AVR и мяса в них все больше и больше. А каждой шкварке нужно свои периферийные регистры ввода-вывода. И вот, дожили, в ATMega88 (что пришла на замену Mega8) периферии уже столько, что ее регистры ввода-вывода уже не умещаются в лимит адресного пространства 3F.

Опаньки, приплыли. И вот тут у тех кто пересаживается с старых камней на новые начинаются недоуменные выражения — с чего это команды OUT/IN на одних периферийных регистрах работают, а на других нет?

А все просто — разрядности не хватило.

А ядро то единое, его уже не переделать. И вот тут ATMELовцы поступили хитро — они ввели так называемые memory mapped регистры. Т.е. все те регистры, что не влезли в лимит 3F доступны теперь только одним способом — через Load/Store.

Вот такой прикол. Если открыть какой нибудь m88def.inc то там можно увидеть какие из регистров ввода-вывода «правильные» а какие memory mapped.

Будет там бодяга вот такого вида:

```
; ***** I/O REGISTER DEFINITIONS ********************************************
; NOTE:
; Definitions marked "MEMORY MAPPED"are extended I/O ports
; and cannot be used with IN/OUT instructions
.equ   UDR0   = 0xc6   ; MEMORY MAPPED
.equ   UBRR0L   = 0xc4   ; MEMORY MAPPED
.equ   UBRR0H   = 0xc5   ; MEMORY MAPPED
.equ   UCSR0C   = 0xc2   ; MEMORY MAPPED
.equ   UCSR0B   = 0xc1   ; MEMORY MAPPED
.equ   UCSR0A   = 0xc0   ; MEMORY MAPPED

бла бла бла, и еще много такого

.equ   OSCCAL  = 0x66   ; MEMORY MAPPED
.equ   PRR      = 0x64   ; MEMORY MAPPED
.equ   CLKPR   = 0x61   ; MEMORY MAPPED
```

```
.equ   WDTCSR  = 0x60   ; MEMORY MAPPED
.equ   SREG     = 0x3f  ;<------ А тут пошли обычные
.equ   SPL      = 0x3d
.equ   SPH      = 0x3e
.equ   SPMCSR  = 0x37
.equ   MCUCR   = 0x35
.equ   MCUSR   = 0x34
.equ   SMCR     = 0x33
.equ   ACSR     = 0x30
```

Вот такие пироги.

И на этой ниве понимаешь, что в сторону кроссмодельной совместимости ассемблерного кода летит летит большой мохнатый орган с целью наглухо его накрыть. Ведь одно дело подправить всякие макроопределения и дефайны, описывающие регистры, а другое сидеть и, аки Золушка, отделять правильные порты от неправильных.

Впрочем есть решение. Макроязык! Не нравится система команд? Придумай свою с блекджеком и шлюхами!
Сварганим свою собственную команду UOUT типо универсальный OUT

```
        .macro    UOUT
        .if     @0 < 0x40
        OUT     @0,@1
        .else
        STS     @0,@1
        .endif
        .endm
```

Если значение входного параметра @0 меньше 0x40 значит это «правильный» регистр. Если больше — memory_mapped.

Дальше везде, не думая, используем UOUT вместо OUT и не парим себе мозг извратами адресации. Компилятор сам подставит нужную инструкцию.

```
1           UOUT    UDR,R18
```

Аналогично и для команды IN Вообще, такими вот макросами можно ОЧЕНЬ сильно разнообразить ассемблер, превратив его в мощнейший язык программирования, рвущий как тузик тряпку всякие там Си с Паскалями.

Ну так о чем я… а о ОЗУ.

Итак, с адресацией разобрались. Адреса памяти, откуда начинаются пользовательские ячейки ОЗУ теперь ты знаешь где смотреть — в даташите, раздел Memory Map. Но там для справки, чтобы знать.

А в нашем коде оперативка начинается с директивы .DSEG Помните наш шаблончик?

```
1               .include "m16def.inc"   ; Используем ATMega16
2
3   ;= Start macro.inc ================================
4
5   ; Макросы тут
6
7   ;= End  macro.inc ==================================
```

```
8
9
10
11
12
13
14
15
16
17
18
19
```

```
; RAM =============================================
                .DSEG           ; Сегмент ОЗУ


; FLASH ===========================================
                .CSEG           ; Кодовый сегмент


; EEPROM ==========================================
                .ESEG           ; Сегмент EEPROM
```

Вот после .DSEG можно задавать наши переменные. Причем мы тут имеем просто прорву ячеек — занимай любую. Указал адрес и радуйся. Но зачем же вручную считать адреса? Пусть компилятор тут думает.

Поэтому мы возьмем и зададим меточку

```
1
2
3
```

```
                .DSEG
Variables:      .byte   3
Variavles2:     .byte   2
```

Директива .byte зарезервирует нам столько байт, сколько мы ей указали. Таким образом, на Variables у нас будет три байта, а на Variables2 два байта.

Если считаем, что у нас Atmega16, а у ней адреса RAM начинаются с 0x0060, то компилятор посчитает адреса так:

Variables = 0x0060
Variables2 = 0x0063

А в памяти это будет лежать следующим образом (приведу в виде линейного списка):

```
0x0060 ## ;Variables
0x0061 ##
0x0062 ##
0x0063 ## ;Variables2
0x0064 ##
0x0065 ## ;Тут могла бы начинаться Variables4
```

В качестве ## любой байт. По дефолту FF. Разумеется ни о какой типизации переменных, начальной инициализации, контроля за переполнениями и прочих буржуазных радостей говорить не приходится. Это Спарта! В смысле, ассемблер. Все ручками.
Если провести аналогию с Си, то это как работа с памятью через одни лишь void указатели. Сишники поймут. Поймут и ужаснутся. Т.к. мир этот жесток и коварен. Чуть просчитался с индексом — затер другие данные. И хрен ты эту ошибку поймаешь если она сразу не всплывет.

Так что внимание, внимание и еще раз внимание. Все операции с памятью прогоняем через трассировку и ничего у нас не вылезет и не переполнится.

В сегменте данных работает также директива .ORG Работает точно также — переносит адреса, в данном случае меток, от сих и до конца памяти. Одна лишь тонкость — ORG 0000 даст нам самое начало ОЗУ, а это R0 и прочие регистры. А нулевой километр ОЗУ на примере Мега16 даст ORG 0x0060. А в других контроллерах еще какое-нибудь значение. Каждый раз в даташит лазать лениво, поэтому есть такое макроопределение как SRAM_START указывающее на начало ОЗУ для конкретного МК.

Вообще полезно почитать файл m16def.inc на предмет символических имен разных констант.

Так что если хотим начало ОЗУ, скажем 100 байт оставить под какой нибудь мусорный буффер, то делаем такой прикол.

```
1
2
3
4
```

```
                .DSEG
                .ORG SRAM_START+100

Variables:      .byte 3
```

Готово, расчистили себе буфферную зону от начала до 100.

Ладно, с адресацией разобрались. Как работать с ячейками памяти? А для этих целей существует две группы команд. LOAD и STORE самая многочисленная группа команд.

Дело в том, что с ячейкой ОЗУ ничего нельзя сделать кроме как загрузить в нее байт из РОН, или выгрузить из нее байт в РОН.

Записывают в ОЗУ команды Store (ST**), а считываю команды Load (LD**).

Чтение идет в регистр R16…R31, а адрес ячейки задается либо непосредственно в команде. Вот простой пример. Есть трехбайтная переменная Variables, ее надо увеличить на 1. Т.е. сделать операцию Variables++

```
1
2
3
4
5
```

```
                .DSEG
Variables:      .byte   3
Variavles2:     .byte   1

                .CSEG
```

```
6
7       ; Переменная лежит в памяти, сначала надо ее достать.
8               LDS     R16, Variables          ; Считать первый байт Variables  в R16
9               LDS     R17, Variables+1        ; Считать второй байт Variables  в R17
10              LDS     R18, Variables+2        ; Ну и третий байт в R18
11
12      ; Теперь прибавим к ней 1, т.к. AVR не умеет складывать с константой, только
13      ; вычитать, приходиться извращаться. Впрочем, особых проблем не доставляет.
14
15              SUBI    R16,(-1)        ; вообще то SUBI это вычитание, но -(- дает +
16              SBCI    R17,(-1)        ; А тут перенос учитывается. Но об этом потом.
17              SBCI    R18,(-1)        ; Математика в ассемблере это отдельная история
18
19              STS     Variables,R16           ; Сохраняем все как было.
20              STS     Variables+1,R17
21              STS     Variables+2,R18
```

А можно применить и другой метод. Косвенную запись через индексный регистр.

```
1               .DSEG
2       Variables:      .byte   3
3       Variavles2:     .byte   1
4
5               .CSEG
6       ; Берем адрес нашей переменной
7               LDI     YL,low(Variables)
8               LDI     YH,High(Variables)
```

```
9
10      ; Переменная лежит в памяти, сначала надо ее достать.
11              LD      R16, Y+         ; Считать первый байт Variables  в R16
12              LD      R17, Y+         ; Считать второй байт Variables  в R17
13              LD      R18, Y+         ; Ну и третий байт в R18
14
15      ; Теперь прибавим к ней 1, т.к. AVR не умеет складывать с константой, только
16      ; вычитать, приходиться извращаться. Впрочем, особых проблем не доставляет.
17
18              SUBI    R16,(-1)        ; вообще то SUBI это вычитание, но -(- дает +
19              SBCI    R17,(-1)        ; А тут перенос учитывается. Но об этом потом.
20              SBCI    R18,(-1)        ; Математика в ассемблере это отдельная история
21
22              ST      -Y,R18          ; Сохраняем все как было.
23              ST      -Y,R17          ; Но в обратном порядке
24              ST      -Y,R16
```

Тут уже заюзаны операции с постинкрементом и преддекрементом. В первой сначала читаем, потом прибавляем к адресу 1. Во второй вначале вычитаем из адреса 1, а потом сохраняем.

Подобными инкрементальными командами удобно перебирать массивы в памяти или таблицы какие.
А там есть еще и косвенная относительная запись/чтение LDD/STD и еще варианты на все три вида индексов (X,Y,Z). В общем, кури даташит и систему команд.

Стек

О, стек это великая вещь. За что я его люблю, так это за то, что срыв стека превращает работоспособную программу в полную кашу. За то

что стековые операции требуют повышенного внимания, за то что если где то стек сорвет и сразу не отследишь, то фиг это потом отловишь… В общем, прелесть, а не штуковина.

Почему люблю? Ну дык, если Си это тупое ремесло, быстро и результативно, то Ассемблер это филигранное искусство. Как маньяки вроде Jim'а из бумаги и только из бумаги клепают шедевры, хотя, казалось бы, купи готовую сборную модель и клей себе в удовольствие. Так и тут — от самого процесса прет нипадецки. В том числе и от затраха с отладкой :))))

Так вот, о стеке. Что это такое? А это область памяти. Работает по принципу стопки. Т.е. какую последнюю положил, ту первой взял.

У стека есть указатель, он показывает на вершину стека. За указатель стека отвечает специальный регистр SP, а точнее это регистровая пара SPL и SPH. Но в микроконтроллерах с малым обьемом ОЗУ, например в Тини2313, есть только SPL

При старте контроллера, обычно, первым делом инициализируют стек, записывая в SP адрес его дна, откуда он будет расти. Обычно это конец ОЗУ, а растет он к началу.

Делается это таким вот образом, в самом начале программы:

```
1
2
3
4
5
```

```
        LDI R16,Low(RAMEND)
        OUT SPL,R16

        LDI R16,High(RAMEND)
        OUT SPH,R16
```

Где RAMEND это макроопределение указывающий на конец ОЗУ в текущем МК.

Все, стек готов к работе. Данные кладутся в стек командой PUSH Rn, а достаются через POP Rn.
Rn — это любой из РОН.

Еще со стеком работают команды CALL, RCALL, ICALL, RET, RETI и вызов прерывания, но об этом чуть позже.

Давай-ка поиграемся со стеком, чтобы почувствовать его работу, понять как и куда он движется.

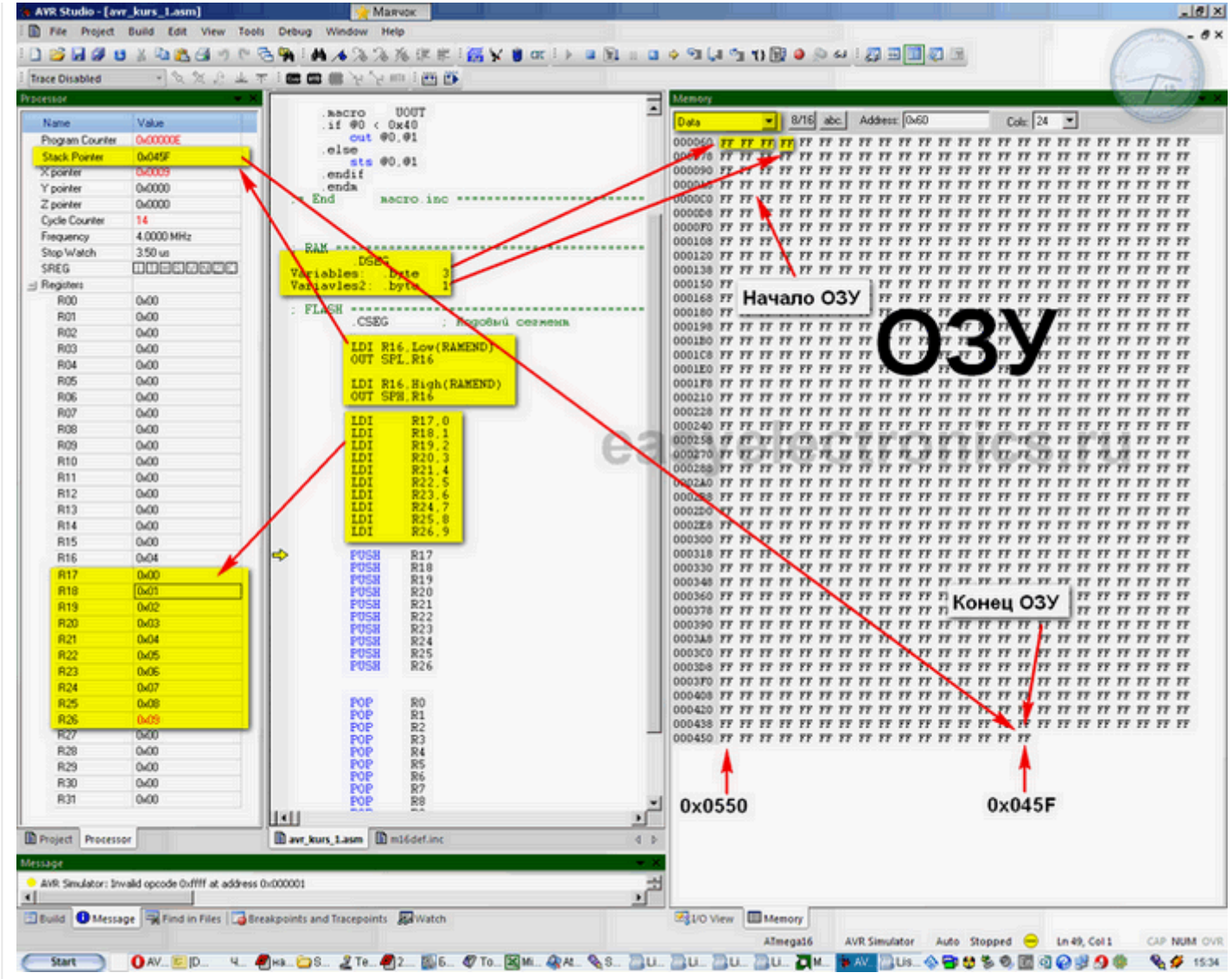
Вбей в студию такой код:

```
.CSEG                   ; Кодовый сегмент

LDI R16,Low(RAMEND)     ; Инициализация стека
OUT SPL,R16

LDI R16,High(RAMEND)
OUT SPH,R16

LDI     R17,0   ; Загрузка значений
LDI     R18,1
LDI     R19,2
LDI     R20,3
LDI     R21,4
LDI     R22,5
LDI     R23,6
LDI     R24,7
LDI     R25,8
LDI     R26,9

PUSH    R17             ; Укладываем значения в стек
PUSH    R18
PUSH    R19
PUSH    R20
```

```
24                  PUSH    R21
25                  PUSH    R22
26                  PUSH    R23
27                  PUSH    R24
28                  PUSH    R25
29                  PUSH    R26
30
31
32                  POP     R0      ; Достаем значения из стека
33                  POP     R1
34                  POP     R2
35                  POP     R3
36                  POP     R4
37                  POP     R5
38                  POP     R6
39                  POP     R7
40                  POP     R8
41                  POP     R9
```

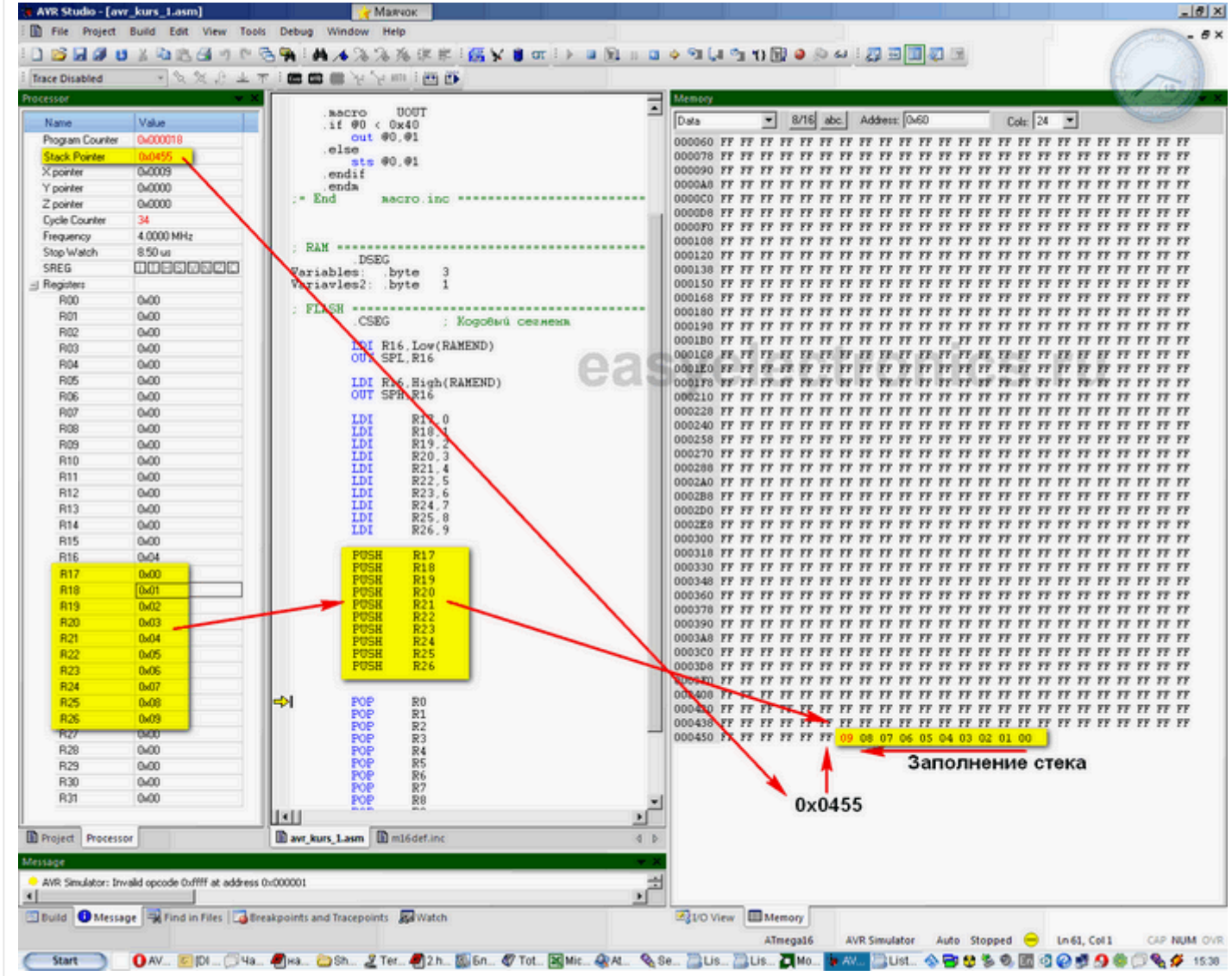
А теперь запускай студию в пошаговое выполнение и следи за тем как будет меняться SP. Stack Pointer можно поглядеть в студии там же, где и Program Counter.

Вначале мы инициализируем стек и загрузим регистры данными. В результате получится следующая картина:



увеличить

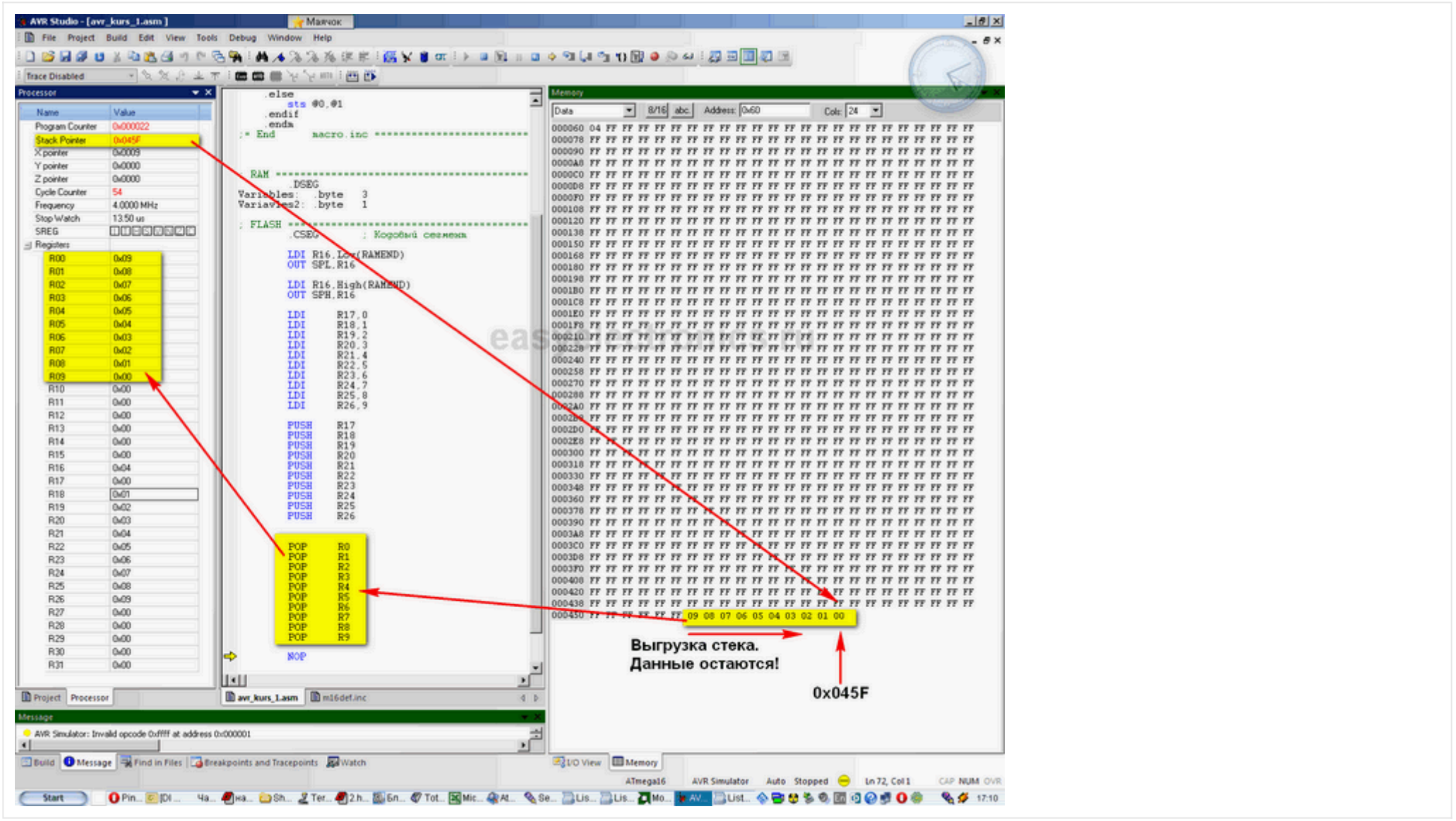
Затем начнем по одному пихать данные в стек. При этом будет видно, как данные заполняют память начиная от конца, к началу. А SP меняется в сторону уменьшения. Указывая на следующую ячейку.
После всех команд PUSH наши данные окажутся в памяти:



увеличить

Дальше, командой POP, мы достаем данные из стека. Обрати внимание на то, что нам совершенно не важно откуда мы положили данные в стек и куда мы их будем сгружать. Главное порядок укладки! Ложили мы из старших регистров, а достанем в младшие. При этом указатель

стека будет увеличиваться.



увеличить

Да, еще немаловажный момент. Данные при этом никуда из памяти не деваются, так и остаются висеть в памяти. При следующем заполнении стека их просто перепишет и все.

Как пользоваться стеком?
Ну во первых, стек используют команды вызовов и возвратов (CALL, RCALL, ICALL, RET, RETI), а еще это удобное средство по быстрому свапить или сохранять байты.

Вот, например, надо тебе обменять содержимое двух регистров R17 и R16 местами. Как сделать это без использования третьего регистра? Самое простое — через стек. Положим из одного, достанем в другой.

```
1
2
3
4
```

```
        PUSH    R16
        PUSH    R17
        POP     R16
        POP     R17
```

Или сохранить какой нибудь значение регистра, пока его используем в другом месте.

Например, я уже говорил про ограничение младших РОН — они не дают записать в себя число напрямую. Только через регистры старшей группы. Но это же неудобно!

Проблема решается с помощью макроса. Я назвал его LDIL — LDI low

```
1
2
3
4
```

```
        .MACRO LDIL
        PUSH    R17     ; Сохраним значение одного из старших регистров в стек.
        LDI     R17,@1  ; Загрузим в него наше непосредственное значение
        MOV     @0,R17  ; перебросим значение в регистр младшей группы.
```

```
5           POP     R17     ; восстановим из стека значение старшего регистра.
6           .ENDM
```

Теперь можно легко применять нашу самодельную команду.

```
1           LDIL    R0,18
```

Со временем, файл с макросами обрастает такими самодельными командами и работать становится легко и приятно.

Стековые ошибки
Стек растет навстречу данным, а теперь представьте что у нас в памяти есть переменная State и расположена она по адресу, например, 0x0450. В опасной близости от вершины стека. В переменной хранится, например, состояние конечного автомата от которого зависит дальнейшая логика работы программы. Скажем если там 3, то мы идем делать одно, если 4 то другое, если 5 то еще что-то и так до 255 состояний. И по логике работы после 3 должна идти 4ре, но никак не 10

И вот было там 3. И тут, в один ужасный момент, условия так совпали, что стек разросся и его вершина дошла до этой переменной, вписав туда значение, скажем 20, а потом борзо свалила обратно. Оставив гадость — классический пример переполнения стека. И логика программы вся нахрен порушилась из-за этого.

Либо обратный пример — стек продавился до переменных, но в этот момент переменные обновились и перезаписали стековые данные. В результате, со стека снялось что-то не то (обычно кривые адреса возврата) и программе сорвало крышу. Вот такой вариант, кстати, куда более безобидный, т.к. в этом случае косяк видно сразу и он не всплывает ВНЕЗАПНО спустя черт знает сколько времени.

Причем эта ошибка может то возникать, то исчезать. В зависимости от того как работает программа и насколько глубоко она прогружает стек. Впрочем, такое западло чаще встречается когда пишешь на Си, где не видно насколько активно идет работа со стеком. На асме все

гораздо прозрачней. И тут такое может возникнуть из-за откровенно кривого алгоритма.

На долю ассемблерщиков часто выпадают другие стековые ошибки. В первую очередь забычивость. Что то положил, а достать забыл. Если дело было в подпрограмме или в прерывании, то искажается адрес возврата (о нем чуть позже), стек срывает и прога мгновенно рушится. Либо невнимательность — сохранял данные в одном порядке, а достал в другом. Опа и содержимое регистров обменялось.

Чтобы избегать таких ошибок нужно, в первую очередь, следить за стеком, а во вторых грамотно планировать размещение переменных в памяти. Держа наиболее критичные участки и переменные (такие как состояния конечных автоматов или флаги логики программы) подальше от стековой вершины, поближе к началу памяти.

У некоторых возникнет мысль, что можно же взять и стек разместить не на самом конце ОЗУ, а где нибудь поближе, оставив за ним карман для критичных данных. На самом деле не слишком удачная мысль. Дело в том, что стек можно продавить как вниз, командой PUSH так и вверх — командами POP. Второе хоть и случается намного реже, т.к. это больше грех кривых рук, чем громоздкого алгоритма, но тоже бывает.
Но главное это то, что стек сам по себе сверхважная структура. На ней держится весь механизм подпрограмм и функций. Так что срыв стека это ЧП в любом случае.

Стековые извраты
Моя любимая тема. =)))) Несмотря на то, что стековый указатель сам вычисляется при командах PUSH и POP, никто не мешает нам выковырять его из SP, да использовать его значения для ручного вычисления адреса данных лежащих в стеке. Либо подправить стековые данные как нам угодно.
Зачем? Ну применений можно много найти, если напрячь мозг и начать думать нестандартно :))))

Плюс через стек, в классическом Си и Паскале на архитектуре x86 передаются параметры и работают локальные переменные. Т.е. перед вызовом функции вначале все переменные пихаются в стек, а потом, после вызова функции, в стек пихаются байты будущих локальных переменных.

После, используя SP как точку отсчета, мы можем обращаться с этими переменными как нам угодно. А при освобождении стека командой POP они аннигилируются, освобождая память.

В AVR все несколько не так (видимо связано с малым обьемом памяти, где в стек особо не насуешься, зато есть прорва РОН, но механизм этот тоже можно попробовать использовать.

Правда это уже напоминает нейрохирургию. Чуть ошибся и пациент труп.

Благодаря стеку и ОЗУ можно обходиться всего двумя-тремя регистрами, не особо испытывая напряг по поводу их нехватки.

Флеш память

Память EEPROM маленькая, всего считанные байты, а иногда нужно сохранить кучу данных, например, послание инопланетянам или таблицу синусов, чтобы не тратить время на ее расчет. Да мало ли что нужно заранее заныкать в памяти. Поэтому данные можно забивать в память программ, в те самые килобайты флеша, что имеет контроллер на борту.

Записать то мы запишем, а как достать? Для этого сначала надо туда что-либо положить.
Поэтому добавляй в конце программы, в пределах сегмента .CSEG метку, например, data и после нее, используя оператор .db, вписывай свои данные.

Оператор DB означает что мы на каждую константу используем по байту. Есть еще операторы задающий двубайтные константы DW (а также DD и DQ).

```
1
```

```
data:   .db     12,34,45,23
```

Теперь, метка data указывает на адрес первого байта массива, остальные байты находятся смещением, просто добавляя к адресу единичку.

Одна тонкость — дело в том, что адрес метки подставляет компилятор, а он считает его адресом перехода для программного счетчика. А он, если ты помнишь, адресует двубайтные слова — ведь длина команды у нас может быть либо 2 либо 4ре байта.

А данные у нас лежат побайтово и контроллер при обращении к ним адресует их тоже побайтово. Адрес в словах меньше в два раза чем адрес в байтах и это надо учитывать, умножая адрес на два.

Для загрузки данных из памяти программ используется команда из группы Load Program Memory

Например, LPM Rn,Z

Она заносит в регистр Rn число из ячейки на которую указывает регистровая пара Z. Напомню, что Z это два регистра, R30 (ZL) и R31 (ZH). В R30 заносится младший байт адреса, а в R31 старший.

В коде выглядит это так:

```
1         LDI     ZL,low(data*2)  ; заносим младший байт адреса, в регистровую пару Z
2         LDI     ZH,high(data*2) ; заносим старший байт адреса, в регистровую пару Z
3                                 ; умножение на два тут из-за того, что адрес указан в
4                                 ; в двубайтных словах, а нам надо в байтах.
5                                 ; Поэтому и умножаем на два
6                                 ; После загрузки адреса можно загружать число из памяти
7
8         LPM     R16, Z          ; в регистре R16 после этой команды будет число 12,
9                                 ; взятое из памяти программ.
10
11
12    ; где то в конце программы, но в сегменте .CSEG
13    data:   .db     12,34,45,23
```

199 thoughts on “AVR. Учебный Курс. Работа с памятью”

Cluster

3 Ноябрь 2008 в 4:14

Тогда уж и на других языках показать надо было. В свое время я далеко не сразу разобрался с этим на Си.

---

http://phil-sx.livejournal.com/

3 Ноябрь 2008 в 5:25

__asm__(«…»);

Шучу)))

★ DI HALT

3 Ноябрь 2008 в 12:15

Так там же указатель тока поставить вроде. Разве нет? Я Си практически не использую, поэтому точно не скажу.

Cluster

3 Ноябрь 2008 в 15:48

Нет. Если просто объявлять константу, то они забивают оперативку. Мы с народом в своё время это у меня в ЖЖ обсуждали. У меня в коде было много текстовых констант, из-за чего микроконтроллер стал зависать.

★ DI HALT

3 Ноябрь 2008 в 18:46

А чо ей надо? Модификатор Flash перед строкой?

Cluster

3 Ноябрь 2008 в 21:06

В WinAVR в pgmspace.h есть определения:
#define __ATTR_PROGMEM__ __attribute__((__progmem__))
#define PROGMEM __ATTR_PROGMEM__

Соответственно при подключении этого модуля константы объявляются так:
char Text_Loading[] PROGMEM = «Пожалуйста подождите…»;

Чтение же данных из флеша производится функциями типа pgm_read_byte, pgm_read_word и т.д. Напрямую к ним обращаться нельзя.

★ DI HALT

3 Ноябрь 2008 в 21:14

Хм, а
flash char и далее по тексту… откуда?
IAR? PIC? где то я же видел похожее. Любопытно, а почему обращение напрямую нельзя. Ведь команды даже есть такие, странно что Си компилятор их не обрабатывает по человечески. По идее то закинул в указатели адрес и дальше гонишь через LPM и все.

Cluster

3 Ноябрь 2008 в 21:41

Не понял вопроса, что значит «откуда»?
Почему по-человечески нельзя я так и не понял. Если просто определить константу, то она забивает собой оперативку, которой и так немного. В программах, где много текста, это критично.

★ DI HALT

3 Ноябрь 2008 в 22:04

Да просто где то видел такую запись, но в упор не помню в каком из диалектов Си и под какой микроконтроллер.

http://tripsin.ya.ru/?ncrnd=741

12 Ноябрь 2008 в 10:33

Так делается на CodeVisionAVR. Но почему-то от этого Proteus выдает ошибку доступа в AVR.DLL, хотя может это у меня руки кривые.

MasterAlexei

5 Ноябрь 2008 в 5:54

В AVR'ках RAM и ROM находятся в разынх адрессных пространствах, потому и нужны разные команды по чтению данных из RAM и ROM (тоже и про EEPROM).

Еще один момент — в WinAVR компиляторе есть небольшая фича, если даные лежат выше 64 кб границы (например в ATMega 128) — то они не читаются из Си обычными библиотечными функциями для ROMa, такими как pgm_read_* strcpy_P memcpy_P и т.д. Из АСМа не пробовал. Я больше на Си лабаю.

uzer_v

3 Ноябрь 2008 в 19:04

В каком номере журнала Хакер, была опубликована эта статья?

★ DI HALT

3 Ноябрь 2008 в 19:24

#117 вроде бы. Точно не помню уже. Там еще статья Криса про червь RUSTOK.C была и микроскоп на обложке.

nwanomaly__

4 Ноябрь 2008 в 2:26

есть ли прога, которая удобно делает еепром_hex?
мне как раз понадобилось 500 байтов тех синусов, а руками их долго вбивать…
конечно, сделал для себя прогу, которая сохраняет мои данные в интеловском виде, но хотелось бы иметь нормальный вариант не только под мой случай!

★ DI HALT

7 Ноябрь 2008 в 16:37

Не сталкивался.

gravizappa

7 Ноябрь 2008 в 15:43

задал тут вопрос.
вопрос снят.. =) сам дурак…

YurkaM

7 Ноябрь 2008 в 19:40

Можно ещё было упомянуть о команде с автоинкрементом Z, типа
LPM R16,Z+
Очень удобно при считывании нескольких байт подряд из массива.

Alex2008

18 Декабрь 2008 в 1:33

Так все-таки как объявит константу в памяти программ (не в ассемблере, а на си).
Использую WinAVR и AVRStudio. Желательно примерчик!!!

Cluster

18 Декабрь 2008 в 3:05

См. ниже.

Cluster

18 Декабрь 2008 в 3:01

#include «avr/pgmspace.h» // заменить кавычки на символы больше и меньше, почему-то их тут нельзя написать. Кривая защита от тегов?

// Так объявляем саму константу
char Text[] PROGMEM = «Тут текст-константа, бла, бла, бла»;

/*
Чтение производится через функции pgm_read_byte, pgm_read_word и т.д.
Например у меня функция для вывода таких констант на LCD:
*/
void lcd_print_pgm(char* text)

```
{
char c;
while (c = pgm_read_byte(text), c)
{
lcd_putchar(c);
text++;
}
}

// Вызывается просто:
lcd_print_pgm(Text);
```

Хотя есть и специальные функции для работы со строками, но у меня они что-то отказались работать…

NeMorozoff

30 Декабрь 2008 в 16:03

здравствуйте, дамы и гаспада!вопрос вам такой: почему не возможно использование первых шестнадцати регистров общего назначения с R0 по R15

★ DI HALT

30 Декабрь 2008 в 17:28

Да можно их пользовать, тока они закастрированные донельзя. С ними 2/3 команд не работает. Ни записать в них константу, ни сравнить, ни как указатель применить. Раньше я пробовал их юзать, через некоторое время натыкался на то, что нельзя с ними сделать что нибудь, приходится через задницу. Вот и забил на них. Юзаю очень редко, обычно мне второй половины регистрового файла за глаза хватает.

dima_m

10 Июнь 2010 в 8:59

Ну да они не работают с теми командами которые используют вторым аргументом константу. А с остальными командами работают. Реальная подстава. Интересно, а почему так? Это наверное связано как то с аппаратной частью МК, может быть?

nwanomaly__

30 Декабрь 2008 в 20:06

работать с ними можно, только не так непосредственно, как со старшими)
лично мне ещё одна допкоманда, чтобы поместить что-либо в «спец регистр TMP», а оттуда в нужный — не напрягает, если очень надо.

и я уж лучше так сделаю, чем потом буду разбираться, что там со стеком и всё такое…

dima_m

10 Июнь 2010 в 9:19

Ну в принципе и дом можно построить, пользуясь только топором. Только каким качество и профессионализм будет? А вот если у тебя есть и пила и гвозди и молоток? И когда каждый инструмент применяешь по назначению, тогда не дом а конфетка получится. Так же и с МК. Знать нужно все. Как работает, куда движется, что на что влияет. Тогда профессионализм и качество программ будет на должном уровне. Меньше сбоев и т.д. Поэтому сейчас сам сижу и мозги сушу над всеми примочками и возможностями МК.

nwanomaly

10 Июнь 2010 в 20:24

программирование тут тем хорошо, что можно результат получить разными путями, адекватными в разных ситуациях. если есть «кастрированные» регисты))), то почему бы ими не воспользоваться? например, когда надо сделать на коленке за пять мин закладку или флажок в чужой проге, а времени на разборку всего нет.

zloisop

2 Апрель 2009 в 23:47

блин.. я наверное совсем тупой, но я не понял откуда умножение на два.
те у нас 16 битные адреса:
$0000
$0001
…
$0010 <- пусть data указывает сюда
$0011
$0012
$0013

мы загружаем в ZL low(data*2) те. low(0010*2) те. low(0020) те 0x20
в ZH high(data*2) те 0x00
тоесть в Z 0x0020

Ничего не понял..)=
где я не прав?

---

zloisop

3 Апрель 2009 в 0:19

или у в ячейке 0000 лежит 0й и 1й байт в 0001 2й и 3й итд?
тоесть data указывает на слово?
если
$0000 — 0й, 1й
$0001 — 2й, 3й
$0002 — 4й, 5й <- data

data*2 будет равна 4м те 4й байт.
тогда в команду low/high передается первый байт слова, или как?

★ DI HALT

3 Апрель 2009 в 1:38

Просто data это всего лишь метка указывающая на адрес 0010. В области ПЗУ компилятор считает адреса словами по два байта. Т.е. ты как ни пыжься не сможешь поставить метку на нечетный адрес — ошибка будет.

А теперь, допустим, у тебя массив по
0000 Data: 00,01
0002 Data2: 02,03
0004 Data3: 04,05

С физической точки зрения, Data3 = 4, так как указывает на 4й байт массива, но если выполнишь

LDI R16,Low(Data3)

То компилятор подставит вместо Data3 не 4, а два. Т.к. он считает словами, а Data3 это второе по счету слово. Потому то и приходится умножать на два.

---

zloisop

3 Апрель 2009 в 11:06

спасибо!(= понял.

Spaun

21 Февраль 2013 в 16:58

До меня дошло только когда нашёл следующее объяснение :
«Что значит «*2″?
Дело в том, что каждая команда содержит два байта информации и занимает, таким образом, две ячейки ПЗУ. Поэтому, счетчик команд считает 2 адреса как один. Метка содержит именно данные для счетчика команд. Чтобы получить реальный адрес ПЗУ, необходимо увеличить адрес метки в 2 раза. Что мы и делаем.»

Ana-bio-z

13 Февраль 2014 в 19:51

Чтобы разобраться как это работает я, сначала в приведенном Di Haltom примере намеренно не стал умножать на двойку. В итоге у меня в индексный регистр загрузился именно тот адрес, который отображался в окне Memory Program (пословный, у меня был 0x0038), из него же в РОН загрузилось значение команды, которую я обнаружил по аресу — 0x001C. Что неудивительно, т.к. 0x0038/2 равно именно 0x001C. Затем сделал как надо (умножил на 2) и в индексный регистр пришло соответственно 0x0070 (0x0038*2). Вот теперь команда LPM загрузила в РОН то что нужно (содержимое слова в Program Memory с адресом 0x0038). Тут я подумал что через пару недель практики написания и отладки кода в AVR студии я смогу на лету умножать и делить на двойку в уме числа в шестнадцатеричном виде. Но почти сразу обнаружил в студии удобную возможность: если в окне Memory Program в контекстном меню поставить галочку на пункте Byte Address, то адресация будет отображаться уже не по порядку в словах, а через один в байтах (только четные адреса). В подобных случаях очень удобно использовать.

Ana-bio-z

13 Февраль 2014 в 20:06

Исправлюсь. Команда LPM (с постинкрементом Z) загрузила в РОН один байт слова из Program Memory с адресом 0×0038. Второй байт закинул следующей командой LPM в следующий регистр. И т.д.

---

paschen

22 Май 2009 в 20:54

а можно ли в FLASH ROM записать данные полученные в ходе выполнения программы (скажем 60кбайт) по типу EEPROMA а потом считать?…

Нужно множество значений с одного порта сохранить где то а потом считать, можно ли все эти значения сохранять в FLASH ROM. (EEPROMA не хватает просто).

★ DI HALT

22 Май 2009 в 21:59

Есть вариант писать во флеш во время выполнения программы, но писать можно только блоками по несколько килобайт, сразу весь блок. Вначале в буффер специальный, потом в блок флеша. Почитай про самопрограммирование авр. Не хватает епрома, навесь внешнюю епромину. Или возьми другую мегу.

---

chirik

24 Январь 2010 в 4:08

А как задать массив не в одну строку а по строкам ? 10*10 ?
чтоб 100 элементов в длину не записывать?

★ DI HALT

24 Январь 2010 в 10:52

massiv: db 1,1,1,1
db 1,1,1,1
db 1,1,1,1

и так далее. Главное чтобы в строке было четное число элементов.

chirik

24 Январь 2010 в 19:28

я делаю так, и где-то в середине 2ой строки при симуляции в AVRstudio возникает ошибка переполнение стека. Пробовал если разместить все в одну строку <= 18элементов ошибки нет и программа идет дальше.

mas:
.db 8,17,26,35,44,53,61,70,78,87
.db 95,103,111,119,127,135,142,149,156,163
.db 170,177,183,189,195,200,206,211,216,220

★ DI HALT

24 Январь 2010 в 19:31

А строки четные? А то если строки не четные то возникает такой косяк:

Адресация флеша в словах. Т.е. минимальное деление которое понимает компилятор (но не процессор, ему пофиг) это слово. Т.е. если у тебя будет три байта в строку, то он адресует их как четыре байта, дополнив строку нулем. В результате твоя обработка поплывет черти куда.

А если у тебя четыре байта в строке (или любое четное число) то все будет ок и байты твои в памяти лягут как надо, без нулей.

chirik

25 Январь 2010 в 3:53

Не много разобрался, поместил таблицу в конец программы и ошибка пропала)

Sheld

11 Февраль 2010 в 10:02

Долго рвал себе мозг если мы командой LPM загружаем в РОН, из памяти программы, получается что в РОН должно лечь СЛОВО, но это бред, вычитал свое спасение:
«При этом страшие 15 разрядов сожедримого регистра (им. Z) будут определять адрес слова (0..32К),а младший байт будет определять, какой из байтов будет прочитан «0»-младший байт, «1»- старший байт.»
А.В. Евстифеев

Конечно работало и без этого понимания, но все равно было не очень понятно (

★ DI HALT

11 Февраль 2010 в 23:02

А зачем рвать себе мозг если проц у нас 8ми разрядный. Адресация же по памяти ROM побайтная!!! И лишь в компиляторе указывается в словах.

Anton

4 Ноябрь 2011 в 16:29

Зачем вы всех путаете со своим компилятором??? Организация памяти AVR выполнена по схеме Гарвардского типа, в которой разделены не только адресные пространства памяти программ и памяти данных, но также и шины доступа к ним. Так вот шина FLASH памяти 16 разрядная! из FLASH в регистр команд(далее это все идет в дешифратор) считывается сразу все слово а не по байту за 2 цикла. И програмный счетчик тоже указывает на слова а не на байты. И компилятор тут не причем. Так работает сам контроллер

★ DI HALT

4 Ноябрь 2011 в 17:30

Дело в записи. Адресация флеша «команд» идет в словах (команды перехода JMP оперируют адресами указанными в словах). Адресация флеша «данных» идет в байтах (команды LPM). Вот и получается прикол, что компилятор метки указывающие на

данные понимает также как и метки указывающие на код (собственно метка она и есть метка), а если нам надо считать данные из флеша, то приходится пересчитывать адрес из слов в байты. А поскольку всей этой адресной арифметикой занимается компилятор, сам вычисляя по меткам адреса и смещения, то получается, что он очень даже причем.

---

http://obstinatus.myopenid.com/

3 Март 2010 в 17:12

Еще один нубский обпрос про массивы. Допустим, есть у нас массив ‘array’, количество элементов которого меньше 256, и нам надо считать из него n-й элемент. Я смотрел несколько чужих кодов и практически во всех происходит следующее:

```
clr r16
; Сначала загружаем в регистровую пару адрес нашего массива:
ldi ZL,Low(array*2)
ldi ZH,High(array*2)
; Теперь нужно добавить смещение до нашего элемента
ldi r16,n
; где ‘n’ — номер элемента, не забываем что отчет начинается с нуля
add ZL,r16
clr r17
adc ZH,r17
; теперь нужно дунуть иначе чуда не будет ;)
lpm
mov r16,r0
```

Всё, r16 хранит нужное значение.

Так вот, я не понимаю как тут работает строка adc ZH,r17! Мне не понятно почему тут добавляется смещение до элемента в младший ZL, совершенно не заботясь о carry-флаге, а к старшему ZH добавляем ноль, но с помощью adc, и это автоматически срабатывает — если значение ZL превысило 0xFF, то после adc ZH,r17 значение ZH автоматически увеличивается на единицу.

Т.е. если бы я писал код сам, то у меня бы получилось нечто вроде этого:

```
; заносим в Z адрес
```

```
ldi ZL,Low(array*2)
ldi ZH,High(array*2)
; загрузить в РОН номер элемента
ldi r16,n
; заносим смещение
adc ZL,r16
; если переполнения ZL не было идем дальше
brcc label
; переполнение было, значит инкрементировать ZH
inc ZH
label:
lpm
mov r16,r0
```

Подозреваю что это всё как-то связано со сложением двух двухбайтных чисел:

```
; Add R1:R0 to R3:R2
add r2,r0 ; Add low byte
adc r3,r1 ; Add with carry high byte
```

Но разобраться как adc работает в случае с массивом не могу.

★ DI HALT

3 Март 2010 в 18:02

Вот адс и дает заботу о флаге переноса.
Такой метод если у нас смещение в пределах 256.

Т.е. мы вначале прибавляем смещение. Флаг С при этом может быть установлен где то ваще в левом месте по итогам другой какой то операции. Чтобы нам не парить на этот счет мозг, и не очищать его предварительно (чего ты не сделал, кстати) мы сложение младшего адреса делаем без переноса. Но флаг переноса выставится если будет переполнение. И чтобы учесть его мы просто

прибавляем к старшему байту нуль с учетом переноса. Т.е. если был флаг С то это и даст тебе тот самый инкремент старшего байта адреса.

★ DI HALT

3 Март 2010 в 18:05

adc работает просто.

adc r3,r1
эквивалентно
R3 = R1+R3+C

Evg333

3 Июнь 2010 в 14:28

а как работает
SUBI R16,(-1)
SBCI R17,(-1)
SBCI R18,(-1)
Почему к регистру R17 ничего не прибавляется? по идее к каждому байту должно прибавиться 1, а получается как положено, только к 1му.

★ DI HALT

10 Июнь 2010 в 9:14

-1 это 0xFF (1111 1111)

Теперь, допустим, в R16 = 00000001

0000 0001
—
1111 1111

В беззнаковом эквиваленте 0x01 намного меньше чем 0xFF. Поэтому будет заем из старшего разряда (девятого) — и поднимется флаг C. Т.е. будет сделано действие вида:
├— это будущий заем
V
1 0000 0001 (R16)
— 1111 1111 (-1)
C 0000 0010 (Результат)

Т.е. в регистре будет число 2 (минус на минус дало +) и флаг C.

Следующее действие это тоже вычитание, но с учетом флага C и работает оно так:
R17-(-1)-C

-(-1) по идее должно дать +1, но т.к. у нас образовался флаг C, то он даст еще одну -1 и результат операции будет нулевой. Т.е. к R17 ничего добавлено не будет.

А когда будет добавлено? Тогда, когда в результате первой операции не будет заема. Т.е. когда значение в R16 достигнет 255
├— Заем, но он не пригодится
V
1 1111 1111 (R16)
— 1111 1111 (-1)
0000 0000 и флага C нет, т.к. числа равные были. Нет нужды занимать.

Но что произошло? Младший разряд нашего счетчика R16 обнулился как бы по переполнению. Флаг C не возник и в результате следующая команда отработала на плюс.

Понятно?

Evg333

11 Июнь 2010 в 0:29

Почти) Извините за тупость. А после SBCI R17,(-1) флаг C не сбрасывается видимо, поэтому и с R18 также. А при прибавлении 20 тогда неясно как (Спасибо за разъяснения, откопал книжку Ассемблер Питера Нортона, буду там читать, может дойдет)

★ DI HALT

11 Июнь 2010 в 0:46

Да, если там будет заем, то пойдет по цепочке до победного конца.

При прибавлении 20 то же самое будет. Покажу на примере не 20, а 15 чтобы проще было рисовать. Прибавим 15 к 1 через вычитание

-15 получается так. Нам надо из 255 отнять 15 и прибавить 1 (покури двоичное представление чисел со знаком).
+1111 1111
-0000 1111
+0000 0001
=1111 0001 = -15

Теперь вычитаем с заемом
1 0000 0001
— 1111 0001
C 0001 0000 = 16 + C
————-

Arseniy Muradov

19 Ноябрь 2011 в 0:55

Я не понял
1111 0001 — это «-15» в дополнительном коде
1 — это 0000 0001

теперь мы вычитаем
0000 0001
1111 0001
—
0001 0000 и почему у нас вылез знак C если мы в доп. считали? Почему не V?

Чего я не правильно понял?((

★ DI HALT

19 Ноябрь 2011 в 1:07

Контроллер не знает ни о допкоде ни о каких либо еще представлениях. Это уже человеческая абстракция. число 11110001 больше чем 00000001, а при вычитании из меньшего большего всегда вылазит заем.

Arseniy Muradov

19 Ноябрь 2011 в 1:11

Тогда как вообще МК может выставить флаг V?
Это ведь флаг переполнение доп. кода? Но МК-то о нем не знает…

★ DI HALT

19 Ноябрь 2011 в 1:28

Он работает как переполенине на знаковых данных.

К примеру:
ldi r20,100
ldi r21,100

add r20,r21

даст V т.к. число 200 не влезает в знаковый формат, который от -127/+127

а вот:

ldi r20,64
ldi r21,63

add r20,r21

флаг V не даст. Т.к. результат меньше 128.

Конечно контроллер понятия не имеет каого типа данные у тебя лежат в регистрах. Может это и беззнаковые вовсе. Но флаг V ставит все равно, на всякий случай. Вдруг ты считаешь знаковые числа и тебе эта информация нужна будет.

Arseniy Muradov

19 Ноябрь 2011 в 1:45

Ааа, теперь понял. Спасибо большое.

Arseniy Muradov

21 Ноябрь 2011 в 0:09

Хотя снова, вот:
ldi r20,140
subi r20, 50
выдает флаги S и V
Почему?
140
-50
=90
в двоичном:
1000 1100
-0011 0010
=0101 1010

теперь подробно по флагам:
«S — флаг знака. 1 — значит минус. При вычислении чисел со знаком он возникает если после арифметической операции возник отрицательный результат.»(c)
У нас же он положительный, не? 0.101 1010 = +101 1010

«V — Флаг переполнения дополнительного кода. Это если мы считаем число в дополнительном коде со знаком и оно вылезло за пределы регистра»(c)
и
«флаг V не даст. Т.к. результат меньше 128″(c) чуть выше

0101 1010 -> +90 в десятичной. А у нас же от -127 до +127. то есть все ок

В чем опять вся пичаль? :(

Arseniy Muradov

21 Ноябрь 2011 в 0:15

Хотя снова, вот:
ldi r20,140
subi r20, 50
выдает флаги S и V
Почему?
140
-50
=90
в двоичном:
1000 1100
-0011 0010
=0101 1010

теперь подробно по флагам:
«S — флаг знака. 1 — значит минус. При вычислении чисел со знаком он возникает если после арифметической операции возник отрицательный результат.»(c)
У нас же он положительный, не? 0.101 1010 = +101 1010

«V — Флаг переполнения дополнительного кода. Это если мы считаем число в дополнительном коде со знаком и оно вылезло за пределы регистра»(c)
и
«флаг V не даст. Т.к. результат меньше 128″(c) чуть выше

0101 1010 -> +90 в десятичной. А у нас же от -127 до +127. то есть все ок

Али это все выполняется СЛОЖЕНИЕМ положительного доп. с отрицательным?
1000 1100

+1100 1110
=1 0101 1010
Тем не менее 346 — это за пределами [-127;+127], так что V ставиться.
Но тут переполнение байта, разве не должен C выпасть?

Как меня запутали эти флаги-то)))

В чем опять вся пичаль? :)

Arseniy Muradov
21 Ноябрь 2011 в 0:15
Хотя снова, вот:
ldi r20,140
subi r20, 50
выдает флаги S и V
Почему?
140
-50
=90
в двоичном:
1000 1100
-0011 0010
=0101 1010

теперь подробно по флагам:
«S — флаг знака. 1 — значит минус. При вычислении чисел со знаком он возникает если после арифметической операции возник отрицательный результат.»(с)
У нас же он положительный, не? 0.101 1010 = +101 1010

«V — Флаг переполнения дополнительного кода. Это если мы считаем число в дополнительном коде со знаком и оно вылезло за пределы регистра»(с)
и
«флаг V не даст. Т.к. результат меньше 128″(с) чуть выше

0101 1010 -> +90 в десятичной. А у нас же от -127 до +127. то есть все ок

Али это все выполняется СЛОЖЕНИЕМ положительного с отрицательным в доп.?
1000 1100
+1100 1110
=1 0101 1010
Тем не менее 346 — это за пределами [-127;+127], так что V ставиться.
Но тут переполнение байта, разве не должен C выпасть?

Как меня запутали эти флаги-то)))

В чем опять вся пичаль? :)

Arseniy Muradov

21 Ноябрь 2011 в 0:16

Хотя снова, вот:
ldi r20,140
subi r20, 50
выдает флаги S и V
Почему?
140
-50
=90

в двоичном:
1000 1100
-0011 0010
=0101 1010

теперь подробно по флагам:
«S — флаг знака. 1 — значит минус. При вычислении чисел со знаком он возникает если после арифметической операции возник отрицательный результат.»(с)
У нас же он положительный, не? 0.101 1010 = +101 1010

«V — Флаг переполнения дополнительного кода. Это если мы считаем число в дополнительном коде со знаком и оно вылезло за пределы регистра»(с)
и
«флаг V не даст. Т.к. результат меньше 128″(с) чуть выше

0101 1010 -> +90 в десятичной. А у нас же от -127 до +127. то есть все ок

Али это все выполняется СЛОЖЕНИЕМ положительного с отрицательным в доп.?
1000 1100
+1100 1110
=1 0101 1010
346 — это за пределами [-127;+127], так что V ставиться.
Но тут переполнение байта, разве не должен C выпасть?

Как меня запутали эти флаги-то)))

В чем опять вся пичаль? :)

Arseniy Muradov

21 Ноябрь 2011 в 0:17

ЧЕРТ!)) Удали пожалуйста первые 3))))

А то сам не заметил, думал я редактирую коммент, а оказалось что новый пишу!)

★ DI HALT

11 Июнь 2010 в 0:48

Флаги меняются исходя из результата предыдущей арифметической операций. Если был заем/переполнение то будет C, если не было то не будет. То же самое касается и флага нуля, например. Образовался ноль? поднялся флаг Z

Temp

5 Апрель 2010 в 10:45

«Делается это таким вот образом, в самом начале программы:

LDI R16,Low(RAMEND)

OUT SPL,R16

LDI R17,High(RAMEND)

OUT SPH,R16»

У Вас тут ошибочка в тексте :)) А вообще курс мне нравится. Если б преподы так рассказывали, то я б уже профи был :)))

★ DI HALT

5 Апрель 2010 в 10:49

Благодарю, щас пофиксю

auara

25 Апрель 2010 в 14:26

1) закралась очепятка VariaBles2 VariaVles2
Лучше оставить VariaVles2, т.к. на рисунке «ak9m.gif» оно уже отмечено.
2) >>»Таким образом, на Variables у нас будет три байта, а на Variables2 два байта.»
Там указан 1 байт. И чуть ниже поправить (перенеся)
5 0x0064 ## ;Тут могла бы начинаться Variables3

test

---

dima_m

24 Май 2010 в 17:24

Слушай DI HALT в тексте выше есть директивы .IF, .ELSE. они работают так же как связка команд в Си
IF … THEN …;
ELSIF … THEN …;
ELSE …;
END_IF;

или как то по другому? Если по другому то в чем отличия? Есть ли на сайте здесь подробное их обьяснение? Даже здесь про них умалчивают почему то- http://www.mymcu.ru/Articles/Atmel11.htm#Assembler%20directives

---

★ DI HALT

24 Май 2010 в 17:30

Это директивы условной компиляции. Т.е. если условие выполняется, то в выходной код подставляется один кусок кода, если не выполняется, то другой. Т.е. пользоватся ими для обработки условий в программе нельзя. А вот для построения макросов под конкретные условия самое то.

У меня про них должно быть описано в разделе Макроассемблер.

dima_m

24 Май 2010 в 19:02

Там только о некоторых упоминается кратко в комментариях с примерами. В тексте к сожалению нет ничего. Может добавишь? А пока хоть то что в комментариях есть, можно рассмотреть. Часок другой попарюсь, думаю разберусь.

Evg333

3 Июнь 2010 в 11:03

Странно у меня после выполнения команды STS 0x2C,R18 регистр UDR сбрасывается в 0, хотя должно быть тоже что и после OUT UDR,R18 (0xA), проверил все не один раз, вроде в 3х командах не ошибся.

Evg333

3 Июнь 2010 в 11:42

Вопрос снят, если дальше нет кода такой эффект, если сделать RJMP или хотябы NOP все работает)

Evg333

3 Июнь 2010 в 14:57

а в примере с индексным регистром последний инкремент (Y+) и первый декремент (Y-) получаются лишние и без них все работает. Или обязательно надо чтобы указатель выходил за пределы переменной?

Buxxter

5 Июнь 2010 в 22:54

В макросе «с блэкджеком и шлюхами» ругается на строку
.if @0 < 0x40
avr_kurs_1.asm(10): error: syntax error, unexpected REGISTER

В чем трабла?

Ну и сразу попутно вопрос: как сделать, чтобы в редакторе показывались номера строк?

Buxxter

6 Июнь 2010 в 3:08

Хммм… чйорд, в нулевой аргумент передавал РОН, а не регистр ввода/вывода.
Вопрос снят. Прошу прощения.

Второй вопрос еще на повестке

★ DI HALT

6 Июнь 2010 в 3:15

2. Никак.

Sergey Shyian

13 Июнь 2010 в 16:29

LPM R16, Z ; в регистре R16 после этой команды будет число 12,
; взятое из памяти программ.

; где то в конце программы, но в сегменте .CSEG
data: .db 12,34,45,23

А как достать второе число или третее?

Sergey Shyian

13 Июнь 2010 в 16:32

Обьясните пож. что означают @0, @1

nwanomaly

13 Июнь 2010 в 18:44

нулевой и первый передаваемые параметры макросу

Fossa

14 Июнь 2010 в 19:42

А возможны ли макросы с переменным числом параметров? А-ля перезагруженные функции в Си?

Ghecc

30 Июнь 2010 в 20:31

Здравствуйте, кто-нибудь разбирался как подключить внешнее ОЗУ к МК (например, к atmega128). Столкнулась с тем, что из внешнего ОЗУ могу считать данные, а записать не могу. И, вообще, там постоянно какой-то хлам записывается на все 64 Кбайта.Пробовала прогу из даташита — не пишет и все!хоть тресни моя голова.

MasterAlexei

1 Июль 2010 в 0:44

Ну к меге 128, в принципе, память внешняя подключается довольно просто. Так, как в даташитке описано.
Вот тут вот у меня пример есть.

http://www.fun-electronic.net/lang/ru/2009/02/23/mazda-mp3-player-continue/
3-я картинка сверху в статье.
Он малость сложный, но из него можно выдрать все что надо для работы с памятью, и не обязательно делать в CPLD/FPGA, но можно на рассыпухе сделать.

MasterAlexei

1 Июль 2010 в 0:58

Правда именно по той ссылке там на картинке ошибочная версия (в статье про это описано — там 0й и 31й банки — одни и теже). В третьем продолжении лежит правильная прошивка CPLDы.
Но смысл, я думаю, должен быть понятен. Там основная проблема — разделить на порту A данные и нижнюю часть адреса по стробу сигнала ALE. Т.е. когда ALE в единице — на PORTA у нас нижняя часть адреса. И ее надо как то залочить. Для этого используется элемент LD8, который можно заменить в рассыпухе микросхемой 74HC646.
Ну а потом надо как то декодировать адрес, чтобы определить, когда селектировать память. В прошивке CPLD это определяется по двум адресным линиям ADRH6 и ADRH7. ТАким образом при попытке прочтения/записи адресов с 0xC000 по 0xCFFF будет происходить запись во внешнюю память, так как CS линия (на картинке SRAMCS_N) будет активироваться. (это если я правильно все вспомнил. Дело было то почти полтора года назад)

Br.Misha

26 Июль 2010 в 3:41

а как мне записать во флеш байт по указаном адресу?

★ DI HALT

26 Июль 2010 в 10:56

Именно во флеш? Только через команду SPM но там не так все просто. Она пишет ТОЛЬКО страницами по сколько то там килобайт. И в 99% юзается для бутлоадеров. Для записи во флеш своих каких то данных это не самый удачный способ.

А если ты хочешь записать во флеш данные из программы, в момент ее составления, то просто пишешь

.ORG нужный адрес
byte: .db 0x00

Ну или лбые другие данные

Br.Misha

26 Июль 2010 в 13:29

Тоесть, если я, к примеру, по адресу 0x0140 хочу записать 0xAF, то так:
.ORG 0x0140
byte: .db 0xAF

★ DI HALT

27 Июль 2010 в 10:17

Именно. Только учти, что адрес будет в словах.

Br.Misha

26 Июль 2010 в 13:31

DI HALT, да, записывать нужно просто константу во время ее составления.

---

darksab0r

27 Июль 2010 в 8:52

«Т.е. адреса с 0000 по 001F занимают наши регистры, дальше вплоть до адреса 003F идут ячейки ввода-вывода — порты.»
Вроде как, согласно картинке, порты идут до 003F, а до 3F идут в своем адресном пространстве.

darksab0r

27 Июль 2010 в 8:53

Тьфу ты, написал…
Порты идут до 005F, естественно.

★ DI HALT

27 Июль 2010 в 10:16

Спасибо, подправил.

Alice

7 Октябрь 2010 в 18:32

Как из флеш-памяти вытаскивать по 3 бита?

★ DI HALT

7 Октябрь 2010 в 19:13

Именно бита? Побитно никак. Только доставать байт, а потом масками убирать лишние биты.

moroz

11 Октябрь 2010 в 16:43

Доброго всем времени суток. У новичка возникли вопросы:
1. Обязательно ли данные в кодовом сегменте размещать после основной программы (а не где-то в теле)- это вопрос читаемости кода, либо чревато сбоем программы.
2. При запуске в студии нижеследующего примера шаг через строку с data: обновляет содержимое R19, причем при циклическом выполнении данные каждый раз иные — как счетчик какой-то. Подскажите пожалуйста — что это такое.

```
.include «m16def.inc»

.ORG 0x10
LDI R16,Low(RAMEND) ; Инициализация стека
OUT SPL,R16
LDI R16,High(RAMEND)
OUT SPH,R16
LDI ZL,low(data*2+3) ; Смещением достаю 4й байт данных
LDI ZH,high(data*2+3)
LPM R16, Z
data: .db 0x11,0x22,0x33, 0x44 ; Чтоб в памяти хорошо заметны были
```

3. Извините за глупый вопрос — а как вообще использовать SRAM, если она энергозависима, доступ — только обмен с рон… Забивать данными, полученными в результате работы программы, а значит функции в табличном виде в нее не занесешь. Остается — только большой стек. Или я совсем туплю.
Заранее — очень спасибо.

★ DI HALT

11 Октябрь 2010 в 17:11

1
данные в коде выполняются как инструкции. Ибо процессору пофигу, он просто берет по счетчику PC следующий байт из памяти.

2
Т.е. твоя строка
data: .db 0×11,0×22,0×33, 0×44 с точки зрения выполняющего код процессора выглядит как набор команд с кодами

0x1122
0x3344

Можешь сам в даташите посмотреть что это за команды и почему это они вдруг меняют R19:)

Поэтому данные и размещают в конце кода, за командой RJMP Main. Или там где процессор не сможет их выполнить. Т.е. их можно размещать и посреди кода, но тогда их придется перепрыгивать с помощью RJMP дабы проц их не выполнил. Что дает лишние команды. Оно нам надо? Об этом, кстати и в статье было написано. Читать надо внимательней.

3. Так и использовать. Как хранилище текущих данных. Как переменные программы. В большой программе тебе 31 регистра явно не хватит. Ну и стек тоже.

Табличные данные вносятся либо во флеш либо в еепром.

---

moroz

12 Октябрь 2010 в 12:28

Спасибо за быстрый ответ. Читаю и изучаю крайне внимательно. Но только в последнем листинге комент: «; где то в конце программы, но в сегменте .CSEG». И то что для гуру само-собой, для очень новых новичков вовсе не очевидно. Поэтому не флуда ради, но подобных мне чайников для — заостряю: через область данных в кодовом сегменте надо совершать «RJMP метка», а уж куда метка ведет — по обстоятельствам (если сам правильно понял).
Кстати, так же чайнику не очевидно, что макрос транслируется не полностью, но только частью, подходящей по условию.
И еще вопрос: файл с макросами может иметь любое расширение, лишь бы plane text внутри? И как подключать тот или иной макрос — компилятор сам подхватит, найдя соответствующий заголовок в include — файле?

★ DI HALT

12 Октябрь 2010 в 19:45

CSEG это все что будет после компиляции определено и адресовано как флеш.

DSEG — адресуется как оперативка

ESEG — EEPROM

Если у тебя идут в коде данные, то выглядит это, например так:

```
INC R16
MOV R17,R18
SEI
CPI R31,12
RJMP M1

data: db «blablabla»
M1: MOV… .
```

и так далее. Т.е. тебе надо чтобы контроллер перепрыгнул код и не попытался выполнить данные, иначе это плохо кончится. При этом возникает лишняя команда. Которой не будет если разместить данные где нибудь в конце, где контроллер не сможет выполнить код. Скажем после RJMP который зациклит программу.

Расширение любое. Лишь бы текст. Инклюду то все равно что включать. Он включает то что ты ему подсунешь.

Макрос подключается через инклюд, например, или вписывается в текст программы где нибудь вначале. До того как пойдет .CSEG И он должен быть обьявлен ДО того как где нибудь встретится в коде, иначе компилятор не поймет. Т.е. вписать портянку макросов в конце проги нельзя.

---

dundich

30 Октябрь 2010 в 0:45

Здравствуйте, а что значит «значение входного параметра @0»??? то есть это адрес порта??? и что означает «собачка» в асме?

★ DI HALT

30 Октябрь 2010 в 1:35

Это параметр макроассемблера. Т.е потом ,в коде, вместо @0 подставится первый операнд макрокоманды. А он может быть чем угодно уже.

---

dundich

30 Октябрь 2010 в 11:44

ааааа….понял. тепеть ясно, как работает этот макрос. спасибо!!!

---

dima_m

4 Ноябрь 2010 в 22:35

Интересная фишка. Вбил я для теста во флеш .db 1,2,3,4 Скомпилировал и запустил прогу.
Открыл DISASSEMBLER в студии и там такое дело:

.db 1,2,3,4
+00000020: 0201
+00000021: 0403

мои цифры 1,2,3,4 расположены слева направо во флеш памяти. Далее открыл вкладку Memory Program. Там уже эти цифры по адресу 20 и 21 расположены справо налево:
000020 0102
000021 0304

Почему все наоборот? Я понял что в дисассемблере все так как есть внутри МК. Ведь байты то во флеше справа налево увеличиваются по адресам?

А ты DI вообще пользуешся DISASSEMBLER-ом? Кажется полезная штука посмотреть что где легло. Особенно когда какие нибудь индексные переходы надо лепить.

★ DI HALT

4 Ноябрь 2010 в 22:42

Там правило — младший байт по младшему адресу. А зеркалит их от того, что поди представление идет не в байтах ,а в словах. Вот и получается, чтобы представить тебе слово, как оно было бы в человеческом виде приходится зеркалить.

Т.е. слово 0x1234 будет в памяти как 34,12

А твоя запись байтовая и в памяти лежит как 1,2,3,4 но вот 1,2 и 3,4 образуют слова. И хекс редактор их «для удобства» переворачивает т.к. думает, что орудует словами, а не байтами. Полазь там по настройкам представления, где то вроде бы было показывать в байтах, а не в словах. На самом деле они лежат верно. 1 по младшему адресу 4 по самому старшему.

Дизассемблер пользую постоянно.

C_X_Z

5 Ноябрь 2010 в 22:56

Здравствуйте у меня возникает вот такая ситуация при опробовании примера работы со стеком.

Запускаю симуляцию (Ctrl+F7)
Выдает следующее:
D:\avr\work\new1\new1.asm(5): error: Undefined symbol: RAMEND
D:\avr\work\new1\new1.asm(6): error: Undefined symbol: SPL
D:\avr\work\new1\new1.asm(8): error: Undefined symbol: RAMEND
D:\avr\work\new1\new1.asm(9): error: Undefined symbol: SPH

Что не так? Эти команды надо каким то макросами прописывать? Потому как в предыдущих примерах в строке
LDI ZL,low(data*2)

При запуске симуляции тоже ругается на «Undefined symbol: ZL»
Если поменять просто на «R30» то все работает

★ DI HALT

5 Ноябрь 2010 в 23:28

Не подключен файл определений ***def.inc Либо подключен с ошибкой.

---

cruorvult

7 Ноябрь 2010 в 16:29

Здравствуйте!
Пытаюсь понять как работает стек. Контроллер tiny2313. Stack Pointer = 0xDF, в стек записываю ,допустим, 3 числа. В окне Memory нахожу адрес 0xDF но там пустота(что не так?

★ DI HALT

7 Ноябрь 2010 в 16:47

Каким образом ты заносишь в стек числа. А во вторых, в какой именно памяти ты смотрел? У авр их три вида каждая со своим адресным пространством.

---

cruorvult

7 Ноябрь 2010 в 17:04

В окне Memory смотрел вкладку Data(как на скрине)
Так записываю:
LDI R16,Low(RAMEND)
OUT SPL,R16

```
ldi r17,10
ldi r18,20
ldi r19,30

push r17
push r18
push r19
```

★ DI HALT

7 Ноябрь 2010 в 22:59

Должно все нормально работать. Ищи в конце памяти. Вообще открой память так, чтобы видеть все сразу и погрузи в стек что нибудь. Там где будет красные символы — там и идет запись.

cruorvult

8 Ноябрь 2010 в 21:45

Спасибо! Почему-то не было видно конец памяти. Покрутил окошко и всё стало нормально=)

SiO2

26 Ноябрь 2010 в 23:14

А можно быстро обнулить память в симуляторе? Немного мешает, когда вместо девственных FF в памяти висят куски старого кода.

★ DI HALT

26 Ноябрь 2010 в 23:39

Хы, а ты наивно думаешь, что при старте МК в памяти девственно чистые FF? Да там порой такая каша бывает уххх… Делай инициализацию памяти в любой программе.

Vovan91

14 Март 2011 в 16:58

Помогите, не могу понять почему после 9 значения начинает коряво читать данные из массива в ПЗУ

```
LDI XL,low(Mas1)
LDI XH,High(Mas1)
;ROM arr
LDI ZL,low(Array*2)
LDI ZH,high(Array*2)

;*****************************************************************************
;забиваем в ОЗУ массив из ПЗУ массива=)
Write_next:
ldi Temp1,0

add ZL,Cell_Arr ;номер элемента массива
adc ZH,Temp1 ;просто ноль :)
LPM Temp, Z ;значение массива заносим в переменую Temp

add XL,Cell_Arr
adc XH,Temp1

st X,Temp

mov TempA,Temp
rcall Out_Port

inc Cell_Arr ;увеличиваем адресс ячейки массива на будущие
```

```
cpi Cell_Arr,32;если перебор то выход
BRNE Write_next
ldi Cell_Arr,0

…
Array:
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101

.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101

.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101

.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101

.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
```

```
.db 0b10101010,0b10101010
.db 0b01010101,0b01010101
```

---

d-lun

16 Апрель 2011 в 11:53

```
SUBI R16,(-1) ; вообще то SUBI это вычитание, но -(- дает +
SBCI R17,(-1) ; А тут перенос учитывается. Но об этом потом.
SBCI R18,(-1) ; Математика в ассемблере это отдельная история
```

Если после выполнения сложения такого вида необходимо проверять бит переноса, то его надо инвертировать.
Т.е. если r18:r17:r16 = 1, то в таком случае

r18:r17:r16 + 1 = 2

и флаг переноса будет установлен.

Если же r18:r17:r16 = $FFFFFF, то

r18:r17:r16 + 1 = 0

а флаг переноса будет сброшен.

---

tooth_fairy

1 Май 2011 в 2:05

здравствуйте. немогли бы вы подсказать как Disassembler настроить так, чтобы адреса отображались побайтово, а не словами.

★ DI HALT

1 Май 2011 в 17:16

Так оно в словах адресуется.

tooth_fairy

1 Май 2011 в 21:05

Т.е. никак нельзя?! Просто вроде в комментах где-то видел, что можно в настройках порыться…

★ DI HALT

1 Май 2011 в 21:06

ХЗ. Никогда не интересовался. А если где и говорил, то это про отображение в окне Memory в режиме отладки, но там просто все — кнопочки выведены сразу же.

tooth_fairy

1 Май 2011 в 21:08

ок. спасибо

tooth_fairy

1 Май 2011 в 21:18

Еще такой вопрос. Чем отличается адресное пространство данных от адресного пространства кода?

★ DI HALT

1 Май 2011 в 21:24

Они просто разные. У них у каждого своя адресация.

Адресное пространство кода — это флеш. Адресное пространство данных — ОЗУ. Ты не можешь положить исполняемый код в ОЗУ, равно как не можешь положить переменные во флеш.

DmitriS

7 Май 2011 в 9:57

Блин, кажется у меня каша в голове после прочтения :)) Хочу разобраться.

У нас есть 2 типа памяти:
1. EEPROM (это флеш на котором лежит код с константами)
2. SRAM (это оперативка)

Также у нас есть регистры.

После включения контроллер считывает с EEPROM и выполняет код. Правильно ли я понимаю что код напрямую к оперативке обратиться не может т.к. у нас Гарвардская модель памяти? Т.е. код работает с памятью только через регистры? Т.е. например нельзя записать в порт сразу байт из ОЗУ?

★ DI HALT

7 Май 2011 в 11:09

Нет не так.

У нас ТРИ типа памяти
1 FLASH — тут лежит код и константы
2 SRAM — тут лежат все переменные сюда же торчат регистры и периферия (в некоторых моделях)

3 — EEPROM — энрегонезависимая память куда можно произвольно писать и читать. Медленная имеет ограниченное число циклов перезаписи.

При включении контроллер начинает выполнять программу из FLASH с адреса 000000. Тут лежит код. А также, обычно, всякие строки данных, массивы таблиц и прочее барахло, что вносится на этапе программирования и потом нужно только на чтение.

Все промежуточные результаты, что не влезли в регистры, лежат в SRAM и после сброса теряются. Тут все локальные (они могут быть и полностью в регистрах, зависит от компилятора и их числа) и глобальные переменные программы.

В Проце все делается через регистры. Любое обращение с данными. Порты это тоже область памяти SRAM т.е. там расклад такой. Вот есть адресное пространство SRAM с 0 адреса по X адрес это регистры, с X по У адреса это порты ввода вывода, с У и до конца — пользовательское ОЗУ под что угодно, те самые 1кб или чо там написано в даташите в разделе SRAM.

Для записи в порт мы должны прописать в него значение. Делается это через регистры, т.к. авр не умеет копировать в режиме память-память. Только память-регистр-память. Не умеет модифицировать память. Только память-регистр-модификация-выгрузка в память.

DmitriS

7 Май 2011 в 11:33

Вот теперь понятно!) Записали прошивку на FLASH и работаем. FLASH по нормальному не меняем в процессе работы прошивки. Как я заметил EEPROM бывает в контроллере, либо цепляется внешняя. Интересно, в случае если цепляется внешняя то будет такая же адресация как и с внутренней просто памяти станет больше? Или там будет все сложнее через порты и шины… ?

★ DI HALT

7 Май 2011 в 11:52

Там все сложней обычно. Связь идет по последовательному протоколу i2c. Естественно это другая адресация (ведь это вообще внешнее устройство)

DA

2 Август 2011 в 20:45

Уважаемый DI HALT,
Вы написали «Вообще, такими вот макросами можно ОЧЕНЬ сильно разнообразить ассемблер, превратив его в мощнейший язык программирования, рвущий как тузик тряпку всякие там Си с Паскалями.»
Хотел вас спросить — что вы думаете про HLA (High Level Assembly)? Есть ли смысл изучать, есть ли будущее?
Спасибо.

★ DI HALT

2 Август 2011 в 21:33

Не слышал про такой.

kurtsvl

15 Август 2011 в 15:30

DI HALT спасибо большое за то что очень остроумно делитесь своими знаниями , ваш ресурс лучший для начинающих что я нашел в рунете , вы разложили бардак в моей голове по полкам . Ps: особенно понравилось про летающий оргАн(сначала подумал про муз инструмент) потом дошло Орган :)

nes

16 Август 2011 в 19:03

У меня вопросы про самый последний в статье пример.

Вопрос 1: Интересно, в какой момент в AVRstudio в EEEPROM «виртуального МК» записываются числа 12,34,45,23? Кажется, даже не в момент компиляции и запуска…

Вот делаю я ctrl+F7, открываю в memory раздел EEEPROM, а там всё FF да FF — никаких 12,34 и т.д. Далее при выполнении LDI (как первой, так и второй) ничего никуда не записывается, а вот при выполнении LPM все регистры наполняются какой-то кашей.

Вопросы 2 и 3: что я делаю не так и что я недопонял?

★ DI HALT

16 Август 2011 в 20:07

А туда они вообще записываются? Насколько помню нет. Если ты создал ЕЕП сегмент, то у тебя на выходе будет еще и еер файл. А вот подцепляет ли его студия при компиляции автоматом или надо где то вручную его подцепить я уже не помню. Т.к. давно не использовал еепром.

---

tokiw

30 Август 2011 в 18:34

Я так нифига и не понял, зачем умножать адреса на 2. Вот пусть наша eeprom имеет емкость 4 Кб. Тогда, раз адресация идет двухбайтными словами, наше адресное пространство составляет 2048 адресов.

Допустим, в ячейке с адресом 0x06D3 и меткой label1 располагаются байты 0F и B5 соответственно. Тогда по командам:

LDI ZL,low(label1*2) ;ZL примет значение 0x06 * 2 = 0x0C
LDI ZH,high(label1*2) ;вообще произойдет переполнение 0xD3 * 2 = 0x01A6 и вероятное значение регистра станет 0xA6

В результате в регистре Z окажется адрес 0x0CA6, который вообще вылезает за пределы адресного пространства.
Автор статьи, разрешите мои мучения, а то цифровая каша в голове мешает жить))

★ DI HALT

30 Август 2011 в 23:31

Вот именно что каша. В AVR ТРИ РАЗНЫХ адресных пространства.
ФЛЕШ
ОЗУ
ЕЕПРОМ

И у них свои адреса, свои приколы. И умножать надо на два только при обращении к флешу. Т.к. он адресуется в ассемблере в словах (на деле также в байтах, потому и умножать надо)

tokiw

30 Август 2011 в 23:40

Так я про еепром говорю. Я понимаю прекрасно, что это особый тип памяти, но ее «приколов» так и не понял. Можно какой-нибудь пример на пальцах? Или может где почитать поподробнее?

★ DI HALT

31 Август 2011 в 0:34

Пример на пальцах вот он есть. Если берем адрес по метке, то его надо умножать на два. Т.к. в ассемблере (в смысле программе) метки адресуются в словах. А нам нужны байты.

tokiw

31 Август 2011 в 0:40

То есть если адрес указывает на ячейку со словом, то, умножив его, мы получим ячейку с байтом? Я честно не понимаю))

★ DI HALT

31 Август 2011 в 0:47

Нет ячейка содержит только байт. Контроллер же восьмибитный и он словами оперировать не умеет.

А вот адрес этой ячейки двубайтный.

Но вот в студии он задается в СЛОВАХ. Т.е. если у тебя есть метка M и если она она реально располагается по физическому адресу 2222, то компилятор присвоит метке значение 1111, т.к. для него первичным являются переходы по командам (которые делаются в словах, т.к. команда двухбайтная), нежели пользовательская адресация. Поэтому значение адреса взятого по метке и надо умножать на два, чтобы получить реальный физический адрес нашего байта.

---

tokiw

31 Август 2011 в 1:10

>первичным являются переходы по командам (которые делаются в словах, т.к. команда двухбайтная), нежели пользовательская адресация.

тут я умер)

★ DI HALT

31 Август 2011 в 1:24

Не пытайся это понять умозрительно (Как и многое в ассемблере) запусти проект в отладчике и погляди ЧТО такое метка на самом деле (во что превращается LDI R16,low(M1) и LDI R17,High(M1) какой адрес у метки на самом деле (можно поглядеть в map файлах или в листинге, где у каждой метки подписан ее адрес. Погляди где по этой метке располагаются данные в памяти и какой у этих данных адрес в реальной памяти программ, Посмотри что и откуда читает команда LPM из какого адреса. И тогда все поймешь.

---

tokiw

31 Август 2011 в 1:51

Так я и поступлю) Спасибо за помощь и советы!

---

alar

2 Сентябрь 2011 в 23:20

Так как длины команд в avr'овском равны либо 2, либо 4, последний бит регистра PC при побайтовой адресации был бы всегда равен 0, разработчики архитектуры решили сэкономить и не записывать младший бит адреса в регистр PC. Чтобы осуществить данный финт ушами, адрес надо сдвинуть на один бит вправо, что эквиваллентно делению адреса на два. В получившейся архитектуре, для того, чтобы правильно осуществлять переход по меткам, адреса следует делить на два, что и делает компилятор. Однако, считывать данные из флеша, мы по прежнему можем только побайтово, и адрес следует указывать полностью, поэтому адрес метки (делённый компилятором на 2) следует умножить на два. Вот как-то так вот вроде бы.

---

tokiw

3 Сентябрь 2011 в 0:53

Вот теперь все стало понятно! Большое-пребольшое спасибо!))

---

tokiw

31 Август 2011 в 0:31

Прошу прощения, перепутал названия! Конечно же я говорил про флеш!

---

http://jabacrack.livejournal.com/

8 Сентябрь 2011 в 16:10

Причем обратите внимание, что у регистров I/O есть еще своя адресация — адресное пространство регистров ввода-вывода (от 00 до 3F), она указана на правой части рисунка.

Все-таки на левой части рисунка.

Nalyana

22 Октябрь 2011 в 17:36

Доброе время суток!
Извините, если вопрос совсем уж элементарный, но для меня стал целой проблемой…

Необходимо по прерыванию (таймера) ставить указатель стека на определенное место, которое вычисляется как смещение от конца памяти на R26*6. В регистре хнанится целое однобайтовое число, которое иногда меняется)
Помогите, пожалуйста, кодом и словом, как установить указатель стека таким образом…

★ DI HALT

22 Октябрь 2011 в 18:26

А в каком месте проблема то?

Nalyana

22 Октябрь 2011 в 18:29

не могу разобраться с двубайтовыми числами

```
ldi R22, SPL
ldi R23, SPH

sub R22, R26*6
sbci R23,0

out SPL,R22
out SPH,R28
```

так можно делать?
если так, то неверно считывает текущее состояние указателя… 3C3D вместо 045F

Nalyana

22 Октябрь 2011 в 18:33

только что понял.. ldi нельзя так использовать…
и вот из-за такого тупления беда (((

★ DI HALT

22 Октябрь 2011 в 18:35

Ну как минимум sub R22,R26*6 должно дать ошибку. Либо скомпилится в какую то херь.

Т.к. нельзя так умножать. Это вам не Си, компилятор за вас умножать не будет (только в пределах констант). Так что все умножение надо делать командами процессора.
Проще выбрать множитель не 6, а 8 и умножать сдвигом.

Т.е. надо умножить R26 на 8?

сдвигаем содержимое R26 три раза влево.

Nalyana

22 Октябрь 2011 в 18:49

В том и проблема.. Я знаю, что этот код не верен. Поэтому и прошу помощи… Как правильно отправить указатель стека в нужное место? Пусть даже с умножением на 8..

★ DI HALT

22 Октябрь 2011 в 18:54

Вычислить нужное смещение.
запретить прерывания
Взять текущее значение SP
Вычесть из него смещение
Положить его в SP
разрешить прерывания.

Nalyana

22 Октябрь 2011 в 18:56

все гениальное просто!) спасибо

★ DI HALT

22 Октябрь 2011 в 19:26

Вот только зачем? Модификация стекпоинтера это весьма нежная и очень глюкогенная операция. Без лишней нужды туда лезть не стоит или отлаживать все на десять раз.

Nalyana

22 Октябрь 2011 в 19:32

Ассемблер пришлось учить только ради решения курсовой… Поэтому и алгоритм коряв, и методы «глюкогенные»… Но увы, ничего более умного придумать я не в силах… Леплю из того, что есть в арсенале фантазии

Nalyana

22 Октябрь 2011 в 19:33

Но вам удалось меня напугать… Теперь пытаюсь обойти это ((

★ DI HALT

22 Октябрь 2011 в 20:49

А в чем тайный смысл всех этих манипуляций?

Nalyana

22 Октябрь 2011 в 20:52

Я уже придумала другой способ…
Но цель одна, циклически считывать содержимое стека и выводить побайтно на портС.
А если уже совсем глобально описывать проблему, то это табло с бегущей строкой…

★ DI HALT

22 Октябрь 2011 в 21:17

Мда, как все запущено :)

Смена указателя стека нужна, пожалуй, только в одном случае — когда пишешь ядро операционной системы с вытесняющей многозадачностью. Бегущая строка явно не тянет на такое :)

Как понял, тебе надо просто брать данные из какого-то массива в памяти и плевать в порт С побайтно. Так?

Nalyana

22 Октябрь 2011 в 21:20

Так… Но при этом еще добавлять данные в этот массив…

Боюсь наглеть, но не хочется выставлять вот так свою глупость на всеобщее обозрение. Есть возможность писать на почту или куда-то еще? Очень бы хотелось, чтобы Вы мне помогли в этом сложном и архиважном деле))

★ DI HALT

22 Октябрь 2011 в 21:48

Ну во первых не глупость. Тут далеко не светила и гуру обитают :) Мне тоже есть чему поучиться. А так глядишь наша дискуссия кому полезной станет, ага.

Т.е. данные еще приходят откуда то извне, с другого порта или УАРТ?

Тут тоже стек не нужен. Нужен буфер в памяти. Т.е. берешь и делаешь что то вроде
.DSEG
Buffer: .byte 256

потом CSEG. и все дела, как обычно.

Т.е. теперь у тебя есть в ОЗУ блок Buffer размером 256 байт, с адресом Buffer

Дальше заводишь два указателя (лучше в памяти, в отдельных переменных, но можно и регистры выделить) которые будут указывать на то куда надо добавить данные и откуда читать. Типичный кольцевой буфер.

Поскольку размер буффера 256 (главное только чтобы памяти ОЗУ у МК хватило. Если не хватит, то соответственно брать меньше) и максимальное число в регистре 256, так что можно не проверять начало и конец, оно будет автоматичесськи.

Дальше просто (считаем что R20 это входной индекс, R21 выходной) байт пришел, обработан и лежит в R16:

Грузим адрес буфера:
LDI ZL,Low(Buffer)
LDI ZH,High(Buffer)

Вычислем смещение нашего указателя, путем прибавления к адресу нашего входного индекса:
ADD ZL,R20
CLR R17 ; получили 0
ADC ZH,R17 ; учли перенос при сложении ZL

Теперь пара Z содержит адрес того места куда нам надо писать входные данные.

ST Z,R16 ; Записали данные в буфер.
INC R20 ; увеличили входной индекс, чтобы запись следующего байта была в следующую ячейку нашего буффера.

Чтение и вывод на порт аналогично, только берем другой индекс — выходной.
Грузим адрес буфера:
LDI ZL,Low(Buffer)
LDI ZH,High(Buffer)

ADD ZL,R21
CLR R17 ; получили 0
ADC ZH,R17 ; учли перенос при сложении ZL

Теперь пара Z содержит адрес того места откуда надо читать входные данные.

LD R17,Z ; считали данные из буфера.
INC R21 ; увеличили входной индекс, чтобы запись следующего байта была в следующую ячейку нашего буффера.

OUT PORTC, R17 выплюнули данные в порт.

★ DI HALT

22 Октябрь 2011 в 21:53

Разумеется надо добавить защиту от перехлеста, чтобы процесс чтения не опережал процесс записи, иначе в порт пойдет мусор.

Но это просто. Надо только сравнивать значение входного и выходного указателей и не допускать того, чтобы выходной был меньше входного. Как то так. Или заинициализировать строку нулями в самом начале, а дальше пусть поток чтения читает по кругу оттуда непрерывно, а поток записи пишет что ему вздумается. Они друг другу мешать не будут.

Nalyana

22 Октябрь 2011 в 22:19

эхх… красиво все, конечно… понятно, подробно… Но так много переделывать, и так много неучтенных деталей задачи в таком алгоритме…
Спасибо огромное за помощь, но мне сейчас проще доделать своего уродца, чем вникнуть в вышеизложенное…
И если уж речь зашла о пояснении, то мне, как новичку, надо объяснять даже места для задержек и вызовов (((

Спасибо!

★ DI HALT

22 Октябрь 2011 в 22:23

http://easyelectronics.ru/avr-uchebnyj-kurs-peredacha-dannyx-cherez-uart.html

Тут еще почитай. Ближе к концу есть готовый пример буфера. Там правда он для UART, но что стоит в выходе заменить UDR на PORTC или куда там надо делать вывод?

Nalyana

22 Октябрь 2011 в 22:23

Последний вопрос… Значение Z-регистра должно быть таким же как ячейки памяти, в Data при отладке?

то есть для mega16 Z=045F покажет на последнюю ячейку?

★ DI HALT

22 Октябрь 2011 в 22:27

Ничего не понял.

Z регистр это иное название пары R30:R31 так сделано просто для удобства. Т.е.

LDI ZL,10 экивалентно
LDI R30,10

Они ранвы нулю на старте (в идеале), а дальше как кривая выведет. Что туда загрузишь то там и будет. Обычно их используют как адресные указатели для работы с памятью (а еще пары Y и Z).

★ DI HALT

22 Октябрь 2011 в 22:30

Т.е. если в Z загрузить адрес 045F
LDI ZL,low(045F)
LDI ZH,High(045F)
то содержимео Z будет = 045F а вот команды LD/ST котоыре с Z работают будут брать/сохранять данные в ту ячейку ОЗУ на которую указывает значение Z. Т.е. в данном случае на адрес 045F

akocur

23 Январь 2012 в 22:49

Не понял вот эту тонкость: "Одна тонкость — дело в том, что адрес метки подставляет компилятор, а он считает его адресом перехода для программного счетчика. А он, если ты помнишь, адресует двубайтные слова — ведь длина команды у нас может быть либо 2 либо 4ре байта.
А данные у нас лежат побайтово и контроллер при обращении к ним адресует их тоже побайтово. Адрес в словах меньше в два раза чем адрес в байтах и это надо учитывать, умножая адрес на два."
Здесь по пунктам я разложил, что мне не понятно http://narod.ru/disk/38627126001/AVR%20LPM.jpg.html

★ DI HALT

23 Январь 2012 в 23:25

Ну вот ты и сам ответил на свой вопрос. Компилятор тебе показывает в словах (и метка data тоже равна 00007). Т.е. адрес 00007 это адрес 7го слова, где лежит твое число. В слове два байта. Но LPM оперирует с адресами в байтах. Потому надо слова умножать на два, чтобы получить адрес в байтах. 7*2 = 14 или 0E

koliakrasnoff

11 Сентябрь 2012 в 11:35

Здравствуйте! Никак не разберусь с Оперативной памятью. Т. е. первые 32 бита её — это РОНы, вторые 64 бита (или сколько там) — это регистры периферии. С последнего байта оперативки и до некоторой глубины вниз — это стек.

А между дном стека и верхушкой периферии — это получается остаются свободные ячейки (и достаточно много), куда мы можем писать все, что нам вздумается, но только эти ячейки более ущербные, так как АЛУ с ними не работает. Работать с ними можно только перезаписью через РОНы.

Я правильно понимаю? И вообще, нужны ли эти «лишние» ячейки между стеком и периферией, если целых 32 аккумулятора имеются в наличии?

★ DI HALT

11 Сентябрь 2012 в 11:39

Все так.

А на одних регистрах далеко не уедешь. Попробуй сохранить в регистр, например TCP|IP кадр. Или крупную посылку которую надо обработать, а вот конкретно сейчас это делать некогда. Опять же состояния автоматов и прочее. В регистрах можно многое сделать, но без памяти все равно никуда. Хотя некоторые тини, вроде тини12, не имеют ОЗУ вообще.

koliakrasnoff

11 Сентябрь 2012 в 13:20

Спасибо огромное, кажется начинаю въезжать…

koliakrasnoff

11 Сентябрь 2012 в 22:55

Разрешите еще вопрос: А вот когда прерывание вызывается, текущий адрес команды и переменные там всякие в стеке сохраняются, программа на адрес вектора прерывания перескакивает и уже оттуда — на обработчик прерываний. Адрес и переменные в стек автоматически сохраняются сразу после вызова прерывания, и выгружаются назад тоже автоматически, как только прерывание отработало, или программист сам должен позаботиться об этом (загрузить и выгрузить)?

Хотя — ну как он сам позаботится. Прерывание же возникает неожиданно.

Получается, если в программе используются прерывания, стеком и командами PUSH и POP пользоваться нельзя. Вдруг прерывание сработает и данные в стеке , закинутые перед этим Пушем испортятся.

★ DI HALT

11 Сентябрь 2012 в 23:09

Адрес возврата сохраняется/восстанавливается автоматом. А вот регистры надо сохранять вручную. Делается это сразу же после входа в прерывание и возвращается взад сразу же после выхода. Сохраняют обычно в тот же стек.

Скажем, прерывание использует в своей работе R1, R16, R17 и Z указатель. Так что мы первым же делом должны

```
PUSH R1
PUSH R16
PUSH R17
PUSH R31
PUSH R32
```

А перед RETI сделать POP всего этого барахла в обратном порядке. Да, еще если в программе используются АЛУ команды (сравнения, вычитания, переходы и тыыд, в общем все, что меняет флаги) то сохранять/восстанавливать надо и SREG. Обычно вначале сохраняем. Причем раз SREG сразу нельзя сохранить, то вначале ныкаем регистры, потом только SREG через SomeVector:

```
PUSH R1
PUSH R17
PUSH R31
PUSH R32
PUSH R16

IN R16,SREG
PUSH R16 Это в стек ушел сред
```

Ну, а потом, на выходе:

POP R16 достали сред
OUT SREG,R16
POP R16 достали R16
POP R32
pop r31
pop r17
POP R1
reti

Высокие языки, вроде Си это делают автоматом.

ОЗУ Обычно сохранять не надо. Что ему будет? Но тут есть другая фигня — атомарность, надо сделить, чтобы прерывание не вклинилось в запись двубайтной переменной и не записало туда же что нибудь. (скажем, по уарту приходят двухбайтные данные. Фоновая программа читает буфер уарта, а прерывание в буфер пишет. Вот чтобы не получилось, что первый байт от одной посылки, а второй байт от другой). В этих случаях при чтении буфера в фоновой проге делают запрет прерываний перед чтением и отпускание прерывания после.

---

ek50hey

11 Ноябрь 2012 в 12:19

Привет!
Вопрос такой: когда пишу
data: .db 12,34,45,23
в сегменте .CSEG, то данные заносятся во FLASH, а не в EEPROM. А как положить именно в EEPROM и достать оттуда?
Пробовал писать в сегменте .ESEG, но тогда метка получается 0x00000000 и при обращении к ней с помощью Z (как в последнем учебном примере) выдается значение не из EEPROM а из FLASH (значение строки LDI ZL,low(data*2)).

---

ek50hey

11 Ноябрь 2012 в 12:26

Все нашел ответ)) http://easyelectronics.ru/avr-uchebnyj-kurs-ispolzovanie-eeprom.html

---

dima_m

15 Ноябрь 2012 в 13:43

DI привет. нуждаюсь в маленькой консультации. Есть ли в си способ как разместить в озу переменные по очереди? Если делаю так:
uint8_t A;
uint8_t B;
uint8_t C;
то они разбрасываются по озу где попало(не знаю почему). А мне надо чтоб по порядку шли. Если сделать так:
uint8_t [3] тогда пропадают имена(A,B,C). Как допустим в таком случае присвоить ячейке uint8_t [0] имя A? Может подскажешь где глянуть? Что то пока не могу найти ничего. Пишу в AVRStudio_6 на СИ.

★ DI HALT

15 Ноябрь 2012 в 15:31

Укатай их в структуру.

Valentin_Ko

18 Февраль 2013 в 2:44

Привет! Нужна консультация. Студия 4.19 ругается на постдекремент и прединкремент в ld и st
ld r18, Y- ; пишет — error: syntax error, unexpected ‘\n’
ld r18, +Y ; пишет — error: syntax error, unexpected ‘+’
постинкремент и предекремент проходит нормально.
Почему?

★ DI HALT

18 Февраль 2013 в 8:34

Потому, что таких команд просто нет.

stream

23 Февраль 2013 в 20:52

Добрый день

Очень прошу пояснить мне разницу конструкций
.db 1, 2
.equ 1

Я не про количество байт, а по расположение в памяти и доступ к ним

Почему .db нужно именно в конце cseg писать, почему нельзя вместо equ написать db с одним значением?

★ DI HALT

23 Февраль 2013 в 21:42

equ это то же что и define в Си. Т.е. мы просто говорим что такая то цифровая величина имеет имя. Т.е. сам формат equ таков:

.equ UDR0 = 0xc6 т.е. вместо UDR0 комплиятор подставит его реальны адрес 0xC6

А db это просто указание компилятору вставить непосредственный байт в код. Вот где написал, там и вставит. Можешь ради лулзов через db в код программы вставлять команды, прописывая их машинные коды. На некоторых процессорах таким образом недокументированные команды юзают.

stream

23 Февраль 2013 в 21:40

Все я сам разобрался.

---

stream

25 Февраль 2013 в 20:59

Прошу прощения. Все перепробывал ничего не пойму, ситуация такая

Data .db 0x01, 0x02

LDI ZL, low(Data * 2)
LDI ZH, high(Data * 2)

LPM R16, Z

Так работает

А вот так нет, возвращает в регист 0
LDD R16, Z + 0

Очень прошу подмкажите причину?
Благодарю

★ DI HALT

25 Февраль 2013 в 22:31

Ну все просто же. LPM работает ТОЛЬКО С флешем, а LDD работает ТОЛЬКО с ОЗУ. ОЗУ и флеш находятся в разных адресных пространствах. Т.е. адрес 0000 в ОЗУ и адрес в 0000 это разные адреса.

stream

25 Февраль 2013 в 23:20

Большое спасибо, у иеня такие мысли и были
А как тогда мне .db размещенный во флеше забирать произвольным индексом? Какая есть аналогичная команда для флеша, чтоб можно было отдать ей значение произвольное индекса, не помледовательное типа Z+ это работает, а именно произвольное, т.е. Z + 3 например?

★ DI HALT

26 Февраль 2013 в 0:55

Нет таких команд. Это же ассемблер, потому только вручную, только хардкор. Грузанул в Z, в другой регистр сунул смещение, сложил, загрузил. А все команды есть в конце даташита в разделе Instruction set summary

stream

25 Февраль 2013 в 23:48

Хотя вроде у меня получилась идея, буду пробовать :)

vek

18 Май 2013 в 19:44

DI_HALT обратите пожалуйста внимание на такую часть статьи «СТЕК»,а именно на самую первую «картинку-фото» ОЗУ. В нижнем правом углу (содержимое ОЗУ) вы подписали с красной такой мощной стрелкой адрес 0x550,и эта жирная красная стрелка указывает на 0x450,очепятка по всей видимости

BIYur

8 Июнь 2013 в 14:53

Вопрос новичка:
Program Counter содержит адрес команды.
Адреса команд двубайтные.
А в Program Counter содержится… трехбайтное значение??? Например, 0x00000E

★ DI HALT

8 Июнь 2013 в 16:18

Вообще он двубайтный. Позволяет организовать память до 65535 адресов слов, т.е. до 128кбайт включительно (на больших МК AVR с 256кб и более уже спец регистр используется). А вот почему он в студии трехбайтный я хз.

---

serj32

27 Октябрь 2013 в 23:57

.macro UOUT
.if @0 < 0x40
OUT @0,@1
.else
STS @0,@1
.endif
.endm
_____

Мне непонятно почему число 0x40 ,а не 0x3F
Спасибо.

---

max_spiral

2 Май 2014 в 22:04

Привет, DI HALT,
Я только начал изучать микроконтроллеры, и у меня сразу вопрос про твой макрос
.macro UOUT
.if @0 < 0x40
OUT @0,@1
.else
STS @0,@1
.endif
.endm
При выполнении STS будет записывать в тот же адрес, что и OUT, а на самом деле он должен
быть сдвинут (он же в пространстве данных, а не регистров), это опечатка или я чего то не догоняю?

★ DI HALT

2 Май 2014 в 22:13

Нет не будет. IF же там не зря стоит. По нему и определяется какую команду надо брать и вставлять вместо макроса. Все что больше 0x40 это мемори маппед адреса и к ним нужна команда STS, а адрес в инк файле уже записан верный.

★ DI HALT

2 Май 2014 в 22:14

Т.е. для каждого вида адреса будет вставляться только правильная команда, а не подходящая будет игнорироваться и в финальном листинге ее не будет.

---

max_spiral

2 Май 2014 в 22:27

А, понял, то есть для UDR я не могу воспользоваться командой STS (STS UDR,R16), это макрос для тех кому лень (или надоело) лазать в даташит.

★ DI HALT

3 Май 2014 в 6:38

ага

★ DI HALT

3 Май 2014 в 6:44

Точнее обратиться в UDR по STS ты можешь, но для этого тебе придется адрес регистра UDR высчитывать вручную учитывая смещение.

А зачем это делать если есть пространство адресов IO и адреса эти уже прописаны в инк файле, а данная команда будет вдвое короче и вдвое быстрей чем если бы все было через STS

anatoli_nik

5 Август 2014 в 13:54

[quote]Вот, например, надо тебе обменять содержимое двух регистров R17 и R16 местами. Как сделать это без использования третьего регистра? Самое простое — через стек[/quote]
На это уйдет четыре команды и 8 тактов процессора.
А можно сделать так
[quote]
.macro obmen
eor @0,@1
eor @1,@0
eor@0,@1
.endm
;Вызов простой
obmen r16,r17

[/quote]
На это уйдет три команды и три такта процессора. Профит!

★ DI HALT

5 Август 2014 в 18:02

О, круто! Спасибо.

Alexm

23 Октябрь 2014 в 20:52

Всем привет, друзья!
Прошу вашей помощи.
В Симуляторе AVR Studio 6.2 не получается выполнить самый первый пример. Код примера:

LDI R18,10
OUT UDR,R18
STS 0x2C,R18

Вторая и третья команды не выполняются (значение в регистре UDR остаётся равным 0). Попытался поэкспериментировать с другими адресами, и столкнулся со странными явлениями. Например, выполняя этот код
LDI R18,255
OUT 0x0B,R18
я пытаюсь присвоить значение 0xff регистру UCSRA с адресом 0x0B (0x2B) 255, то он он станет равным 0x23. Такое ощущение, что что-то не так с регистрами ввода-вывода. Или я что-то делаю неверно?

Alexm

25 Октябрь 2014 в 1:01

Ребята, вопрос снят.
В случае с регистром UDC это был глюк Симулятора Atmel Studio 6.2. Проверил в AVR Studio 4 — всё прекрасно работает.
Что касается регистра UCSRA — там некоторые биты Read Only, поэтому и не получается присвоить этому регистру 255.

---

svd

11 Июнь 2015 в 12:18

data: .db 12,34,45,23
А если использовать оператор .dw, на два тоже нужно умножать, чтобы занести данные в регистровую пару?
Правильно ли я понимаю, что из трёх регистровых пар XYZ можно для этих целей использовать любую?

★ DI HALT

11 Июнь 2015 в 14:49

От типа данных адресация не зависит. Метка же указывает на голову данных. А то что там лежат слова, а не байты знаете только вы. Вам и брать их оттуда придется в два приема, первый байт и второй байт.

Какую индексную пару использовать надо смотреть к конкретной команде. Емнип LPM работает только с Z