AVR. Учебный курс. Флаги и условные переходы

AVR. Учебный курс | 8 Июль 2008 | DI HALT | 42 Comments

Есть в AVR (да и, пожалуй, во всех остальных процессорах) особый регистр **SREG**. О нем я несколько раз упоминал в прошлых статьях, но не вдавался в подробности. Чтож, пришло время рассказать, что же это же **SREG** такой и зачем он нужен.

SREG это регистр состояния ядра. Он так называется **S**tatus **Reg**ister. В этом регистре находится независимых битов — флажков. Которые могут быть либо 1 либо 0, в зависимости от выполненных в прошлом операций.

И вот по тому какие флаги стоят, можно понять что произошло с процессором и что нам дальше делать.

Например, если флаг **Z** (Zero) выставлен в 1, значит в ходе вычисления предыдущей математической операции в результате образовался ноль.

А если выставлен флаг **C** (Carry — заем, перенос), то мы из меньшего числа отняли большее, или же прибавили такое число, что результат стал больше 255.

А теперь подробней по каждому флагу.

- **I** — флаг разрешения прерываний. Когда установлен в 1 — прерывания разрешены.
- **T** — пользовательский флаг. Можно юзать по своему назначению.
  Кроме того, есть две команды которые позволяют в этот бит записать любой бит любого из 32 регистров общего назначения R0-R31 (далее буду их звать РОН). Это команды BLD Rn,bit и BST Rn,bit

- **H** — флаг полупереноса. Это если произошел заем бита из старшей половины байта в младшую. То есть когда из числа 0001 0111 пытаются вычести 0000 1000, то происходит заем бита из более старшего разряда, так как младшая тетрада уменьшаемого меньше чем младшей тетрады вычитаемого. Используется этот флаг в некоторых математических операциях.
- **S** — флаг знака. 1 — значит минус. При вычислении чисел со знаком он возникает если после арифметической операции возник отрицательный результат. Флаг S = V XOR N.
- **V** — Флаг переполнения дополнительного кода. Это если мы считаем число в дополнительном коде со знаком и оно вылезло за пределы регистра.
  Число в дополнительном коде с заемом — самая естественная форма представления числа во вселенной! Вот возьмем, например, число -61 как его получить? Ну не знаем мы про отрицательные числа! Просто! Вычтем его из нуля 00 — 61 = 39 Во как! Заняли из старшего разряда! Не похоже, да? Хорошо, проверим столбиком:

61
+
39
—
00 А единичка не влезла в разрядность!

ЧТД!!! (с) Лохов С.П. (Наш преподаватель по ассемблеру в универе)

Вот двоичный доп код работает точно по такому же принципу. А в процессоре есть команды для перевода числа из в доп код за одну команду.

- **N** — флаг отрицательного значения. Если в результате арифметической операции 7 бит результата стал 1, то этот флаг тоже станет 1.
- **Z** — флаг нуля. Если в результате какой либо операции получился ноль, то вылазит этот флаг. Чертовски удобно для всех целей!
- **C** — флаг переноса. Если в результате операции произошел выход за границы байта, то выставляется этот флаг. Вообще самый юзаемый флаг после Z и I. Что только с ним не творят, как только не юзают.

Флаги, кроме автоматической установки, можно устанавливать и сбрасывать вручную. Для этого есть команды

SE* для установки и CL* для сброса. Вместо звездочки подставляется нужный флаг, например, CLI — запрет прерываний.

В даташите, в разделе Instruction Set Summary, написано какая команда что делает и на какие флаги влияет.

Пример:

```
1     INC     Rd      Increment                   Rd = Rd + 1           Z,N,V
2     DEC     Rd      Decrement                   Rd = Rd – 1           Z,N,V
3     TST     Rd      Test for Zero or Minus      Rd = Rd AND Rd        Z,N,V
```

4	CLR	Rd	Clear Register	Rd = Rd XOR Rd	Z,N,V
5	SER	Rd	Set Register	Rd = $FF	None

Команда INC прибавляет к регистру 1, но несмотря на то, что она может добить регистр до переполнения, флаг переполнения C она не ставит. Такая особенность.

Зато она ставит флаги нуля, отрицательного значения и переполнения доп кода. Как инкремент может дать нуль? Элементарно, прибавив к -1 +1. Минус 1 в двоичной знаковой арифметике = FF. FF+1=1 00 ,но 1 не влезла в разрядность, поэтому 00 Логично же? :)

Или хотим мы, например, узнать в каком регистре число больше в R17 или R18

Нет ничего проще — к нашим услугам команда CP (нет, не детское порно, а Compare). Эта команда, у себя в уме, из R17 вычитает R18 при этом содержимое регистров не меняется, а меняются только флаги. И если, например, выскочил флаг C, значит при R17-R18 произошел заем, а значит R18 больше R17. Если флаг C не появился, то значит R17 больше чем R18. А коли у нас выскочил нуль, то значения равны. Есть еще команда CPI Rn,### работает также, но сравнивает регистр (только старшую группу) с произвольным числом. Используется чаще.

1	CP R17,R18

А потом смотрим флаги. И дальше в ход вступают команды условных переходов из группы BRANCH (ветвление). BR**

И вот в этом месте товарищей из ATMEL надо хватать за ноги и бить головой об стену. Потому что я не понимаю на кой хрен было так все запутывать, изобретая команды которых реально нет?

Суди сам. Флагов у нас 8, соответственно должно быть 16 возможных бранчей. 8 по условию — флаг есть, 8 по условию — флага нет. Бранчевых команд же у нас аж 20 штук.

```
BRBC #          переход если бит # в регистре SREG=0
BRBS #          переход если бит # в регистре SREG=1

BRCS            переход если C=1
BRCC            переход если C=0

BREQ            переход если Z=1
BRNE            переход если Z=0

BRSH            переход если C=0
BRLO            переход если C=1

BRMI            переход если N=1
BRPL            переход если N=0

BRGE            переход если S=0
BRLT            переход если S=1

BRHC            переход если H=0
BRHS            переход если H=1

BRTC            переход если T=0
BRTS            переход если T=1

BRVS            переход если V=1
BRVC            переход если V=0

BRID            переход если I=0
BRIE            переход если I=1
```

Однако, если глубоко копнуть, поглядеть на реальные коды команд, то окажется, что BRCS=BRLO и BRCC=BRSH — у них вообще одинаковый код.

А таких команд как BRBS # и BRBC # вообще не существует. Это всего лишь иносказательная запись всех остальных бранчей, в зависимости от номера бита #.

А таких команд синонимов там дофига :)
Так что гордое «131 Powerful Instructions – Most Single-clock Cycle Execution» на самом деле является не более чем гнилым маркетинговым высером. Нет там 131 команды! Есть 131 мнемоника, а это несколько разные вещи. Потому как ты помидор не обзови — картошкой он от этого не станет.

Правда, возможным оправданием такого поведения может служить то, что, дескать, так удобней — в зависимости от ситуации подставлять ту мнемонику которая написана более логично.

Может быть, но как по мне — только ситуацию запутывают. Из-за этого я после ассемблера C51 долго плевался на систему команд AVR, потом привык и ничо так, вкатило.

Итак, как работает любой из бранчей (да, тому кто придумал переименовать родимый J** в BR** я тоже ежедневно посылаю лучи поноса, надеюсь в сортире он себе уже кабинет обустроил).

Проверяется условие и если оно верное делается переход.

Например, на входе у нас значение.

- Если значение = 1, то делаем действие А
- Если значение = 2, то делаем действие Б
- А если значение меньше 13, то делаем действие ЦЭ

- Во всех остальных случаях не делаем ничего

Нет ничего проще, пусть значение приходит через регистр R16

```
1                   CPI     R16,1           ; Сравниваем R16 с 1
2                   BREQ    ActionA         ; Переход если равно (EQual, Z=1)
3                                           ; Если не равно, то идем дальше
4
5                   CPI     R16,2           ; Сравниваем R16 с 2
6                   BREQ    ActionB         ; Переход если равно
7                                           ; Если не равно, то идем дальше
8
9                   CPI     R16,13          ; Сравниваем R16. т.е. R16-13
10                  BRCS    ActionC         ; Если возник перенос, значит R16 меньше 13
11                                          ; Если не возник - идем дальше
12                  RJMP    NoAction        ; Переход на выход
13
14  ActionA:        NOP
15                  NOP                     ; Делаем наш экшн
16                  NOP
17                  RJMP    NoAction        ; Выходим, чтобы не влезть в ActionB
18
19
20  ActionB:        NOP
21                  NOP                     ; Делаем наш экшн
22                  NOP
23                  RJMP    NoAction        ; Выходим, чтобы не влезть в ActionC
24
25
```

```
26  ActionC:        NOP
27                  NOP                     ; Делаем наш экшн
28                  NOP
29
30  NoAction:       NOP
31
32
33  ; Вместо NOP, разумеется, должны быть какие-нибудь полезные команды.
```

Сложно? Вот и я думаю, что делать нефиг :)))))) Ничуть не сложней чем на Си забабахать из if then конструкцию или switch-case какой-нибудь.

Бег по граблям

У команд BR*** есть правда весьма ощутимое западло. Дальнобойность их составляет всего 63 команды. Т.е. это относительный переход. Поначалу ты этого не заметишь — короткая программа обычно не допускает переходов дальше 63.

Но вот, со временем, твоя программа распухнет, появится дофига кода и дальности бранча перестанет хватать. Вроде бы все работало, а добавил пару команд — и компилятор начал ругаться злыми словами — Error out of range или что то в этом духе. Как быть?

Перехватываться через промежуточные переходы. Т.е. находим ближайший к нам безусловный переход за который контроллер, как бы не старался залезть не сможет.

```
1   ActionA:        NOP
2                   NOP
3                   NOP
4                   RJMP    NoAction
5
6                   NOP             ; То есть если я вот тут поставлю островок
```

```
                                ; то он никогда не выполнится. Т.к. сверху
                                ; Процессор перепрыгнет сразу по RJMP
                                ; А снизу вход будет сразу же на ActionB

ActionB:        NOP
                NOP
                NOP
                RJMP    NoAction


И вот, в этом островке, мы создаем капсулу телепортер такого вида:


ActionA:        NOP
                NOP
                NOP
                RJMP    NoAction

;-----------------------------------
Near:           JMP     FarFar_away
;-----------------------------------


ActionB:        NOP
                NOP
                NOP
                RJMP    NoAction
```

Т.е. бранчу теперь достаточно дострелить до островка Near, а там дальнобойный JMP зашлет его хоть в бутсектор на другой конец флеша.

В случае большой программы, чтобы не плодить островки, его лучше сделать один, сразу после пачки CP.

Test & Skip
Кроме Branch‘ей есть еще один тип команд: проверка — пропуск.
Работает она по принципу “Проверяем условие, если справедливо — пропускаем следующую команду”
Запомнить просто, первая буква это Skip — пропуск. Вторая условие — C=Clear, т.е. 0 S=Set, т.е. 1. Соответственно S**C пропуск если сброшено. S**S пропуск если установлено.

Команды SBRC/SBRS

Указываешь ей какой РОН, и какой номер бита в этом регистре надо проверить. И если условие верное, то следующая команда будет пропущена.

Пример:

```
1       SBRC    R16,3   ; Если бит 3 в регистре R16 = 0, то прыжок через команду, на NOP
2
3       RJMP    bit_3_of_R16_Not_Zer0
4
5       NOP
6
7
8       SBRS    R16,3   ; Если бит 3 в регистре R16 = 1, то прыжок через команду, на NOP
9
10      RJMP    bit_3_of_R16_Zer0
11
12      NOP
```

Аналогичная команда

SBIC/SBIS Но проверяет она не биты регистров РОН, а биты регистров периферийных устройств. Те самые,что идут в памяти между блоком РОН и оперативкой. Но есть у ней ограничение — она может проверить только первые 1F регистров ввода вывода. Если заглянешь в m16def.inc (для мега16, для других МК в их файл дефайнов)

То увидишь там это:

```
.equ    UBRRH   = 0x20
.equ    UCSRC   = 0x20
.equ    EEARL   = 0x1e <-- а выше уже хрен :(
.equ    EEARH   = 0x1f <-- до сих можно обращаться через эти команды
.equ    EEDR    = 0x1d <-- ну и до нуля вниз.
.equ    EECR    = 0x1c
.equ    PORTA   = 0x1b
.equ    DDRA    = 0x1a
.equ    PINA    = 0x19
.equ    PORTB   = 0x18
.equ    DDRB    = 0x17
.equ    PINB    = 0x16
.equ    PORTC   = 0x15
.equ    DDRC    = 0x14
.equ    PINC    = 0x13
.equ    PORTD   = 0x12
.equ    DDRD    = 0x11
.equ    PIND    = 0x10
```

```
19  .equ    SPDR    = 0x0f
20  .equ    SPSR    = 0x0e
21  .equ    SPCR    = 0x0d
22  .equ    UDR     = 0x0c
23  .equ    UCSRA   = 0x0b
24  .equ    UCSRB   = 0x0a
25  .equ    UBRRL   = 0x09
26  .equ    ACSR    = 0x08
27  .equ    ADMUX   = 0x07
28  .equ    ADCSRA  = 0x06
29  .equ    ADCH    = 0x05
30  .equ    ADCL    = 0x04
31  .equ    TWDR    = 0x03
32  .equ    TWAR    = 0x02
33  .equ    TWSR    = 0x01
34  .equ    TWBR    = 0x00
```

Вот все это богатство твое. Проверяй не хочу :))))

А мне мало! Я хочу дальше!!! Что делать???
А дальше жизнь наша трудна и опасна. Чуть вправо,чуть влево — взрыв и ошметки. Страшно?

На самом деле ничего такого. Просто нам придется заюзать битмаски. И будет это не в одну команду, а в три.

Например, надо нам проверить состояние третьего бита в регистре UCSRC. Как видишь, это выше чем 1F Так что тест-скип не прокатит. Не беда, делай как я!

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

```
        IN      R16, UCSRC      ; Хватаем байт из регистра, целиком.
        ANDI    R16,1<<3        ; Накладываем на него маску 00001000
                                ; В результате, все байты кроме третьего
                                ; Обнулятся. А третий каким был таким и
                                ; останется. Был нулем - будет нуль и будет Z
                                ; Не был нулем, не будет Z флага. =))))

        BREQ    Bit_3_of_R16_Equal_Zero
```

Говорил же в три команды уложимся =)

А можно проверять и не по одному биту сразу. Накладывай нужные маски, оставляй только нужные биты. А дальше смотри какие числа эти биты могут образовать и сравнивай напрямую с этими числами через CPI. Красота!!!

Ладно, к битмаскам мы еще не раз вернемся. Это мощнейшее средство, но пока, не влезая в периферию, его не прочуешь. А до периферии мы еще не до росли. Всему свое время.

В следующей статье я тебе буду взрывать мозг индексными табличными переходами. Сходи лучше воздухом подыши предварительно. Такие вещи надо раскуривать со свежими мозгами

Assembler AVR Программирование Флаги

42 thoughts on “AVR. Учебный курс. Флаги и условные переходы”

Xenomorph

15 Октябрь 2008 в 2:09

Начал изучать прерывания опять же на C. И вот какое дело, обработчик прерываний у меня срабатывает когда на линии PB5 меняется уровень, в обработчеке у меня простой код если PB5 = 0 и PB6= 0 то пишу ноль, исли PB6=1 пишу еденицу. Так вот за 100 ms
у меня проходит 100 едениц и нулей. Суть в чем если я просто посылаю по UART, то принимаю все биты. Но стоит мне записать биты в массив, а потом перевести их в число, как на выходе я получаю какую то биллиберду. Я вот думаю успевает ли выполнится обработчик прерывания, до возникновения другого прерывания? Или может я что не так делаю???

Тьфу ты не туда написал, как удалить не знаю ((

★ DI HALT

15 Октябрь 2008 в 2:13

Посчитай такты за который выполняется обработчик прерывания. Прикинь время на выполнение. С учетом кварца. Потом позырь на то с какой скоростью у тебя прерывания вызваются — нет ли затыка.

Xenomorph

15 Октябрь 2008 в 13:57

Что то совсем у меня худо с прерываниями. Такое ощущений что программа постоянно перезапускается. Вот допустим в главной функции если после всей инициализации поставить UDR = ‘S’, при включении у меня, валится это S всех щелей. Что такое немогу ни как разобрался, отключаю прерывания, все нормально работает. Но вроде обработчик не вызывается до тех пор пока уровень не изменится на ножке PB5. Чудеса да и только)

---

milvus

1 Ноябрь 2009 в 14:49

чего-то я не пойму, флаг S и N одно и то же? ведь оба знак показывают
S это что главный флаг знака, а V и N частные случаи?
как вообще определить отрицательность числа по двоичноми коду, ведь может быть положительное число с 1 в 7 бите

★ DI HALT

2 Ноябрь 2009 в 12:08

По одним только флагам никогда ничего сказать определенно нельзя.

Только в контексте предыдущей выполненной команды. А вот уже она выставляет флаги в зависимости от результатов операции.

Знак у двоичного числа тоже никак нельзя определить в отрыве от контекста.

---

fistaka

14 Апрель 2010 в 0:02

Читал статью — возник вопрос:
«BRMI переход если N=1
BRPL переход если N=1»
— как это прокоментируешь???
p.s. спасибо за сайт!!!

★ DI HALT

14 Апрель 2010 в 0:21

Опечатка была. BRPL N=0

---

drs

5 Ноябрь 2010 в 20:56

Опечатка похоже

BREQ ActionA ; Переход если равно (EQual, Z=0)

Они равны если Z=1 ведь вроде? т.е. когда из одного другое вычитаешь получаешь ноль, и Z устанавливается в ноль (Zero).

★ DI HALT

5 Ноябрь 2010 в 23:35

Ошибка. Исправил.

---

randrew

10 Ноябрь 2010 в 18:16

Списибо. Только вот никак понять не могу, что же atmel имели в виду, когда в справке для команд ветвления написали описания типа «if(Z==1) PC = PC + k + 1». После такого логично предположить, что написав «BRCS 1» я увеличу PC на 2 в случае, если флаг поднят? Однако в эмуляторе в PC заносится 1.

---

salpingot

18 Март 2011 в 14:22

SBIC/SBIS Но проверяет она не биты регистров РОН, а биты регистров периферийных устройств. Те самые,что идут в памяти между блоком РОН и оперативкой. Но есть у ней ограничение — она может проверить только первые 1F регистров ввода вывода. Если заглянешь в m16def.inc (для мега16, для других МК в их файл дефайнов).
Я смотрел в справочнике Микроконтроллеры AVR семейства Mega.djvu (http://narod.ru/disk/7773184001/Микроконтроллеры%20AVR%20семейства%20Mega.djvu.html) стр.237 SBIC/SBIS, ограничения там к сожалению не указаны. Не подскажете как называется Ваш справочник где указаны ограничения в командах? (по возможности где можно скачать).

★ DI HALT

18 Март 2011 в 14:34

Прочитал где то в даташите. А вообще это понятно из двоичного кода команды, взятого в Евтсифееве из приложения. Там битами XXXX обозначены адреса, так что там всего пять битов адреса приходится на адрес порта, из этого следует, что адрес может быть от 00000 до 11111 т.е. от 0 до 1F

Arseniy Muradov

30 Январь 2012 в 14:31

DI HALT, можешь подсказать?
У меня в программе 2 прерывания(2 таймера)
программа крутиться в основном цикле из условных переходов BR и SBIC
Так вот с BR проблем нет, а вот с sbis — непонятки.
Вот кусок кода, вроде ни чего такого страшного, просто считываются биты с портов ввода-вывода(внешне подтянутые через 10к к земле, внутренняя подтяжка отключена ну и кнопка обычная на плюс)

```
sbic pinc, 5 ;условный переход «кнопка режима»
jmp mode
sbic pinc, 4 ;условный переход «войти в настройки»
jmp settings
sbic pinc, 3 ;усл.пер. «спать»
jmp sleep_on
```

так вот почему-то у меня срабатывают (редко 1-2 раза в день) эти переходы и программа перебрасывается на выполнение. Ставил светодиоды дополнительные и заменял jmp на sbi, что бы быть уверенным что программа именно от туда выпрыгивает, так вот они загораются. то есть почему-то идет ложное считывание.
Что может быть?
Условия разные пробовал, питал от лаб. ИП, USB, даже в воздухе подвешивал и чуть ли не медным тазом накрывал.

★ DI HALT

30 Январь 2012 в 17:52

А точно ложное считывание? может что реально какая помеха дрыгает ногой? Если глюк периодически, то может его генерирует твоя же плата. Попробуй частоту понизить радикально и посмотреть что будет.

Arseniy Muradov

1 Февраль 2012 в 0:25

Вчера ночью отключил прерывание одного из таймеров, уже 24 часа ни какого сбоя и ложного срабатывания. то есть работает как надо

Но это ведь странно? такие переходы ведь не зависят ни от флагов(хотя я их сохраняю перед прерыванием), ни от чего вообще не зависят кроме конкретно указанного бита. да? Или есть какие-то заковырки?

Щас попробую вернуть прерывания, изменить частоту(была 8 внутрениия), ножки четко на землю посажу без всяких резюков, что б уж наверняка не колыхалась.

★ DI HALT

1 Февраль 2012 в 1:56

А атомарного доступа нигде нет? Т.е. нигде не получается, что прерывание вклинивается в обработку двубайтной переменной между первым и вторым байтом, а потом, в результате, заносится не то. А регистры сохраняешь которые в прерывании используются?

Arseniy Muradov

1 Февраль 2012 в 7:48

да не(

Нет ни чего такого…. в цикле у меня только условные переходы и ни какие регистры кроме флагового не используются, но его я

сохраняю. С от него зависимыми br переходами собственно проблем и не было…

Arseniy Muradov

2 Февраль 2012 в 14:33

>>>>>>

Эмм… так вот, я изменил частоту на 4МГц и намертво повесил все кнопки на землю, за 24 часа проблем не было.
Далее изменил частоту на 8МГц, кнопки так и остались висеть на земле, ложно сработали ВСЕ sbic переходы! :(
Вообще не понимаю, изменил только частоту :(

Arseniy Muradov

2 Февраль 2012 в 14:34

ах да, оставил на ночь, утром проснулся и обнаружил что они сработали, то есть где-то за 6 часов…

★ DI HALT

2 Февраль 2012 в 15:36

Бред какой то. А если выкинуть весь остальной код и оставить только детект кнопок? Будет глючить?

Arseniy Muradov

2 Февраль 2012 в 15:44

я отключал один таймер, тупо закомментил его инициализацию и он не запустился само собой — ложных срабатываний не было.

То есть отключил одно из прерываний — переходы работали правильно…

Но блин, не пойму почему так… реально же бред :(

★ DI HALT

2 Февраль 2012 в 16:43

На что влияет прерывание:

На сохранение контекста, на флаги, на стек. Вот тут и копай.

Arseniy Muradov

3 Февраль 2012 в 1:14

да ну…

ну меняет оно флаги, но я их сохраняю, записывает иногда данные в озу, все регистры у прерывания свои, больше ни где не испозуются.
да и какая разница если эти переходы не зависят ни от чего этого.

в общем пичаль. перепишу прогу лучше как-нибудь подругому.

★ DI HALT

3 Февраль 2012 в 12:37

А то, что у тебя может быть срыв стека, например, и прога из прерывания возвращается не по тем адресам и может попасть, например, между условий. Переходы не глючат. У меня, например, никогда не было такого, чтобы sbic прощелкал когда не надо.

Опять же как ты их сохраняешь? Какие регистры у прерывания свои?

Arseniy Muradov

3 Февраль 2012 в 16:24

Срыва нет, там стек попросу не используется внутри прерывания…
Ну РОН… щас точно сказать не могу, но там 3-4 средних и 1 низкий, используются они только в прерывании и нигде больше, во всяком случае мне так кажется, может я конечно чего не углядел после полной перечитки кода, но вряд ли…

флаги сохраняю в начале и загружаю в конце

★ DI HALT

3 Февраль 2012 в 16:42

Ну да, конечно. А как тогда возврат делается? Стек в прерываниях используется всегда. Покажи как флаги сохраняешь-загружаешь

Arseniy Muradov

3 Февраль 2012 в 18:04

Эм, я знаю что возврат через стек делается) я имел ввиду что в самой программе прерывания(т.е. до reti), я стек ни как не использую.

мб вообще весь код кинуть? куда только?

Valerii

14 Август 2013 в 11:06

В примере мелкая опечатка (мелочь, но так как это не просто текст, а пример, то может немного озадачить)

; В результате, все байты кроме третьего
; Обнулятся. А третий каким был таким и

Там, я так понял, имеются в виду биты.

Adler

11 Январь 2014 в 5:10

я бы сделал так :
http://easyelectronics.ru/repository.php?act=view&id=118

теперь этот момент » У команд BR*** есть правда весьма ощутимое западло. Дальнобойность их составляет всего 63 » нам не важен ,так как переходим к «экшенам » по команде RJMP

★ DI HALT

11 Январь 2014 в 6:03

Тогда уж в макрос завернуть все. Получится длинный BR.

Adler

11 Январь 2014 в 11:25

да,и указывать в макросе только адресса
переходов или подпрограмм.(если я Вас правильно понял)
к примеру макрос ,который ожидает значение в пределах 0x30-0x3A
(аски коды цифр) на командах brlo и brsh

★ DI HALT

11 Январь 2014 в 11:58

Его, можно, кстати, попробовать условным сделать. Т.е. если шаг укладывается в пределах, то использовать br** если нет, то через проброс сквозь Rjmp. Но я не знаю хватит ли макропроцессору на это мозгов, он тут очень ограниченный. Условия поддерживает, но сможет ли сам вычислить смещение?

Adler

11 Январь 2014 в 12:31

Вы имеете ввиду ,проверка условия и сразу экшен ,?
(если так ,то в экшене может испортится полученное значение для проверки, нужно будет его сохранять)

если нетрудно напишите вашу мысль в коде(примерно)
а я проверю

★ DI HALT

11 Январь 2014 в 12:42

Ненен я имею ввиду сделать универсальный макрос на .if директивах и проверку адреса средствами препоцессора компилятора и если требуется длинный переход компилятор сам подставит вашу конструкцию, а если короткий, то воткнет просто br** почитайте справку по компилятору avrasm и описание директив компиляции (справка вызывается в авр студио по F1)

Adler

11 Январь 2014 в 13:42

теперь понял…надо с этими директивами разбираться ,там тоже много интересного

Myp3uK

21 Январь 2014 в 2:13

ANDI R16,1<<3
Можно ли в этой строчке "1<<3" заменить на "0b00001000".
Да и как вообще преобразовать двоичное число в такой вид?

★ DI HALT

21 Январь 2014 в 3:02

Можно, но зачем? Так проще и потом можно вместо цифры писать будет имя бита. А суть записи в том, что мы 1 ставим на третий разряд. Это операция сдвига на языке препроцессора компилятора.

Myp3uK

23 Январь 2014 в 2:03

Спасибо. Теперь понял суть масок.

---

andryzil

30 Май 2014 в 14:27

DI HALT, можешь подсказать по флагу V ? Классически переполнение контролируется по не правильному знаку результата (когда результат вычисления вылазит за разрядную сетку). Но когда я например в АВР СТУДИО на АВРе складываю два отрицательных числа -240 + -50 = -290. Результат явно вылазит за разрядную сетку (т.к. в АВР при арифметических операциях используется диапазон представления чисел +255 — -256 отриц. числа в кодах дополнения до 2 без знакового разряда). При этом (по правилам контроля переполнения)знак результата должен быть + и выставляться флаг V. Реально в sreg устанавливается флаг N т.е. минус результата и флаг V не устанавливается???!!! Аналогичный пример вычитания чисел +240 — -100 = +390. При этом знак результата должен быть минус и выставляться флаг V. Но реально в sreg устанавливается только флаг половинного переноса, а результат + и флаг V не устанавливается???!!! Получается что АВР не контролирует переполнение при арифметических операциях???!!! Или я что-то не правильно понял? Уважаемый DI HALT, если я что-то не правильно понял можешь показать на конкретном примере как АВР контролирует переполнение при арифметических операциях(т.е. выставляется флаг V)?

И еще я заметил что AVR как-то хитро понимает отрицательные числа.Знакового разряда нет — под диапазон чисел отводятся все 8 бит РОН. Например сложить -15(0xF1) + -5(0xFB) = -20(0xEC) это все равно что сложить 241(0xF1) + 251(0xFB) = 236(0xEC) плюс перенос.В первом и во втором случае операнды, результат и флаги(S,N,V) абсолютно одинаковы!

---

and_master

8 Август 2014 в 0:44

«Например, надо нам проверить состояние третьего бита в регистре UCSRC. Как видишь, это выше чем 1F Так что тест-скип не прокатит. Не беда, делай как я!»
Почему не прокатит? Если записать значение порта в РОН, а потом применить SBRC/SBRS. Или я что-то не догнал)

★ DI HALT

8 Август 2014 в 1:18

Прокатит. Можно и так :)

---

neoneon

5 Январь 2015 в 21:30

Отлично написано — легко и понятно :)