Яндекс.Директ

Светодиодная **подсветка** LED!

Низкие цены. Гарантия 3 года. Доставка 0 руб. Настройка. Установка.Звоните!

Светодиодный экран
Бегущая строка
Медиафасад LED табло
ledimperial.ru Адрес и телефон

Ищете смартфон **Xiaomi**?

Широкий модельный ряд в наличии. Поможем настроить, доставим бесплатно!

indexiq.ru Адрес и телефон

xiaomi купить

в Новороссийске. Безнал. Кредит, Доставка, Visa/Mastercard!

Смартфоны Аксессуары
Доставка и оплата Гарантия
macmade.ru
Адрес и телефон
Новороссийск

Адаптеры интерфейса !

Коммуникационные интерфейсные модули ICP DAS, OMRON, Advantech, Siemens!

mirasu.ru Адрес и телефон

AVR. Учебный Курс. Типовые конструкции

AVR. Учебный курс | 9 Июль 2008 | DI HALT | 103 Comments

При написании программ постоянно приходится использовать разные стандартные конструкции вроде циклов, операторов выбора, перехода, сравнения. Всякие там if-then-else или case-switch. В высокоуровневых языках это все уже готово, а на ассемблере приходится изобретать каждый раз заново.

Впрочем, такие вещи пишутся один раз, а дальше просто по наезженной тропинке применяются везде, где потребуется. Заодно и оптимизируются по ходу процесса.

Условие if-then-else

Тут проще всего методом последовательной проверки проложить маршрут до нужного блока кода. Приведу пример:

```
if (A>=B)
        {
        action_a
        }
else
        {
        action_b
        }
next_action
```

Как это проще всего сделать на ассемблере?

Считаем, что A в R16,B в R17, а полезные действия, которые могли бы быть там, заменяем на NOP,NOP,NOP.
Сравнение делается путем вычитания. По итогам вычитания выставляются флаги. Заем (когда A стало меньше B) означает, что условие не выполнилось.

```
                CP      R16,R17         ; Сравниваем два значения
                BRCS    action_b        ; когда A>=B флага C не будет
                                        ; Перехода не произойдет
action_a:       NOP                     ; и выполнится действие A
```

```
5                   NOP
6                   NOP
7                   RJMP next_action        ; Но чтобы действие B не произошло
8                                           ; в ходе естественного выполнения
9                                           ; кода -- его надо перепрыгнуть.
10
11      action_b:   NOP                     ; Действие B
12                  NOP
13                  NOP
14
15
16      next_action:NOP
17                  NOP
18                  NOP
```

Но надо учитывать тот момент, что в случае A=B флаг C не вылезет, зато будет флаг Z. Но переход то у нас исходя из какого-нибудь одного флага (по C=0 или по C=1). И, исходя из выбора команды для построения конструкции if-then-else (BRCC или BRCS), будет разная трактовка результата условия if (A>=B) в случае A=B.

В одном случае (нашем), где переход идет на else по C=1, A=B будет эквивалентно A>B — флага C не будет. И в том и другом случае услове if (A>=B) даст True и переход по BRCS на then.

Если перестроить конструкцию наизнанку, через команду BRCC то условия (A>=B) уже не получится. При A=B не будет флага C и произойдет переход по BRCC на else.

Для создания строгих неравенств нужна проверка на ноль.
Теперь A больше B:

```
1
2
3
4
5
6
7
8
9
```

```
if (A>B)
        {
        action_a
        }
else
        {
        action_b
        }
next_action
```

Получили:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
                CP      R16,R17         ; Сравниваем два значения
                BREQ    action_b        ; Если равно (флаг Z), то переход сразу.
                                        ; Потом проверяем второе условие.
                BRCS    action_b        ; когда A>B флага C не будет
                                        ; Перехода не произойдет

action_a:       NOP                     ; и выполнится действие A
                NOP
                NOP
                RJMP next_action        ; Но чтобы действие B не произошло
                                        ; в ходе естественного выполнения
                                        ; кода -- его надо перепрыгнуть.

action_b:       NOP                     ; Действие B
```

```
15                  NOP
16                  NOP
17
18
19  next_action:    NOP
20                  NOP
21                  NOP
```

Сложно? Думаю нет.

Усложним еще раз наше условие:

```
1  if (C<A AND A<B)
2          {
3          action_a
4          }
5  else
6          {
7          action_b
8          }
9  next_action
```

Считаем, что A=R16, B=R17, C=R18

```
;IF
                CP      R16,R18         ; Сравниваем C и A
                BREQ    action_b        ; Переход сразу на ELSE если C=A
                BRCS    action_b        ; Если A оказалось меньше, то будет C
                                        ; и сразу выходим отсюда на else
                                        ; Но если C<A то идем дальше и проверяем
                                        ; Второе условие (A<B)

                CP      R16,R17         ; Сравниваем два значения
                ; BREQ  action_b        ; Переход на ELSE если B=A. Команда BREQ специально
                                        ; закомментирована -- она тут не нужна.  Я ее поставил
                                        ; для наглядности. Ведь в случае A=B
                                        ; флага C не будет и однозначно будет переход по BRCC.

                BRCC    action_b        ; Когда A>B флага C не будет
                                        ; А переход по условию C clear сработает.
;THEN
action_a:       NOP                     ; Выполнится действие A
                NOP
                NOP
                RJMP next_action        ; Но чтобы действие B не произошло
                                        ; в ходе естественного выполнения
                                        ; кода -- его надо перепрыгнуть.
;ELSE
action_b:       NOP                     ; Действие B
                NOP
                NOP


```

```
30
31
32
```

```
next_action:    NOP
                NOP
                NOP
```

Просто же! Вот так вот, комбинируя условия, можно размотать любую логическую конструкцию.

Битовые операции

Но, пожалуй, самые распространенные операции в контроллере — битовые. Включить выключить какой-нибудь параметр, инвертировать его, проверить есть или нет. Да масса случаев где это нужно. Основная масса операций идет через битовые маски.

Есть замечательные команды SBI и CBI первая ставит указанный бит в порту, вторая сбрасывает. Например:

CBI PORT,7 — обнулить 7й бит в регистре ввода-вывода PORT
SBI PORT,6 — выставить 6й бит в регистре ввода-вывода PORT

Все замечательно, но работают эти команды только в пределах первых 31 адресов в пространстве регистров ввода вывода. Для мега16 это:

```
1
2
3
4
5
6
7
8
9
```

```
.equ    EEARH   = 0x1f
.equ    EEDR    = 0x1d
.equ    EECR    = 0x1c
.equ    PORTA   = 0x1b
.equ    DDRA    = 0x1a
.equ    PINA    = 0x19
.equ    PORTB   = 0x18
.equ    DDRB    = 0x17
.equ    PINB    = 0x16
```

```
10  .equ    PORTC   = 0x15
11  .equ    DDRC    = 0x14
12  .equ    PINC    = 0x13
13  .equ    PORTD   = 0x12
14  .equ    DDRD    = 0x11
15  .equ    PIND    = 0x10
16  .equ    SPDR    = 0x0f
17  .equ    SPSR    = 0x0e
18  .equ    SPCR    = 0x0d
19  .equ    UDR     = 0x0c
20  .equ    UCSRA   = 0x0b
21  .equ    UCSRB   = 0x0a
22  .equ    UBRRL   = 0x09
23  .equ    ACSR    = 0x08
24  .equ    ADMUX   = 0x07
25  .equ    ADCSRA  = 0x06
26  .equ    ADCH    = 0x05
27  .equ    ADCL    = 0x04
28  .equ    TWDR    = 0x03
29  .equ    TWAR    = 0x02
30  .equ    TWSR    = 0x01
31  .equ    TWBR    = 0x00
```

И ни байтом дальше. А с более старшими адресами — облом.

Ну ничего, что нам мешает взять и записать в регистр ввода-вывода (далее РВВ, а то я уже задолбался). Этот бит самому?

Только то, что доступ там к порту идет не побитный, а сразу целыми байтами.
Казалось бы, в чем проблема? Надо выставить биты 1 и 3 в регистре TWCR, например, взял да записал двоичное число 00001010 и выставил.

```
1          LDI     R16,1<<3|1<<1
```

Да, можно и так, но если при этом другие биты в этом регистре равны нулю, а то ведь мы их все накроем сразу.

В таком случае, алгоритм у нас такой:

- Взять старое значение
- Подправить в нем биты
- Записать обратно

Запишем это в коде. Взять значение просто:

```
1          IN      R16,TWCR
```

А установить нужные биты нам поможет битовая маска и операция OR. В результате этой операции, там где в битовой маске были нули будут те же значения, что и раньше. А где в маске 1 возникнут единички.

```
1          ORI     R16,1<<3|1<<1   ; Битовая маска 00001010
```

А затем записать уже измененный байт обратно

```
1
```

```
        OUT     TWCR,R16
```

Сброс битов тоже можно сделать через битовую маску, но уже операция AND. И там где мы хотим сбросить нам надо поставить в маске 0, а где не хотим менять 1

Удобней делать инверсную маску. Для инверсии маски применяется операция побитового НЕ ~

Выглядеть будет так:

```
1
2
3
```

```
        IN      R16,TWCR
        ANDI    R16,~(1<<3|1<<1)
        OUT     TWCR,R16
```

Обрати внимание, для сброса у нас внутри конструкции используется 1. Ведь байт то потом инвертируется!

Для инверсии битов применяется маска по XOR. Нули в этой маске не меняют биты, а единичка по XOR бит инвертирует.

Смотри сам, вверху произвольное число, внизу маска:

```
1
2
3
4
```

```
1100 1100 XOR
1111 0000
_________
0011 1100
```

Правда есть одно ограничение — нет команды XOR маски по числу, поэтому через регистр.

```
1
2
3
4
```

```
        IN      R17,TWCR
        LDI     R16, 3<<1|2<<1  ; Маска
        EOR     R17,R16                 ; Ксорим
        OUT     TWCR,R17                ; Сгружаем обратно
```

Можно увязать всю ботву в макросы и тогда будет совсем хорошо :) Единственно, при использовании макросов, особенно когда они содержат внутри промежуточные регистры, нельзя забывать про эти регистры. А то в ходе исполнения макроса содержимое этих регистров меняется и если про это забыть можно получить кучу глюков :)

Но от такой напасти можно и защититься, например стеком. К примеру, изобретем мы свою команду XRI — XOR регистра с числом.

Макрос может быть таким:

```
1
2
```

```
        .MACRO  XRI
        LDI     @2,@1
```

```
3
4
```

```
        EOR     @0,@2
        .ENDM
```

А вызов макроса

```
1
```

```
        XRI     R16,mask,R17
```

Последний параметр — промежуточный регистр. Указываем его вручную т.к. мало ли какой из регистров нам будет удобней в данный момент.

А можно сделать с прогрузом стека, тогда промежуточные регистры вообще не нужны. Но будет выполняться дольше, занимать больше памяти и требовать стек.

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

```
        .MACRO  XRI
        PUSH    R16
        LDI     R16,@1
        EOR     @0,R16
        POP     R16
        .ENDM
```

Вызов, в этом случае

```
1                       XRI     R17,Mask
```

Но нельзя будет поксорить R16, т.к. его значение все равно затрет при выгрузке из стека.

Также масками можно смело отрезать неиспользованные биты. Например, в регистре TWSR первые два бита это настройки, а остальные — код состояние интерфейса TWI. Так зачем нам эти два бита настройки каждый раз учитывать? Отдавили их по AND через маску 11111100 да дело в шляпе. Или сдвинули вправо.

Сдвиги

Ну тут просто — можно двигать биты в РОН влево и вправо. Причем сдвиг бывает двух типов.

LSR Rn — логический вправо, при этом слева в байт лезут нули.
LSL Rn — логический влево, при этом справа в байт лезут нули.

и через перенос.

ROL и ROR

При этом байт уходящий за край попадает в флаг С, а флаг С вылезает с другого края байта. Получается как бы 9 битная циклическая карусель. Где одна команда дает один шаг влево или вправо.

Как применить? Ну способы разные, например, посчитать число единиц в байте.

Смотри как просто:

```
1         CLR     R18             ; Сюда будем считать единички. Сбросим пока
2         LDI     R17,9           ; Счетчиком циклов будет
3         LDI     R16,0xAA        ; В этом байте будем считать единички
4         CLC                     ; Сбросим флаг C, чтобы не мешался.
5
6   Loop: DEC     R17             ; уменьшим счетчик
7         BREQ    End             ; если весь байт дотикали - выход.
8
9         ROL     R16             ; Сдвигаем байт
10        BRCC    Loop            ; если нет 1 в C, еще одну итерацию
11
12        INC     R18             ; Сосчтитали единичку
13        RJMP    Loop
14
15  End:  NOP
```

Еще через флаг C удобно проверять крайние биты, по быстрому, чтобы не заморачиваться с масками (хотя выигрыша ни по времени, ни по коду не будет).

```
1           ROL     R16
2           BRCS    Label   ; Переход если бит 7 в R16 есть.
```

или

```
1
2
```

```
        ROR     R16
        BRCS    Laber   ; Переход если бит 0 в R16 есть.
```

Циклы

Ну тут все просто — либо сначала, либо потом условие, а затем тело цикла и все окольцовано переходами.

Например, выполнить цикл 20 раз.

```
1
2
3
4
5
6
7
8
9
10
```

```
        LDI     R17,20          ; Счетный регистр

Loop:   NOP
        NOP
        NOP
        NOP
        NOP

        DEC     R17             ; Уменьшаем счетчик
        BRNE    Loop
```

; Переход если не ноль (нет флага Z)

Главное следить за тем, чтобы счетный регистр нигде не запоролся в теле цикла. Иначе получим глюк. Такой глюк, конечно, легко вылавливается трассировкой, но далеко не факт, что он всплывает сразу.

Еще один пример цикла — тупые задержки. Почему я их называю тупыми? Да потому, что они сами ничего не делают и другим не дают. Крайне не рекомендую их использовать, но для простеньких программ подойдет.

Основная идея такой задержки загрузить контроллер бессмысленной работой, чтобы нащелкать как можно больше тактов. У каждого такта есть своя длительность, так что длительность задержки можно формировать от микросекунд, до десятков лет.

Вот решение в лоб:

```
1
2
3
```

```
        LDI     R16, Delay
Loop:   DEC     R16
        BRNE    loop
```

Процессор потратит примерно Delay*2 тактов на бессмысленные операции. При длительности такта (на 8мгц) 1.25E-7 секунд максимальная выдержка будет 6.4E-5 секунды. Немного, но, например, хватит на то чтобы дисплей прожевал отданый ему байт и не подавился следующим.

Если надо больше задержки, то делаются вложенные циклы с несколькими счетчиками. Тогда суммарная длительность выдержки перемножается и уже на трех вложенных циклах можно получить около 2 секунд. Ну, а на четырех уже 536 секунд.

Но есть более красивое решение, нежели вложенные циклы — вычитание многобайтного числа с переносом.

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

```
        LDI     R16,LowByte             ; Грузим три байта
        LDI     R17,MidleByte           ; Нашей выдержки
        LDI     R18,HighByte

loop:   SUBI    R16,1                   ; Вычитаем 1
        SBCI    R17,0                   ; Вычитаем только C
```

```
7           SBCI    R18,0                   ; Вычитаем только C
8
9           BRCC    Loop                    ; Если нет переноса - переход.
```

В результате, вначале у нас отнимается 1 из первого числа, а перенос возникает только тогда, когда заканчивается очередной байт. Получается, что из R16 отнимают на каждой итерации, из R17 на каждой 256 итерации, а из R18 на каждой 65535 итерации.

А всего на таких трех регистрах можно замутить тупняк на 256*256*256 тактов. И никто не мешает в том же ключе навешать еще регистров, вплоть до R31 =) И длительность задержки будет куда точней, т.к. в отличии от вложенных циклов, команда BRCC всегда будет выполняться за 2 такта и лишь в последней случае за один.

Но, опять же, повторюсь, что данные задержки это отстой. Если нужна большая точность — применяют таймеры. Если длительные выдержки, то программные таймеры и цикловые счетчики. О таймерах я расскажу чуть поздней, в периферии, а вот на цикловом счетчике можно остановиться чуть подробней.

Цикловой счетчик

Если нам надо просто затупить, но точность не нужна совершенно — плюс минус несколько миллисекунд не решают. Например, когда задержка нужна лишь для того, чтобы увидеть, что что то происходит, затормозить процесс.

Как у нас в случае тупой задержки выглядит алгоритм:

- Делаем раз
- Делаем два
- Делаем три
- Тупим
- Делаем то, ради чего тупили
- Делаем четыре

Как понимаешь, пока длится этап «Тупим» ничего не работает. Если в «делай раз» было, например, обновление экрана, то оно зависнет. На период этапа «Тупим»

Для этого можно сделать финт ушами — полезную работу ВСЕЙ программы внести в цикл задержки.

Получается так:

- Делай раз
- Делай два
- Делай три
- Тик!
- Натикало? -да- Делаем то, ради чего тупили сбрасываем счетчик.
- Делай четыре

Покажу пример (инициализации стека, и таблицы векторов я опускаю для краткости):

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
        .DSEG
Counter:        .byte   3                       ; Наш счетчик циклов. Три байта.

                .CSEG

MainLoop:

Do_One:         NOP

Do_Two:         NOP

Do_Three:       NOP

```

```
14                  LDS     R16,Counter             ; Грузим наш счетчик
15                  LDS     R17,Counter+1
16                  LDS     R18,Counter+2
17
18                  SUBI    R16,1                   ; Вычитаем 1
19                  SBCI    R17,0                   ; Вычитаем только C
20                  SBCI    R18,0                   ; Вычитаем только C
21
22                  BRCC    DoNothing               ; Не натикало? Переход
23
24  ; Натикало!
25  YES:            NOP                             ; Делаем то, ради чего тупили
26                  NOP
27                  NOP
28
29                  LDI     R16,LowByte             ; Грузим три байта
30                  LDI     R17,MidleByte           ; Нашей выдержки
31                  LDI     R18,HighByte
32
33
34  DoNothing:      STS     Counter,R16             ; Сохраняем обратно в память
35                  STS     Counter+1,R17
36                  STS     Counter+2,R18
37
38                  RJMP    MainLoop
```

Обрати внимание, что другие операции Do_one, Do_Two, Do_Three не ждут когда натикают, они выполняются в каждую итерацию. А операция требующая задержку ее получает, не тормозя всю программу! И таких таймеров можно налепить сколько угодно, пока оперативка не кончится.

Более того, такой цикловой счетчик сам может быть алгоритмозадающим механизмом. Счетчик тикает каждый прогон главного цикла, а каждая операция в этом главном цикле ждет именно своего «номера» в главном счетчике. Получается этакая программная «шарманка», где в качестве барабана с гвоздями выступает наш счетчик.

Но это уже конечные автоматы. К ним я, возможно, вернусь позже. Еще не решил стоит освещать эту тему. А то могу и увлечься. люблю я их :)

Горыныч

Писал я тут библиотечку для подключения LCD к AVR. А чтобы не вкуривать в команды контроллера дисплея я распотрошил Сишный код, дабы подглядеть какими байтами и в каком порядке надо кормить контроллер, чтобы он вышел на нужный режим.

Попутно сварганил один прикольный трюк, который я называю Горыныч. Это когда мы несколько функций объединяем в одну многоголовую — с несколькими точками входа и одной точкой выхода.

Итак, мы имеем две функции обращения к LCD — запись данных и запись команд.

О! Вот эти ребята:

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

```
CMD_WR:         CLI
                RCALL   BusyWait

                CBI     CMD_PORT,RS
                CBI     CMD_PORT,RW
                SBI     CMD_PORT,E
                LCD_PORT_OUT
```

```
8                   OUT     DATA_PORT,R17
9                   RCALL   LCD_Delay
10                  CBI     CMD_PORT,E
11                  LCD_PORT_IN
12                  SEI
13                  RET
```

Нененененене Девид Блейн!!!!

```
1   DATA_WR:        CLI
2                   RCALL   BusyWait
3
4                   SBI     CMD_PORT,RS
5                   CBI     CMD_PORT,RW
6                   SBI     CMD_PORT,E
7                   LCD_PORT_OUT
8                   OUT     DATA_PORT,R17
9                   RCALL   LCD_Delay
10                  CBI     CMD_PORT,E
11                  LCD_PORT_IN
12                  SEI
13                  RET
```

Сейчас я вам покажу особую, оптимизаторскую, магию!

```
CMD_WR:         CLI
                RCALL   BusyWait

                CBI     CMD_PORT,RS
                RJMP    WR_END

DATA_WR:        CLI
                RCALL   BusyWait
                SBI     CMD_PORT,RS
WR_END: CBI     CMD_PORT,RW
                SBI     CMD_PORT,E
                LCD_PORT_OUT
                OUT     DATA_PORT,R17
                RCALL   LCD_Delay
                CBI     CMD_PORT,E
                LCD_PORT_IN
                SEI
                RET
```

Ты что наделал Блейн!!! Ты зачем функцию скукожил??? Нука раскукожь ее обратно! У меня теперь функция беби сайз!!!

Ну, а теперь, для сравнения, поглядим как это сделано в **LCD.c** весь исходник я сюда копировать не буду, только то, что сделано у меня, ну и всю условную компиляцию я тоже выкину.

```
void lcdControlWrite(u08 data)
{
```

```
3       lcdBusyWait();
4       cbi(LCD_CTRL_PORT, LCD_CTRL_RS);
5       cbi(LCD_CTRL_PORT, LCD_CTRL_RW);
6
7       sbi(LCD_CTRL_PORT, LCD_CTRL_E);
8       outb(LCD_DATA_DDR, 0xFF);
9       outb(LCD_DATA_POUT, data);
10      LCD_DELAY;
11      LCD_DELAY;
12      cbi(LCD_CTRL_PORT, LCD_CTRL_E);
13
14      outb(LCD_DATA_DDR, 0x00);
15      outb(LCD_DATA_POUT, 0xFF);
16      }
17
18
19      void lcdDataWrite(u08 data)
20      {
21      lcdBusyWait();
22      sbi(LCD_CTRL_PORT, LCD_CTRL_RS);
23      cbi(LCD_CTRL_PORT, LCD_CTRL_RW);
24      sbi(LCD_CTRL_PORT, LCD_CTRL_E);
25      outb(LCD_DATA_DDR, 0xFF);
26      outb(LCD_DATA_POUT, data);
27      LCD_DELAY;
28      LCD_DELAY;
29      cbi(LCD_CTRL_PORT, LCD_CTRL_E);
30      outb(LCD_DATA_DDR, 0x00);
```

```
31
32
```

```
outb(LCD_DATA_POUT, 0xFF);
}
```

Собственно, строчка в строчку — те же яйца только в профиль.
Что подтверждает, сказанный мной в первом посте, тезис, что не важно на каком языке писать, на си или на ассемблере — главное знать что в какие порты пихать.

Но на асме я могу свернуть свою прогу почти вдвое, не потеряв ни байта и ни такта, а на сях придется для этого вводить внутреннюю функцию, которая захавает оперативку на стек, потребует кучу тактов на переходы и вообще будет выглядеть убого.

И финальная процедура инициализации на сях и на макроасме:

```
1
2
3
4
5
6
7
8
9
10
```

```
void lcdInit()
{
        lcdInitHW();
        lcdControlWrite(LCD_FUNCTION_DEFAULT);
        lcdControlWrite(1<<LCD_CLR);
        lcdControlWrite(1<<LCD_ENTRY_MODE | 1<<LCD_ENTRY_INC);
        lcdControlWrite(1<<LCD_ON_CTRL | 1<<LCD_ON_DISPLAY );
        lcdControlWrite(1<<LCD_HOME);
        lcdControlWrite(1<<LCD_DDRAM | 0x00);
}
```

Ну и у меня:

```
1
2
3
4
5
6
7
8
```

```
        .MACRO  INIT_LCD
        RCALL   InitHW
        WR_CMD  0x38
        WR_CMD  0x01
        WR_CMD  0x06
        WR_CMD  0x0F
        WR_CMD  0x02
        .ENDM
```

Конструкции вида (1<<LCD_HOME) обусловены тем, что Сишный исходник подразумевает конфигурацию LCD — это универсализация кода.

Я тоже мог бы сбодяжить на макросах такое, но мне было лень. Поэтому я просто забил байты. По сути дела разницы нет. Правильно написанный ассемблерный код на очень многих задачах (особенно мелких, вроде этого LCD) куда более компактный и быстрый, **без снижения читабельности**.

Конечно что либо крупное, вроде того же USB я на Асме делать не буду, тут проще взять готовый код. Хотя… чем черт не шутит, может заморочусь и напишу свой крошечный USB драйвер.

Кстати, о готовом коде. В самой студии, в папке первого компилятора есть неплохая пачка примеров работы с периферией:

Готовые примеры кода
Искать это все в Atmel\AVR Tools\AvrAssembler\Appnotes\

Там, помимо привычных нам уже def.inc будут и avr***.asm файлы. Вот это примеры и есть. Там все подробно раскомментировано, правда на английском. И вот что там есть:

- AVR100.asm — работа с EEPROM.

- AVR102.asm — блочное копирование из программной памяти в ОЗУ и из ОЗУ в ОЗУ.
- AVR108.asm — загрузка из программной памяти (пример того, как совать контстанты в код и грузить их оттуда).
- AVR128.asm — работа с компаратором.
- AVR200.asm — умножение и деление.
- AVR200b.asm — умножение и деление вариант 2.
- AVR201.asm — умножение и деление с аппаратными командами.
- AVR202.asm — 16 битная арифметика (сложение. вычитание, умножение).
- AVR204.asm — BCD арифметика. Т.е. на двоично десятичных числах. Нужна для вывода цифр на разные дисплеи.
- AVR220.asm — пузырьковая сортировка.
- AVR222.asm — фильтр с усредняющим скользящим окном. Штука для сглаживания данных, например, с АЦП.
- AVR236.asm — CRC проверка памяти программ.
- AVR240.asm — 4 на 4 матричная клавиатура, с пробуждением от нажатий.
- AVR242.asm — 4 на 4 матричная клавиатура и четырех разрядным 7 сегментным индикатором (часы, таймеры и прочие девайсы такого толка).
- AVR300.asm — i2c Мастер. Простенький, на ожиданиях.
- AVR302.asm — i2c Подчиненный. Тоже простенький.
- AVR304.asm — Программный полудуплексный UART.
- AVR305.asm — Программный полудуплексный UART версия 2.
- AVR320.asm — программный SPI мастер.
- AVR400.asm — аналого-цифровой преобразователь из ШИМ и компаратора. У меня подобный пример тоже в курсе описан.
- AVR401.asm — точный 8 разрядный аналого-цифровой преобразователь. Нужна внешняя обвязка.
- AVR410.asm — декодер RC5 протокола. Один из стандартных протоколов ИК пультов ДУ
- AVR910.asm — ISP программатор.

103 thoughts on “AVR. Учебный Курс. Типовые конструкции”

XDN

17 Сентябрь 2008 в 3:30

Оптимизация -O0 стояла в avr-gcc? У меня самописный драйвер hd44780 с поддержкой чтения добавляет 84 байта при -0s. Тут, конечно, надо учитывать, что код всей прошивки — 3244 Кб. На первых порах, когда ты добавляешь #инклады, размер может резко ползти вверх, но потом этот рост прекратиться.

★ DI HALT

17 Сентябрь 2008 в 8:15

Да на -O0. При -Os код ужимается где то до 5кб. Но при Os не зная что да как легко получить полностью нерабочий код. Т.к. в этом режиме компилер может код так заоптимизировать, что хрен потом концы найдешь. У меня помню по неопытности был прикол, когда я пол дня убил пытаясь понять чего у меня протокол не работает ,а оказывается компилер загнал проверку бита куда то в память и сравнивал ее при проверке с сохраненным значением. Поставил volatile сразу же заработало, но для того чтобы вкурить в эти грабли пришлось кучу документации взрыть.

Ну так то самописный драйвер! Я бы тоже ручками уложил в крошечный обьем. Но в таких случаях предпочитаю нафигачить на асме.

XDN

17 Сентябрь 2008 в 15:46

Использование volatile в таких случая формально указано в стандарте. Оптимизатор — тоже машина, ею управлять надо умеючи. :)

Ну да ладно, не будем холиворить.

★ DI HALT

17 Сентябрь 2008 в 16:21

Так о чем и речь, что при программировании на Сях под МК вылазит куча специфичных тонкостей, завязанных именно на МК, без знания которых ничего не получится сделать. А адекватной документации на предмет того, где да как нужно делать я еще не встречал. Обрывки на форумах не в счет.

XDN

17 Сентябрь 2008 в 16:56

volatile в C99, если не раньше, был введён как раз для таких ситуаций.
Но специфика есть, не спорю. Правда она вся хорошо описывается в документации.

Книгу по Си для МК знаю только одну: «Программирование на языке С для AVR и PIC микроконтроллеров. Ю.А.Шпак.».

Но если взять, например, «Микроконтроллеры AVR семейства Mega. Руководство пользователя. А.В.Евстиеев», то там идут примеры как на Асме, так и на Си. В даташитах Atmel'а тоже с недавних пор стали так делать.

★ DI HALT

17 Сентябрь 2008 в 17:09

Видел я эти примеры и в Евстифееве и в даташитах. Там тот же ассемблер тока вид со стороны Си. Они слишком малы (считаные строчки), чтобы вявить такие приколы с оптимизацией.

---

nwanomaly

17 Сентябрь 2008 в 22:56

у него они скорее как для примера.
и кстати, описывают общий вариант без оптимизаций. я это, например, заметил, когда смотрел его работу с еепромом для меги. всё правильно, но можно сократить)

XDN

18 Сентябрь 2008 в 18:54

Проверил размер своего драйвера «в чистом» виде — 950 байт.

Cyber_RAT

17 Сентябрь 2008 в 11:37

сколько пишу на микроконтроллерах (микропроцессорах начиная от вм80, z80 и конечно же x86) больше всего люблю асм… (я не говорю о большом софте, а об девайсах на этих контроллерах и процикаx)…
ассемблер — более прозрачен что-ли.. или склад ума у меня такой, что линейное программирование для меня проще :(
даже сейчас девайс на меге32 с несколькими протоколами обмена по rs232, mmc+lcd siemens s65 + rtc — все на асме…
p.s. хотя иногда хочется плюнуть и переписать на си (вроде быстрее и проще), но пару попыток это «проще» похоронили в зародыше.
p.p.s. вот такой вот сумбур получился ;)

★ DI HALT

17 Сентябрь 2008 в 11:43

Такая же фигня :))))

SWG

17 Сентябрь 2008 в 20:34

Когда в начале 90х я, поработав уже почти десяток лет на Ассемблере (для 8080, 8086, 8048, Z80) и нескольких BASICах, а также DBASE III+ и CLIPPER, немножко FORT, решил попробовать что — нибудь более серьезное, естественно, начал с C++ 3.0. Про него ходила молва, что он шибко крутой, ужасно мощный и гибкий, но скрытный, как закопанный в землю шланг. В общем, «Настоящие программисты пишут только на C» !!!
Первое, на что обратил внимание, — бедность встроенного набора функций. Даже для элементарного консольного ввода — вывода уже нужна библиотека (stdio.h, если не ошибаюсь). Не было даже такого элементарного понятия, как СТРОКА (string). Вместо нее —

используй одномерный символьный массив, не забывая добавлять в конце символы ВК и ПС. Программку намахал быстро, откомпилировалась без проблем, но чтобы нормально заработала, пришлось попотеть. То вешалась без обьяснения причин, то вообще делала непонятно что.
Для сравнения попробовал написать то же самое на Борланд Паскале 7.0. Написал, компилятор ткнул меня несколько раз во всякие точки и запятые, ругнулся на несоответствие типов данных, после исправления запускаю, и — О, ЧУДО !!! Все работает! и именно так, как надо!
С тех пор я нафиг забросил С, писал на Паскале, затем Дельфи (1, 2, 3, 4, 5, 6…). Работа у меня в последние лет 12 в основном связана с обработкой текстовой информации (биллинговые записи, базы данных, всевозможная статистика, автоматизация отчетности). Конечно, за эти годы С тоже не стоял на месте, многое позаимствовав у других языков, но что — то меня к нему не тянет. А тем более использовать его в микроконтроллерах для пересылки регистр — регистр, чтобы он перед этим на всякий случай сохранил кучу информации в стеке, потом долго возвращал ее обратно…
Правда, языки высокого уровня с микроконтроллерами упрощают такие вещи, как инициализация UART, ADC, I2C, LCD и прочего. Удобно, например, дать одну команду «PWM1_Init(500);» — и ШИМ1 уже настроен на работу на частоте 500гц, вместо того чтобы распихивать битики по десятку служебных регистров.
Или, например:
USART_Init(9600);//-инициализирует USART (9600 baud rate, 1 stop bit, no parity).
А сколько пришлось бы распихать вручную? Или: USART_Write('#'); — и символ ушел в линию. А как вам такое:

```
program ADC_USART;
var temp_res : word;
begin
USART_Init(19200); // initalize USART (19200 baud rate, 1 stop bit, no parity…)
ANSEL := $04; // configure AN2 pin as analog input
TRISA := $FF;
ANSELH := 0; // configure other AN pins as digital I/O
while TRUE do
begin
temp_res := ADC_Read(2) shr 2; //read 10-bit ADC from AN2 and discard 2 LS bits
USART_Write(temp_res); // send ADC reading as byte
```

```
Delay_ms(100);
end;
end.
```

Две команды — и напряжение батареи измерено и ушло в RS232 !
Ради этого стоит написать такие вещи на МикроПаскале или МикроБэйсике, откомпилировать, отладить, а потом взять ассемблерный листинг и дописать на ассемблере все остальное, попутно проанализировав и оптимизировав код, выданный компилятором до этого. Часто это намного ускоряет работу. Например вчера, не шибко напрягаясь, я так проверил работу двух ШИМ, UART, ввод и обработку команд платой управления двигателями своего робота, о чем и похвастался ночью в комментах в теме о робототехнике. Там я написал подробнее, повторяться не буду.

★ DI HALT

17 Сентябрь 2008 в 21:02

У меня под всякие иниты куча макросов обычно пишется :) Так что инклюд файл с макросами и дальше тот же самый UART_INIT_9600 :)

SWG

17 Сентябрь 2008 в 22:03

В конечном итоге так и получается, просто я с микроконтроллерами работаю в последние годы периодически, для души, и не успел еще накопить свою библиотечку макросов, хотя уже кое — что вырисовывается. Конечно, если бы я занимался ими на работе, в серьезных разработках, как было до развала страны, было бы гораздо проще. А так то времени нет, то стимула, то просто лень… Вот и приходится использовать то, что ускоряет и упрощает написание, пока азарт не пропал, и обьемы свободной памяти позволяют. Вот уж наворочу побольше, тогда и займусь оптимизацией и кода, и самого алгоритма, пока же — удерживаясь в рамках поставленной задачи, хочу накопить побольше готовых, функционально законченных, отлаженных кусков, потому что давно уже заметил: чем больше функциональность программы, тем легче добавлять новое. Иногда пара строк или дополнительное условие в уже работающей программе дают такие новые возможности, которые на начальном этапе потребовали бы несколько страниц. Например, сейчас я уже пытаюсь запустить контроль напряжения батареи на своей плате, для начала запуская кнопочкой на одной

из линий порта B, с выдачей результата по RS232. Правда, по ходу выяснилось, что удобнее бало бы использовать для измерения не A4, а например, A0, но два года назад я об этом не подумал, решив при разработке схемы, что подойдет и A4. Но у микроконтроллеров редко входы бывают равноценными, что-нибудь да вылезет.
Но проблема не очень велика и в принципе решаема, в пределах моих требований.
Зато есть возможность шевельнуть мозгой, и получить удовольствие от процесса.

XDN

17 Сентябрь 2008 в 23:15

Re: «Первое, на что обратил внимание, — бедность встроенного набора функций. Даже для элементарного консольного ввода — вывода уже нужна библиотека (stdio.h, если не ошибаюсь). Не было даже такого элементарного понятия, как СТРОКА (string). Вместо нее — используй одномерный символьный массив, не забывая добавлять в конце символы ВК и ПС. Программку намахал быстро, откомпилировалась без проблем, но чтобы нормально заработала, пришлось попотеть. То вешалась без обьяснения причин, то вообще делала непонятно что.»

А что понимается под встроенным набором функций? Так была же библиотека, так? И в названии указано «Standart I/O» (stdio.h). Чем плохо то, что набор функций подключаются опционально?

Чем неудобны нуль-терминированные строки?

FAndrey

17 Сентябрь 2008 в 22:39

Кстати рекомендую сходить на http://code.google.com/p/vector06cc/ они там Вектор эмулируют в АВРке и он у них работает и с ЖК в том числе.

SWG

18 Сентябрь 2008 в 2:42

А что тут удивительного? Процессор Вектора обеспечивал около 400 тыс. операций/сек, и то только на самых коротких, типа MOV A,C. Для AVR это частота порядка 400 — 500 кгц. Кто сейчас такую использует? Для периферии многие микросхемы семейства MCS 51 имеют полноценную системную шину данных и адреса, да еще с отдельными командами для устройств ввода — вывода. И адресация внешней памяти не меньше тех же 64кб. А некоторые так такие обьемы уже внутри имеют. В общем, аппаратно все, что было в 8080 — есть. Система команд тоже похожа, из за чего мне, например, в 80х годах после 8080 легко было освоить 8048. Но кроме того: добавлены команды косвенной адресации, циклы со счетчиком, и даже умножение и деление! А это тоже повышает эффективность кода. Да, еще битовые операции забыл! Так что с современными микроконтроллерами можно многое натворить. Жалко только, что это никому не нужно. Проще нефть продавать, пока не кончилась, а все остальное покупать у китайцев… По крайней мере у нас, в Казахстане, электронщику делать нечего… Программисты еще нужны, и то часто только чтобы научить остальных работников пользоваться 1С или LOTUS. А это уже не программирование, а ликбез.

FAndrey

18 Сентябрь 2008 в 9:23

Да я в принципе в теме :)
Просто проект показался созвучным тому что задумал dihalt поэтому и дал ссылку вдруг еше что полезное найдется, посмотреть чужие реализации иногда не вредно

---

svofski

8 Октябрь 2008 в 15:13

Вектор работает в FPGA, никакой AVR-ки там нету (хоть я и хотел бы иметь мягкую и компактную АВР для эмулятора дисковода).

Cluster

18 Сентябрь 2008 в 3:22

Ох, не знаю что ты там делал, но явно что-то не то. У меня на готовых библиотеках «Hello world» весит меньше двух килобайт. И то, можно оптимизировать её под свои нужды.

А вообще не могу без улыбки читать комменты выше… Глупо опускать какой-то язык, не умея писать на нём.

XDN

18 Сентябрь 2008 в 3:26

Так у тебя hd44780 работал на готовом драйвере?

Cluster

18 Сентябрь 2008 в 3:27

Ну да. А что, не должен? Я его еще немного дописал для своих нужд — вообще шоколадно получилось.

★ DI HALT

18 Сентябрь 2008 в 10:45

А какую библиотеку ты взял? Они же разные есть. Я потащил себе кода на все случаи жизни с http://hubbard.engr.scu.edu/avr/avrlib/

Заюзал библиотеку lcd.c оттуда. Можешь сам посмотреть во что она компилется на Os и O0. Выглядит монструозно.

З.Ы.
Хеллоу ворлд должен быть не более 1к. В идеале байт 300-400.

Cluster

18 Сентябрь 2008 в 15:08

Ту, которая юзается в примерах WinAVR. Там основной минимум. Если убрать все навороты, которые я дописал, то оно будет совсем мелким. Посмотрел то, что по твоей ссылке — там даже какие-то шрифты зачем-то засунули. Если бы ты нашел такой модуль на асме, он весил бы столько же. А оптимизацию ниже O2 я никогда не юзаю, смысла нет.

★ DI HALT

18 Сентябрь 2008 в 15:33

Хм, а где в WinAVR примеры? Щас перерыл доки которые с ним шли чет не нашел ничего вылазящего из стандартной поставки его библиотек.

Cluster

19 Сентябрь 2008 в 20:10

WinAVR\doc\avr-libc\examples\stdiodemo

Саму работу с stdio можно выкинуть нафиг :)

★ DI HALT

20 Сентябрь 2008 в 11:24

Ааа вот она где порылась. Я туда даже не заглядывал.

XDN

18 Сентябрь 2008 в 16:28

Ну не скажи… Взял сейчас свой код и собрал с -O2 — 3558 байт, с -Os — 3244 байт. На мой взгляд, это весьма существенно. При этом никаких подводных камней с -Os у меня ни разу не было.

SWG

18 Сентябрь 2008 в 18:00

Сейчас интереса ради попробовал на МикроПаскале. (Для PIC16F873):

```
program LSD;
var text : string[16];
begin
text := ‘Hello, World !’;
Lcd_Config(PORTB,3,2,1,0,PORTB,4,7,5); // Lcd_Init
LCD_Cmd(LCD_CLEAR); // clear display
LCD_Cmd(LCD_CURSOR_OFF); // turn cursor off
LCD_Out(1,1, text); // print text to LCD, 1st row, 1st column
end.
```

Результат — 616 байт (15%) ROM, 54 байта (30%) RAM
И на симуляторе, и в натуре — все работает.
Добавил еще пару строк:
Delay_ms(1000); // 1 second delay
LCD_Out(2,3,’mikroPascal’); // print text to LCD, 2nd row, 3rd column
Результат — 676 байт (16%) ROM, 66 байт (37%) RAM.

Все заняло минут 15, большее время из них ушло на подключение проводков от индикатора к порту B. И пока еще никакой оптимизации не использовал.
А то: «Os, O0…»

★ DI HALT

18 Сентябрь 2008 в 18:10

А нафига делать конфиг ножек в исполняемой программе? Это же можно и нужно делать в макросах.

SWG

18 Сентябрь 2008 в 18:36

А какой смысл, если я использовал готовую команду (одну!) Паскаля для настройки. Опять же, при другой раскладке ног достаточно перечислить их в самой команде. Вот ее синтаксис из Help:
procedure Lcd_Config(var data_port : byte; db3, db2, db1, db0 : byte; var ctrl_port : byte; rs, ctrl_rw, enable : byte);
То есть я сразу могу указать любые ноги любого порта, остальное сделает компилятор. Но главной целью было поглядеть, какой окажется величина кода, безо всяких оптимизаций, чтобы знать, на что ориентироваться. Так что пока хваленый С в пролете, не смотря на его продвинутую оптимизацию.
Попутно проверил еще один кирпичик к своей программе робота. Быстро и удобно. И всего пять команд, при всей универсальности: Любой текст, в любую строку, с любой позиции, вместе с инициализацией, очисткой и управлением курсором. 5 строк! Куда еще проще — то, чтобы, например, быстро проверить индикатор? Другое дело, когда надо будет уже впихнуть это в готовую сложную программу, я гляну в ассемблерном листинге, что также выдал компилятор Паскаля, какие байтики куда пихаются, и напишу те же макросы или подпрограммки на асме, только уже не столь универсальные (Для экономии ПЗУ).
А вообще, был приятно удивлен полученным результатом. Ожидал, что будет больше, чем на крутом С, да еще с оптимально подобранными оптимизаторами… Правда, тайно надеялся, что ненамного. Но чтоб такое!… Теперь у меня тем более не появится стимула осваивать С для микроконтроллеров. Тем более начитавшись того, что листинги на С одной фирмы не понимаются компиляторами другой, или даже той же, но другой версии… Нет, такой хоккей нам не нужен!

XDN

18 Сентябрь 2008 в 18:50

«какой окажется величина кода, безо всяких оптимизаций» — там настройки оптимизации имеются, надеюсь?

«Другое дело, когда надо будет уже впихнуть это в готовую сложную программу, я гляну в ассемблерном листинге, что также выдал компилятор Паскаля, какие байтики куда пихаются, и напишу те же макросы или подпрограммки на асме,»

А почему не сразу на асме, пользуясь документацией?

SWG

18 Сентябрь 2008 в 19:17

Я уже обьяснял, что, читая эти комменты, решил просто для сравнения посмотреть, сколько будет на Паскале. Конкретно этим компилятором МикроПаскаля я пользуюсь всего с неделю, (раньше пробовал другие компиляторы и С, и Паскаля для микроконтроллеров, но ни один не понравился). Про Микропаскаль же я знал года с 2005, но руки не доходили попробовать. Недавно на каком — то форуме прочитал хорошие отзывы о нем, вот и решил попробовать. А до этого для PIC и AVR использовал asm, как более привычный, да и экономный. Но когда надо быстро проверить разные функции платы контроллера, и размеры программы не имеют пока значения, (Такая ситуация у меня сейчас с платой, сделанной 2 года назад и заброшенной из за семейных обстоятельств), оказалось, что МикроПаскаль вне конкуренции.
Несколько строчек — и уже работает терминал, или АЦП, или ШИМ. На Asme на это ушло бы на порядок больше времени, а жизнь коротка, и интересного в ней еще так много…
Управления опциями оптимизации в МикроПаскале я пока не нашел, но судя по результатам, он и так достаточно оптимален. Размер кода не так уж и велик, а скорость написания и интуитивность понимания — выше всяких похвал. Работаю с ним недолго, но явных ляпов пока не обнаружил. Кстати, у той же фирмы есть еще и МикроСи, и МикроБэйсик, отдельно для PIC, AVR, и еще каких — то контроллеров. Если будет время, попробую проверить еще и их, как на PIC, так и AVR, хотя бы до Меги 32.

XDN

18 Сентябрь 2008 в 19:00

«Тем более начитавшись того, что листинги на С одной фирмы не понимаются компиляторами другой, или даже той же, но другой версии…»

Прекрасно понимаются, если разработчик придерживается стандарта, заданного «свыше». Основная проблема начинается в библиотеках (и Паскаль тут не исключение). Если библиотеки IAR и avr-libc ещё более-менее взаимозаменяемы, то у множества малоизвестных компиляторов — нет.

SWG

18 Сентябрь 2008 в 18:12

В 3 строчке снизу должно быть «подключение». Работаю на клаве вслепую, иногда пальцы промахиваются. Жалко, нет редактирования…

Agrin

16 Апрель 2009 в 15:30

Для примера на CodeVisionAVR v1 то же самое и статистика. Библиотека встроенная, время 10 минут.
Исходник сокращённый:

```
#include
// Alphanumeric LCD Module functions
#asm
.equ __lcd_port=0x18 ;PORTB
#endasm
#include

void main(void)
{
lcd_init(16);

while (1)
{
lcd_gotoxy(0,0);
lcd_putsf(«Hello, World !»);
};
}
```

Статистика самого компилятора:

Data Stack size: 256 byte(s)
Estimated Data Stack usage: 7 byte(s)
Global variables size: 4 byte(s)
Hardware Stack size: 764 byte(s)
Heap size: 0 byte(s)
EEPROM usage: 0 byte(s) (0,0% of EEPROM)
Program size: 291 words (7,1% of FLASH)

..и асма:

ATmega8 memory use summary [bytes]:
Segment Begin End Code Data Used Size Use%
_____
[.cseg] 0x000000 0x000246 556 26 582 8192 7.1%
[.dseg] 0x000060 0x000164 0 4 4 1024 0.4%
[.eseg] 0x000000 0x000000 0 0 0 512 0.0%

вполне оптимально.

SWG

18 Сентябрь 2008 в 20:51

Для сравнения решил попробовать МикроСи той же фирмы, на том же PIC16F873, с аналогичной программой.

```
char *text = «Hello, World !»;
void main() {
Lcd_Init(&PORTB); // Lcd_Init
LCD_Cmd(LCD_CLEAR); // Clear display
LCD_Cmd(LCD_CURSOR_OFF); // Turn cursor off
```

```
LCD_Out(1,1, text); // Print text to LCD, 1st row, 1st column
}
```

Результат — 295 байт (7%) ROM, 40 байт (22%) RAM

Добавил пару строк:
Delay_ms(1000);
LCD_Out(2,6,»mikroE»); // Print text to LCD, 2nd row, 6th column
Результат — 345 байт (8%) ROM, 47 байт (26%) RAM

Стало уже интересней… И никакой оптимизации…
Надо будет еще, например, на Меге 8 для сравнения проверить МикроПаскаль и МикроСи, а также заодним интереса ради МикроБэйсик, тоже на PIC и AVR…

SWG

18 Сентябрь 2008 в 21:23

Для сравнения попробовал еще МикроБэйсик той же фирмы, на том же PIC16F873, с аналогичной программой.

```
program Lcd_Test
dim text as string[16]
main:
text = «Hello, World !»
Lcd_Config(PORTB,3,2,1,0,PORTB,4,7,5) ‘ Lcd_Init
LCD_Cmd(LCD_CLEAR) ‘ Clear display
LCD_Cmd(LCD_CURSOR_OFF) ‘ Turn cursor off
LCD_Out(1,1, text) ‘ Print text to LCD, 1st row, 1st column
Delay_ms(1000)
```

LCD_Out(2,4,»mikroBasic») ‘ Print text to LCD, 2nd row, 6th column
end.

Результат — 675 байт (16%) ROM, 65 байт (36%) RAM.

XDN

18 Сентябрь 2008 в 22:32

Очень интересно. Похоже на то, что версии библиотек разные.

Если бы не было оптимизации — прошивка была бы в 15-20 Кбайт. Она есть, но не настраивается.

★ DI HALT

19 Сентябрь 2008 в 10:10

Да откуда там 15, максимум 5

XDN

19 Сентябрь 2008 в 19:07

Если компилятор будет класть всё «как есть» — просто танком заменяя функции на ассемблерные вставки — можно и 15 получить.

Для примера — мой драйвер hd44780 без оптимизации занимает 9454 байт, супротив 950 байт с рабочей оной.

Int_13h

18 Сентябрь 2008 в 21:54

Сорри за оффтоп но хелп!.. прерывания не пашут почему то ни в авр студио ни в реальном мк вот код

```
.INCLUDE «m16def.inc»
.CSEG
.org 0
rjmp main
.org OVF0addr rjmp Tmr0_OVF ;Overflow0 Interrupt Vector Address

main:

…

OUTI TCNT0,0
OUTI TCCR0,2
OUTI TIMSK,1<<TOIE0

LOOP:
nop
rjmp LOOP

Tmr0_OVF:
…
```

★ DI HALT

19 Сентябрь 2008 в 7:56

SEI забыл в самом начале программы. По дефолту прерывания запрещены.

SWG

20 Сентябрь 2008 в 0:37

Почитал еще раз все комментарии, и жутко стало. Вспомнил, как я, бывало, лет 20 назад писал программки для своих контроллеров, и в основном размер был 2-4 кб…
И это при одновременной и независимой работе в реальном времени с несколькими телеграфными каналами, накоплением и выдачей статистики непосредственно после измерений, или по запросу за заданный интервал, или автоматически через задаваемые отрезки времени (например, раз в два часа)… Да что там мои программы, даже операционная система CP/M 80 была всего 6,5 Кб! (Все три ее секции, вместе с BIOS)! А тут драйвер паршивенького индикатора на пару коротеньких строчек — на несколько килобайт, да еще с разными оптимизациями. Вот до чего техника дошла!
У меня есть несколько приборчиков с этим индикатором (частотомеры, измерители LCF), в основном повторение готовых конструкций с Интернета, и размер кода у них обычно меньше 2 килобайт.
Так чего же надо было напихать в этот паршивый «драйвер на С», чтобы он стал 2-3, а то и 5, (и даже 15)кб? Это же ведь не WINDOWS!
Богато жить стали. Не глядя пару килобайт туда, пару сюда, мало памяти — поставим монстрика на сотню ног, хоть и мелких…
И уже лень мозгой шевельнуть, лучше с опциями оптимизации компилятора поиграться. А ведь когда — то, почти 40 лет назад, Луноходы чуть ли не на лампах делали, и ничего — по Луне по полгода бегали…
Интересно, а сейчас кто-нибудь на территории бывшего Союза смог бы такое смастерить, хотя бы на современной элементной базе? Да чтоб не на пару дней в комнате, а с месяц по Луне? Куда катимся… Стыдно, мужики. И за державу обидно.

XDN

20 Сентябрь 2008 в 1:04

«И уже лень мозгой шевельнуть, лучше с опциями оптимизации компилятора поиграться.»
Что вы подразумеваете под «шевелением мозгой»?

SWG

20 Сентябрь 2008 в 1:34

Да просто написать простенькую подрограммку или макрос на ассемблере. Привыкли, как в Виндах: Чуть что — сразу давай готовый драйвер, вместо того, чтобы просто в нужном месте несколько команд написать. Конечно, универсальность хороша, удобно иметь сразу все на все случаи жизни. Да только во что это выливается, мы уже видим. Чего уж такого сложного в работе с этим индикатором, чтобы тратить на него несколько килобайт? (Ведь это ТЫСЯЧИ команд!). Но в реале всего — то надо настроить

ему режим при старте, да затем писать в него время от времени, зачастую даже не в конкретную позицию, а строку целиком, когда работает внутренний счетчик в индикаторе, только пихай ему байты. Читать из символьного индикатора — вообще какой смысл? проще сделать буфер вывода в программе. Вот и получается, что драйвер как таковой и не нужен. Несколько команд инициализации в начале и простенькая функция вывода строки на дисплей. Кинул ей инфу в буфер или указал адрес первого символа в ПЗУ — и несколько команд в простеньком цикле перекидают все в индикатор. Это же не картинки рисовать.

XDN

20 Сентябрь 2008 в 1:51

Можно использовать чтение, чтобы проверить присутствие модуля как такового.
Когда идёт проверка флага занятости, то почему бы и не прочитать текущий адрес, если его уже и так выдадут?

«Да просто написать простенькую подрограммку или макрос на ассемблере.»
Я так понимаю, вы предлагаете для каждого нового проекта переписывать код управления заново?

SWG

20 Сентябрь 2008 в 2:33

Не так уж он велик, этот код. Просто последовательность из нескольких команд, можно просто выделить в редакторе несколько строчек и вставить в новую программу. Если нужно часто, можно эти строчки сохранить в отдельном файле. Что уж в этом сложного, если вы действительно делаете сложные вещи? А простую программу вообще иногда проще написать заново, чем приспосабливать готовые куски. Ну, а если программы как таковой еще нет, а нужно всего лишь быстро проверить уже готовое железо, как у меня сейчас, — так вообще никаких драйверов не надо, на том же МикроПаскале, к примеру, написал кусочек с парой команд — проверил индикатор, еще несколько команд — RS232, еще несколько команд — проверил ШИМ, и т.д.
Когда же буду писать окончательно уже рабочую программу, скорее всего, все же буду использовать ассемблер, поглядывая иногда на листинги, скомпилированные Паскалем, чтобы меньше лазить по документации (У меня от нее уже глаза болят, зрение сдавать стало. Четверть века пялился на мониторы, часто довольно хреновые). Вот опробую еще I2C, пищалку, ИК локаторы, одометры, еще кое — что из периферии, раскидаю все по прерываниям, — и начну уже писать все начисто. Правда, надо еще

уголь привезти и печки отремонтировать, да и отпуск через неделю кончится, а работу еще никто не отменял, и до пенсии еще 7 лет пахать (у нас мужикам в 63года).

XDN

20 Сентябрь 2008 в 2:40

А чем «копипастинг» отличается от инкладов в самом ассемблере? Думаю, что ничем. А набор инкладов — уже библиотека. Чем она будет лучше самописной библиотеки на любом другом языке?

SWG

20 Сентябрь 2008 в 3:59

В отличие от обсуждавшихся библиотек на С — размером! Десятки ассемблерных команд вместо тысяч (Судя по размерам драйвера LCD в С).

XDN

20 Сентябрь 2008 в 4:06

Обсуждавшиеся библиотеки на Си — это не самописные библиотеки на Си, о коих говорилось постом выше, хотя я и не привязывался к конкретному ЯВУ.

Cluster

20 Сентябрь 2008 в 2:29

Дык драйвер, который я юзаю, только это и делает — инициализация, да запись. Ну ещё я работу с несколькими буферами прикрутил, что в принципе при желании можно убрать…

★ DI HALT

20 Сентябрь 2008 в 11:27

Не забывай, что у AVR жутко громоздкая система команд. Все команды минимум двубайтные — какой нибудь сраный NOP и тот занимает два байта. А если уж переход куда-нибудь, то целых четыре байта. Плюс ко всему нет переходов с проверкой. Т.е. вначале ты проверь, а потом переходи. Я тут поглядел на свой код который у меня на AT89C2051 был так я в шоке был. Мне там память в 2кб безграничной казалась, а тут тока начал уже и 2кб пролетели. Пишу на асме и зажимаю каждый байт, а все равно.

XDN

20 Сентябрь 2008 в 19:31

Кстати, да. Почему-то совершенно забыл про то, что с развитием МК растёт и средний размер команд. Не знаешь, как с этим делом у PIC'ов?

SWG

20 Сентябрь 2008 в 23:03

У PIC12 — 12 бит, PIC16 — 14бит, 33-35команд.
У PIC17 — 16 бит, 58 команд (добавлен встроенный аппаратный CAN).
PIC18 и выше — пока не интересовался. Пока мне PIC16F877 за глаза.
Все команды — 1 слово. Выполнение — 1 цикл (4такта). Команды сравнения и перехода — 2 цикла (8 тактов). Все команды условных переходов при совпадении или несовпадении условия (есть для обоих вариантов) просто пропускают следующую команду, выполняя вместо нее NOP. Обычно, пропускаемая команда содержит переход. Внутренняя структура делает ненужным сохранение чего-либо в стеке, кроме адреса возврата и флагов, что делается автоматически, поэтому и стек небольшой (у PIC16 — 8 вложений, обычно хватает. У Intel 8048 было аналогично, мне хватало).
Команды условных переходов есть как с проверкой байт, так и бит, а также автодекрементные и автоинкрементные циклы.

---

smex

18 Август 2009 в 14:04

Вот немогу понять где сохраняется конфигурация после инициализации устройства. Например SWG писал Lcd_Config(PORTB,3,2,1,0,PORTB,4,7,5); // Lcd_Init. Но ведь где то надо хранить все эти параметры? Пытался найти эти либы в WinAVR — не вышло (. Предполагаю что настройки закидываются в стек — но если так — то необходимо почистить за собой потом — типо lcd_destoy(). Вообщем вопрос — как же на самом деле настройки храняться?

★ DI HALT

18 Август 2009 в 14:11

Конфиг просто записывается в регистры портов и сидит там. Т.к. кроме LCD к ним ничего не подключено, то пускай там и остаются. Стек при этом не используется :)

---

smex

18 Август 2009 в 15:31

Ну тогда еще нюансик. Конфиг пускай сидит в регистрах портов). Но ведь оно должно где то должно помнить что lcd инициализированн именно на PORTB — а не на каком то другом. А там помойму для этого места не хватит…

★ DI HALT

18 Август 2009 в 16:09

Так это в дефайнах задается. И потом, при компиляции, просто выходит так, что прога написана исключительно для LCD на порту Б

---

legion

14 Апрель 2010 в 0:19

Когнитивный диссонанс.

```
CP R16,R17
BRCS action_b
action_a: NOP
; Сравниваем два значения
; когда A>B флага C не будет
; Перехода не произойдет
; и выполнится действие A
```

Но ведь если R17=R16 переход вроде как тоже не происходит, флаг C=0 и мы попадаем в action_a. А по условию if (A>B) при A=B мы должны выполнять action_b.

Такая же фигня с BRCS и при (A>=B), и (C<A AND A<B). Мозг взорван.

★ DI HALT

14 Апрель 2010 в 1:05

Так и есть. Вариант A=B должен обрабатываться дополнительным условием на флаг Z

★ DI HALT

14 Апрель 2010 в 1:49

Вот, поправил статью. Так стало куда логичней.

---

legion

14 Апрель 2010 в 11:22

Для четких неравенств не обязательно проверять на ноль, достаточно CP.

И фраза «разная трактовка результата условия if (A>B) в случае A=B» сносит крышу немного.

Следуя правилу «Критикуешь — предлагай» опишу момент с ветвлением if then так, как я его понял, будучи новичком в асме:

Рассмотрим результат операции CP R16,R17.
1) R16>R17
Результат вычитания R16-R17 больше нуля, перенос не происходит и флаг C=0.
2) R16=R17
Результат вычитания R16-R17 равен нулю, перенос не происходит и флаг C=0.
3) R16=R17 мы можем воспользоваться командой ветвления BRCC и перейти к действию при C=0.
А для четкого неравенства if (A меньше B) then (action_a) else (action_b) ассемблерный вариант будет выглядеть так:

```
;Пусть C16=A, C17=B.

                CP      C16,C17
                BRCC    action_a

action_b:       NOP
                NOP
                NOP
                RJMP    next_actions

action_a:       NOP
                NOP
                NOP

next_actions:   NOP
                NOP
                NOP
```

Для конструкции if (A<B>=B) и (A<B) уместно дать читателю задание переписать код для (AB), и более сложных неравенств. Это позволит закрепить понимание работы CP и ветвлений в целом. Ответ можно разместить в комментариях либо в конце статьи.

---

legion

14 Апрель 2010 в 11:26

Блин, на тэг нарвался.

Для конструкции if (A меньше B) then (action_a) else (action_b):

```
1                CP      C16,C17
2                BRCS    action_a
3
4   action_b:    NOP
5                NOP
6                NOP
7                RJMP    next_actions
8
9   action_a:    NOP
10               NOP
11               NOP
12
13  next_actions: NOP
14               NOP
15               NOP
```

Кстати, если говорить от задачках, то после описания кода при (A больше или равно B) и (A меньше B) уместно дать читателю задание переписать код для (A меньше или равно B), (A больше B), и более сложных неравенств. Это позволит закрепить понимание работы CP и ветвлений в целом. Ответ можно разместить в комментариях либо в конце статьи.

★ DI HALT

14 Апрель 2010 в 11:52

Можно и на одном CP я же добавил в вариант где C<A AND A<B пример работы через BRCC, где при определеном построении конструкции проверка на 0 не нужна.

---

legion

14 Апрель 2010 в 12:13

Мне кажется в варианте где C в сравнении A и C BREQ тоже можно убрать, достаточно лишь поменять местами операнды в CP. Также и в примере для (A больше B)

★ DI HALT

14 Апрель 2010 в 14:30

Можно, но я не стал. Пусть будут разные варианты.

Freerider

29 Сентябрь 2014 в 11:44

Вставлю свои 5 копеек насчет проверки величин
1. If(A>=B){action_a};
{main}
CP A,B
BRLO MAIN (BRLO если безнаковые величины BRLT знаковые например если сравнивать 10>-14)

Action_a
MAIN:
Мне все таки нравятся команды brlo и brsh хотя они идентичны brcs и brcc .Они дают представление какие операции мы проводим с числами

Freerider

29 Сентябрь 2014 в 11:49

Да еще насчет сброса битов
ANDI R16,~(1<<3|1<<1)
Есть такая команда как cbr которое делает то же самое

legion

17 Апрель 2010 в 18:41

А как проверить крайние биты с использованием битовой маски так же быстро, как сдвигом через C?

★ DI HALT

17 Апрель 2010 в 19:41

Хм. Ну разве что

ANDI R16,0b10000000
BREQ Zero

auara

29 Апрель 2010 в 4:06

Можно подправить, хотя не принципиально
>> нет команды XOR маски по числу, поэтому через регистр.
>> IN R17,TWCR
>> LDI R16, 3<<1|2<<1 ; Маска
наверно хотели так:
LDI R16, 1<<3|1<<2

---

moroz

28 Октябрь 2010 в 13:52

Цитата:
ORI R16,1<<3|1<<1 ; Битовая маска 00001001

Сколько не крутил, но 1<<3|1<<1 дает битовую маску 00001010. Или я что-то не понимаю. Сильно задумался…

★ DI HALT

28 Октябрь 2010 в 14:04

Вы правы, это я у меня опечатка. Щас пофиксю.

---

netouch

23 Декабрь 2010 в 19:44

Доброго дня суток!
Объясните пожалуйста. Перед использованием команды CP флаг C в sreg нужно сбрасывать? (или весь sreg в стек пихать)?
Ведь возможен вариант, когда флаг C будет установлен еще до выполнения команды CP, и тогда последующий переход по BR.. будет неправильным.

★ DI HALT

23 Декабрь 2010 в 19:58

Нет не нужно. CP ставит флаг или снимает его в зависимости от результата операции.

---

shiva

2 Январь 2011 в 19:50

«ORI R16,1<<3|1<<1 ; Битовая маска 00001001»

разве битовая маска в этом случае будет не 00001010?

★ DI HALT

2 Январь 2011 в 20:35

Ты прав. Надо пофиксить

---

tooth_fairy

2 Май 2011 в 2:59

прошу помочь разобраться с задержками, разобранными в примерах
LDI R16, Delay
Loop: DEC R16
BRNE loop
в данной программе delay мы сами подбираем для получения нужной задерки или что?

LDI R16,LowByte
LDI R17,MidleByte
LDI R18,HighByte

loop: SUBI R16,1 ; Вычитаем 1
SBCI R17,0 ; Вычитаем только C
SBCI R18,0 ; Вычитаем только C

BRCC Loop ; Если нет переноса — переход.

★ DI HALT

2 Май 2011 в 9:09

Сами подбираем

---

tooth_fairy

2 Май 2011 в 3:03

LDI R16,LowByte
LDI R17,MidleByte
LDI R18,HighByte

loop: SUBI R16,1
SBCI R17,0
SBCI R18,0

BRCC Loop
аналогичный вопрос по этому коду. Значения lowByte, midbyte,highbyte тоже как-то подбираются? и как вы посчитали что задержка будет 256*256*256 почему не 65535?

★ DI HALT

2 Май 2011 в 9:08

Выставляются исходя из необходимой задержки и тактовой частоты.

Потому что 65535 это всего два байта, а у нас три.

tooth_fairy

2 Май 2011 в 11:49

А т.е. это как бы максимальная задержка для трех байт?! Когда во всех разрядах будут 1? или я чего-то не понимаю?!

★ DI HALT

2 Май 2011 в 12:00

Да

tooth_fairy

2 Май 2011 в 12:07

спасибо

tooth_fairy

2 Май 2011 в 14:53

а как применять эти задержки, где их используют?

★ DI HALT

2 Май 2011 в 15:27

Там где нужна задержка. Случаев миллион, хотя чаще всего лучше использовать системный таймер и диспетчеризацию, т.к. тупая задержка по сути дела вешает контроллер на время отработки. Это дальше описано будет.

deses

2 Сентябрь 2012 в 2:23

Может быть конечно не в тему но куда еще написать я не нашел. Есть два вопроса:
использование .if .else в конструкциях макросов на асме — это вроде Си. И если можно про другие варианты Сишных комбинаций в асме. Информации по ним не нашел никакой.
И второй вопрос с подключением файлов к проекту через .include . Допустим есть какой-то фрагмент (например delay) который выполняет «тупую» задержку, и есть фрагмент который выполняет «стартовую инициализацию портов, таймеров » (например init). Я тут загнался попробовать их оформить как подключаемые файлы .include . Во первых я не понял какое расширение давать файлам .INC или .ASM работают оба (?). А вот дальше самое интересное. Если include init.asm поставить в самом конце программы то инициализация зацикливается поскольку
rjmp init вызывает подключенный файл init.asm и при выходе из него программа оказывается опять на той же строке rjmp init. Если include init.asm прописать после всех векторов прерываний то вроде все нормально. Теперь про .include «delay.asm» . Эту строчку приходится прописывать в самом конце программы и встречающаяся строка rcall DELAY (в основном теле) перебрасывает в подключеный файл откуда выполнение благополучно возвращается в основное тело. Если же .include «delay.asm» прописать не в конце а в начале или в середине или после rcall delay то выполнение туда заходит и от туда не выходит. Объяснение нахожу пока только одно это разница в командах перед метками rjmp и rcall …. пока строчил кажись начал понимать. но всеже как и с каким расширением подключать. вот текст проги

```
.include «M16def.inc»
.def temp=r16
.org $000
rjmp init
.org $012
rjmp T0_OVF
.include «init.asm»
LOOP:
sleep
nop
rjmp LOOP
```

```
T0_OVF:
clr temp
out PORTD,temp
rcall DELAY
ser temp
out PORTD,temp
ldi temp,0xFf
out TCNT0,temp
reti
.include «delay.asm»

init.asm
INIT:
ldi temp,low(RAMEND)
out SPL,temp
ldi temp,high(RAMEND)
out SPH,temp
clr temp
out DDRB, temp
ldi temp,0x01
out PORTB,temp
ser temp
out DDRD,temp
out PORTD,temp
ldi temp,0x20
out MCUCR, temp
ldi temp,0x01
out TIMSK, temp
ldi temp,0x07
```

```
out TCCR0,temp
sei
ldi temp,0xFC
out TCNT0,temp

delay.asm
DELAY:
Ldi r19,10
ldi r20,255
ldi r21,255
dd:
dec r21
brne dd
dec r20
brne dd
dec r19
brne dd
ret
```

★ DI HALT

2 Сентябрь 2012 в 2:38

1. Читай описание ассеблера AVR. Есть в справке внутри студии. По F1 вызывается :) Там все директивы компилятора. Их немного. К си они не имеют отношения, хотя похожи.
2. Инклюд тупо берет кусок текста из файла на который ссылается и вставляет вместо себя. Соответственно где воткнешь там этот кусок ссылаемого кода и будет. На расширение ему пофигу, он по дефолту считает, что там текстовый файл.

---

deses

2 Сентябрь 2012 в 3:04

Спасибо DI теперь понял

ALTERoss13

4 Январь 2013 в 7:50

Доброго времени суток!
Вопрос по типовым конструкциям, конкретно случай if (A>B) {action_a} else {action_b}
Допустим, есть такой код:

```
LDI R16, -3
LDI R17, 10

CP R16, R17 ; if R16 > R17
BRCS false
BREQ false

true: NOP
RJMP true

false: NOP
RJMP false
```

R16 меньше R17, следовательно программа должна перейти к метке false.
Но -3 — 10 не дает ни флага Z, ни флага C, поэтому при неправильном условии выполняется метка true.
Не правильней ли здесь будет заменить BRCS на BRMI?

★ DI HALT

4 Январь 2013 в 8:57

А надо различать знаковые и беззнаковые операции. Они вычисляются совсем по разному и применяются в разных случаях. Потому как знаковое -x будет больше беззнакового x на один старший разряд (уходящий под знак)

Wan-Derer

21 Октябрь 2013 в 12:09

Н-да… Немилосердно заставлять новичков сразу думать ассемблером. ИМХО, два первых листинга должны быть проиллюстрированы блок-схемами. Т.е. ты же сначала объясняешь сам принцип ветвления, а потом показываешь его реализацию на языке. Значит надо принцип проиллюстрировать графически, а потом показать как «дерево» вытягивается в линейный код. У тебя это лихо получается жирными стрелочками. Вообще блок-схемы архиполезная вещь. Хоть программисты, даже начинающие, и относятся к ним презрительно и считают придурью преподавателей информатики. Конечно, рисовать всю программу в виде блок-схемы бессмысленно, но отдельные куски можно и нужно! Сразу видно как кусок работает, как лучше задать граничные условия тех же циклов и ветвей, где кроются грабли и каковы пути оптимизации.

Andrey Utkin

16 Декабрь 2013 в 7:09

а на сях придется для этого вводить внутреннюю функцию, которая захавает оперативку на стек, потребует кучу тактов на переходы и вообще будет выглядеть убого.

Inline-функция или макрос сработают наравне с ручным ассемблерным слиянием этих процедур, мне кажется.

---

and_master

12 Август 2014 в 0:17

Привет, Di! Пару вопросов по примеру с цикловым счетчиком. Неочевидны некоторые вещи.
1. Значения counter (+1,+2) должны быть предварительно загружены и именно они определяют задержку куска Yes?
2. Зачем в каждой итерации писать из регистров в оперативку и из оперативки обратно в регистры?

3. Зачем вообще этот counter, если мы можем просто до главного цикла записать счетчик в регистр и с каждой итерацией уменьшая его проверять наши тики до входа в затупленный участок кода.
4. Сбрасывать счетчик разве нельзя напрямую записать опять же значения в регистр. Делай четыре -это do nothing у тебя?
Короче затупил с этим примером)

★ DI HALT

13 Август 2014 в 2:54

1. Да, само собой. Где то мы их должны заполнить данными. Сохранив в память.
2. Регистры слишком ценный ресурс чтобы держать там такую фигню.
3. Ну таписано же зачем. Чтобы не тупить в задержках, а прощелкать его мимо, если время не вышло и не ждать. А как распологать дело десятое. Можно так. можно эдак. Я не заморачивался.
4. Делай 4 там нету. Так что да, do nothing там дальше должно быть «делай четыре»
Данные пишем в память, чтобы не занимать регистры. Чтобы в do one, two и так далее мы могли использовать весь регистровый набор и не париться. Получается этот кусок кода как бы независим ни от кого. Взял данные из памяти, отработал, положил обратно в память, регистры освободил.

---

and_master

14 Август 2014 в 0:12

Приветствую! Еще немного об этом примере.
1. Величина задержки находится в определениях lowbyte, midlebyte и highbyte. То есть до входа в главгый цикл эти значения надо скинуть в counter1,2?
2. По логике кода регистры разве не нужно куда-либо сохранять, в стек например, перед записью в них счетчика?
3. То есть , если в других участках кода регистры r16,r17,r18 не используются, то оперативку для счетчика можно не использовать?

★ DI HALT

14 Август 2014 в 0:23

1. Да
2. Если они где то ранее использованы то да. Но обычно, уходя из процедуры, подметаешь за собой и в регистрах уже нет ничего важного или критичного. Другое дело прерывания, там совсем другой расклад.
3. Да.

Bredov-IV

26 Декабрь 2014 в 12:07

Здравствуйте, вопрос мелко-теоретический:
У арифметических флагов какая дальнобойность? Пока проц не наткнётся на следующую арифметическую команду?
Я-то раньше считал, что флаги C и Z сбрасываются сразу же после следующей ЛЮБОЙ команды, а у DI написано:

```
CP R16,R17
BREQ action_b
BRCS action_b
```

Вот уж не думал, что Carry будет ещё жив, к моменту выполнения BRCS.

Wan-Derer

26 Декабрь 2014 в 12:22

А что по этому поводу говорит эмулятор в AVR Studio? :)

Bredov-IV

26 Декабрь 2014 в 13:29

Да как-то не обращал внимания. Ну установился флаг, ну произошёл условный переход, а дальше внимание ушло в другое место.
А вообще, действительно, надо было проверить.

★ DI HALT

26 Декабрь 2014 в 13:04

Флаг живет до тех пор, пока его кто-нибудь не изменит. Т.е. до следующей команды меняющей флаг. Причем конкретно этот флаг. Какая команда какой флаг меняет смотри в даташите.

Bredov-IV

26 Декабрь 2014 в 13:32

Спасибо, значит, жить станет чуточку легче :)
Ну, а справку по командам я, кроме как в даташите, нигде больше и не смотрю теперь.

---

viphorizon

23 Апрель 2016 в 12:31

CLC — это лишний такт, вроде, ROL сам должен позаботиться о флаге или я ошибаюсь?