AVR. Учебный курс. Ветвления на индексных переходах

AVR. Учебный курс | 8 Июль 2008 | DI HALT | 127 Comments

Таблицы переходов

Вот представь, что нам надо сделать мега CASE, когда на вход валится число от 1 до 100 и нам надо сто вариантов действий.

Как будешь делать? Если лепить сто штук CPI с последующими переходами, то можно дальше убиться головой об стену. У тебя только эти CPI/BR** сожрут половину памяти кристалла. Учитывая, что каждая CPI это два байта, а каждый BR** еще байт. А о том сколько тактов эта шняга будет выполняться я даже не упоминаю.

Делается это все круче. Помнишь я тебе рассказывал в прошлых уроках о таких командах как ICALL и IJMP. Нет, это не новомодная яблочная истерия, а индексный переход. Прикол в том, что переход (или вызов подпрограммы, не важно) осуществляется тут не по метке, а по адресу в регистре Z (о том что Z это пара R30:R31 я пожалуй больше напоминать не буду, пора бы запомнить).

Итак, вначале пишем дофига вариантов наших действий. Те самые сто путей, у меня будет не сотня, а всего пять, но это не важно.

```
1
2
3
4
5
```

```
Way0:   NOP
Way1:   NOP
Way2:   NOP
Way3:   NOP
Way4:   NOP
```

Дальше, где нибудь посреди кода мы херачим таблицу переходов.

```
1
```

```
Table:  .dw     Way0, Way1, Way2, Way3, Way4
```

Распологать ее можно где угодно. Хоть прямо тутже, хоть в конце кода. Главное, чтобы программа при исполнении не выполнила эту строку как команды, иначе будет ошибка. Для этого таблицы либо перепрыгивают с помощью команды RJMP

```
                RJMP    Kudato
; Программа сюда никогда не попадет.
Table:  .dw     Way0, Way1, Way2, Way3, Way4


Kudato:         NOP
                NOP
```

Или (предпочтительней) размещать в слепых тупиках кода, где выполнение не будет по алгоритму. Например между подпрограммами. Так:

```
                ...
                NOP
                RET     ; Точка выхода

; Программа сюда никогда не попадет
Table:  .dw     Way0, Way1, Way2, Way3, Way4

SomeProc:       NOP     ; Точка входа
                NOP
                ...
```

Таблица переходов это всего лишь строка данных в памяти, содержащая адреса Way0…Way4. Обратите внимание на то, что данные у нас двубайтные слова dw!!! Если же мы хотим адресовать расположенные данные, например вложенную таблицу адресов переходов, то адреса нужно умножать на два.

Вообще тут проще при написании по быстрому скомпилить кусок, открыть дамп с map файлом и посмотреть что лежит в памяти, куда что ссылается. Сразу станет понятно надо умножать на два или нет.

Допустим у нас данные появляются в регистре R20 и нам, на основании числа там, нужно выбрать по какому пути переходить. Регистр R21 временный, под всякую хрень.

А теперь делаем финт ушами.

```
1         LSL     R20             ; Сдвигом влево умножаем содержимое R20 на два.
2                                 ; Было, например, 011=3  стало 110=6 Круто да? ;)
3                                 ; опять же изза того, что у нас адреса двубайтные
4                                 ; НЕ ПУТАТЬ С ТЕМ ЧТО КОМПИЛЕР ОБСЧИТЫВАЕТ
5                                 ; ПАМЯТЬ В СЛОВАХ, тут несколько иное. Если непонятно
6                                 ; то в комменты пишите, обьясню.
7
8         LDI     ZL, low(Table*2)        ; Загружаем адрес нашей таблицы. Компилятор сам посчитает
9         LDI     ZH, High(Table*2)       ; Умножение и запишет уже результат.Старший и младший байты.
10
11        CLR     R21             ; Сбрасываем регистр R21 - нам нужен ноль.
12        ADD     ZL, R20         ; Складываем младший байт адреса. Если возникнет переполнение
13        ADC     ZH, R21         ; То вылезет флаг переноса. Вторая команда складывает
14                                ; с учетом переноса. Поскольку у нас R21=0, то по сути
15                                ; мы прибавляем только флаг переноса. Вопросы - в комменты!
16                                ; Таким образом складываются многобайтные числа.
17                                ; Позже освещу тему
18                                ; Математики на ассемблере.
19
20        LPM     R20,Z+          ; Загрузили в R20 адрес из таблицы
21        LPM     R21,Z           ; Старший и младший байт
```

```
22
23
24          MOVW     ZH:ZL,r21:r20          ; забросили адрес в Z
25
26
27      /* Что это было? А все просто! У нас наше число образовало смещение по таблице
28      переходов. Т.е. когда оно равно 0, то смещение тоже нулевое равное адресу Table,
29      а значит выбирается адрес ячейки где лежит адрес на Way0. Следом за ней в памяти сразу же
30      находится адрес Way1, Разница между ними два байта (так как сами адреса двубайтные).
31      Таким образом, если в R20 будет число 1, то команда LSL умножит его на 2 и будет указан на
32      адрес ячейки с Way1 и так далее.  А что же дальше? А дальше, конечно же: */
33
34          IJMP                       ; Обана и переход на наш выбранный Way. Понравилось?
35                                     ; То ли еще будет :)
```

А если хотите повзрывать себе мозг, то пофтыкайте в исходник который я писал для статьи про трояна в мобильный телефон на базе Атмеги. Там конвейер за конвейером. Текстовые строки превращаются в последовательности АТ команд, натыкивающих буквы на сотовом телефоне (Б, например, это два нажатия на двойку и так далее). И вся это бодяга в виде кучи адресных таблиц хранится в памяти. Короче, укур еще тот

Камрады тут напоминают, что данный метод может быть реализован далеко не во всех AVR. Да это так, дело в том, что система команд у разных AVR немного отличается, например некоторые младшие модели (вроде Tiny 11/12) не могут делать индексные переходы и вызовы. Так что тут надо смотреть внимательно систему команд на конкретный контроллер. Но обычно такая фича есть.

Также, можно объединять конструкцию из CPI/BREQ с таблицами переходов. Чтобы получать неравномерные таблицы, в которых значения идут не по порядку. Т.е. мы сначала вычленяем в коде закономерности, которые можно увязать в таблицы, а все что выпадает в виде исключений обрабатываем отдельными CPI/BR** конструкциями.

В результате получаем очень компактный и быстрый код. Который не сможет переплюнуть по оптимальности ни один оптимизирующий компилятор.

Командные конвейеры
Еще мощнейшим инструментом является конструкция под названием командный конвейер. Термин я сам придумал ;), так что не придирайтесь.

Идея в чем — у нас есть множество процедур, делающий какие либо односложные операции. Если, например, это ЧПУ станок то операции могут быть такие:

- Поднять резец
- Опустить резец
- Включить подачу
- Включить привод
- Подать на один шаг вперед
- Подать на один шаг назад
- Подать влево
- Подать вправо.

А нам надо выточить деталь. И для этого есть ТЗ, где сказано в какой последовательности эти операции должны быть выполнены, чтобы получить нужную деталь. Причем программа то у нас одна, а последовательности разные.

Не переписывать же код каждый раз. В этом случае можно сделать командный конвейер.

Команды у нас четко определены, так что мы их можем записать в виде процедур ассемблерных:

```
1
2
3
```

```
BladeUP:        NOP
                NOP
                NOP
```

```
4                       NOP
5                       RET
6
7       BladeDN:        NOP
8                       NOP
9                       NOP
10                      NOP
11                      RET
12
13      DriveON:        NOP
14                      NOP
15                      NOP
16                      NOP
17                      RET
18
19      DriveOFF:       NOP
20                      NOP
21                      NOP
22                      NOP
23                      RET
24
25      Forward:        NOP
26                      NOP
27                      NOP
28                      NOP
29                      RET
30
31      Back:           NOP
32                      NOP
```

```
33                  NOP
34                  NOP
35                  RET
```

И так далее, все нужные команды.

Затем, также как и в случае индексных переходов, создается таблица с адресами процедур.

```
1   Index:          .dw     BladeUP, BladeDN, DriveON, DriveOFF, Forward, Back
```

А в памяти ОЗУ заводится очередь задач. Обычный строковый массив нужной длины:

```
1   TaskList:       .byte   30
```

Потом создается процедура которая извне загружает нашу очередь действий, в виде последовательности кодов этих действий (всего лишь смещение по таблице! Очень быстрое и компактное!) и вызывается диспетчер очереди.

Диспетчер будет в цикле перебирать нашу строку TaskList, выковыривать из нее номера подпрограмм, по таблице, смещением, вычислять адреса переходов на реальные процедуры, переходить по ним, делать полезное действие, возвращаться обратно и брать новое значение из таблицы.

Чуете куда я клоню? Получается своеобразная виртуальная машина. Процессор в процессоре. Более того, сами задачи, вызываемые из очереди, не обязательно должны быть тупо выполняющими конкретное действие. Они же тоже могут набрасывать в эту самую очередь новых кодов, сортировать и на ходу их переделывать. Получается мощный полиморфный алгоритм, который может сам менять свою логику исходя из условий.

Позже, я на этом принципе покажу прообраз операционной системы. С диспетчером задач и кучей потоков, вызывающих друг друга по цепочке.

А еще на индексных переходах можно делать очень компактные конечные автоматы. Причем даже адрес не надо будет вычислять т.к. в виде текущего состояния можно смело использовать прямые адреса других состояний автомата.

Если не понял о чем идет речь, то не забивай пока голову. Дальше будет на это дело отдельная статья =) Или покури интернет на предмет того, что такое конечные автоматы и как они реализуются.

Assembler AVR Таблица переходов Трюки

127 thoughts on “AVR. Учебный курс. Ветвления на индексных переходах”

md

12 Август 2008 в 10:18

Для работы таблицы переходов в таком виде требуется команда LPM, т. к. команды ICALL и IJMP работают с адресным пространством данных, а таблица расположена в адресном пространстве кода. Кроме того, вместо ROL в данном случае лучше использовать LSL, т. к. фиг его знает, что там в переносе попадется. В таблице адреса умножать на 2 не надо, т. к. для меток в скции кода уже назначается адрес слова, а не байта.

Примерно так:
.cseg

table: .dw way0, way1, way2, way3, way4
...
ldi zl,low(table<<1)
ldi zh,high(table<<1)
lsl r20
clr r21
add zl,r20
adc zh,r21
lpm r20,z+ ;загрузка в пару r21:r20 слова из таблицы в памяти программ
lpm r21,z
movw zh:zl,r21:r20 ;Z = r21:r20
ijmp

Добавлю еще, что ICALL, IJMP и LPM r,Z+ и MOVW есть не во всех моделях МК (нет в самых простых либо старых — смотрим в даташите на конкретный МК или на предупреждения при компиляции).

★ DI HALT

12 Август 2008 в 12:22

Да, разуммется! :(Писалось в 4ре ночи, так что забыл важнейший момент — загрузку собственно адреса из таблицы :(Ступил,блин. Щас подправлю. Спасибо!

Кстати, если уж говорить о случайностях, то перед началом вычисления смещения по хорошему надо и флаг C скинуть.

---

http://maddev.livejournal.com/

12 Август 2008 в 12:33

Флаг скидывать не обязательно — LSL и ADD на него пофиг, они его только устанавливают.

★ DI HALT

12 Август 2008 в 12:46

Сейчас в АВРСтудии прогнал, чето у меня глюк какой то был — Z самопроизвольно увеличился на 1, после прохода команд add и adc, хотя смещение было равно нулю. Вот я и заподозрир ранее где то установленный флаг C. Сбросил его принудительно — все заработало. Сейчас закомментил CLC никаких проблем. Что это было хз :/

http://maddev.livejournal.com/

12 Август 2008 в 12:55

Да, всякое бывает :-)

Stebanoid

27 Сентябрь 2008 в 21:05

Раз уж код исправил, то и коментарии поправь. А то у тебя получилось, что в коде LSL, а в коментариях ROR

sinobi

12 Август 2008 в 21:19

Простите,что не по теме,просьба к модеру можно ли курс авр разбить по номерам-1й урок,2ой и так далее,а то распечатал и путаюсь в хронологии.Спасибо.

★ DI HALT

12 Август 2008 в 22:07

На самом деле тут четкой хронологии нет. Каждый урок законченный. Но если открыть весь курс, ткнув на соответствующий пункт рубрикатора, то записи будут в хронологическом порядке, начиная от последней к первой. Через какое то время, когда будет больше материала, я создам локальный документ (например в виде chm файла, где будет весь курс + дополнительные примеры и материалы, иллюстрации поясняющие и много чего еще) в виде электронной книги.

---

http://ximeg.myopenid.com/

7 Апрель 2010 в 21:11

Эн-не-не!!! Только не в виде ужасного chm!!!! Как я его в Linuxe читать буду? Через (_!_)? Лучше всего сделать курс в виде PDF-книжки, если уж на то пошло.

★ DI HALT

7 Апрель 2010 в 22:07

канеш пдф какой вопрос. Я же заебусь эту чмошку верстать. А пдф я в ворде склепаю в два счета.

---

sinobi

12 Август 2008 в 21:21

И еще вопрос, где можно почитать в инете ваши статьи поппрограммингу и мк
?

★ DI HALT

12 Август 2008 в 22:04

Больше нигде. Я только начал выкладывать.

Dghost

16 Сентябрь 2008 в 13:39

нихуя не понял…
если можно, поподробнее объясни про этот case

★ DI HALT

16 Сентябрь 2008 в 20:23

Короче. У нас есть массив-строка адресов переходов.

Значение, приходящее в Case = смещению по которому мы вычислим в какой ячейке массива находится нужный переход и загрузим его в идексную пару Z. После чего переходим на эту Z

Dghost

16 Сентябрь 2008 в 21:20

это я как раз таки понял))
я не понял про смещение и умножение на 2
и команды add и adc
и что значит z+?

★ DI HALT

16 Сентябрь 2008 в 21:29

В памяти программ адресация двубайтная. Т.е. 0,2,4,6,8 и так далее.
Ячейка Case : Реальный Адрес ячейки в которой лежит адрес перехода.
0 : 0
1 : 2
2 : 4

3 : 6
4 : 8

У нас на вход case поступает число 1 2 3 4 (переход по ячейкам кейса)и так далее. Чтобы привести их к реальному адресу их мы умножаем на два. Т.е. пришел в кейс 3, чтобы его отправило по 3 ячейке нужно умножить на 2. На два проще всего умножить сдвигом.

Так как у нас таблица перехода располагается не в начале адресного пространства, а в жопе где нибудь (например с адреса 200), то адреса будут выглядеть примерно так:

1 : 202
2 : 204
3 : 206
4 : 208

Где 200 это база. Пришло в кейс 3. Вначале вычисляем смещение по базе 3*2 потом прибавляем к нему базу 6+200 (команда адд) получаем 206 — 3 ячейку кейса, где лежит адрес по которому мы должны уйти.

Ну а адд и аддс это просто сложение двухбайтных величин. Вначале складываем младшие байты, если вылезло переполнение, а вылазит оно в флаг C, то вторым аддс мы складываем старшие байты и прибавляем переполнение. Т.е. ничего не теряем.

★ DI HALT

16 Сентябрь 2008 в 21:31

z+ это просто команда такая. Т.е. мы вначале загрузили младший байт, при этом увеличился указатель Z на 1 (по аналогии с Си Z++) и получили уже старший байт.

---

dima_m

13 Июнь 2010 в 10:03

Получается что Z+ увеличивает указатель для следующей выполняемой строки, а не в той где она стоит.

1. LPM r16, Z+ ;увеличит указатель для строчки 2
2. LPM r17, Z+ ;увеличит указатель для строчки 3 и т.д.
3. LPM r18, Z ;а здесь не нужно увеличивать, потому что строка последняя.

rjmp start

Dghost

16 Сентябрь 2008 в 23:24

спасибо, все проясняется…
я догадывался что z+ это инкремент, но подумал, что команда инкремента уже есть, значит это что то другое
а не мог бы ты также на примере с числами расписать про сложение двубайтных велечин с переполнением и без?

★ DI HALT

16 Сентябрь 2008 в 23:26

Ок. поставлю в очередь постов. Скоро будет :)

★ DI HALT

16 Сентябрь 2008 в 23:39

Это не просто инкремент. Инкремент командой это икремент любого регистра.

А Z+ это инкремент именно в команде LPM (и еще в некоторых). В других случаях не прокатит.

Dghost

16 Сентябрь 2008 в 23:27

и еще смутно себе представляю сдвиг.
т.е lsl 11111110 — 11111101 (Так?)
lsl 00000001 — 00000010
lsl ….. — ……
lsl 10000000 — ?

★ DI HALT

16 Сентябрь 2008 в 23:36

1 вылезет в флаг [C]

то есть структура такая:

[C]<-[регистр]<-0
или, если сдвиг направо,
0->[регистр]->[C]

после флага C биты теряются

★ DI HALT

16 Сентябрь 2008 в 23:37

Прогоняя байт сдвигом, ты побитно сканируешь его через флаг C. Иногда это жутко удобно.

---

hexFF

7 Май 2009 в 13:28

даже счетчика принятых бит не надо, просто приняв первый бит записываем его в C и смотрим, если это 0- то выставляем регистр, который будем сдвигать (0xff, прям как у меня), а если 1- обнуляем. Дальше, после каждого принятого бита выполняем сдвиг и смотрим последний бит регистра, как только он изменит свое значение- всё, пора переваривать принятое.

На днях присал адаптер iWire, придумал ;-)

★ DI HALT

7 Май 2009 в 14:13

А если не изменит? Если принятый бит будет 0, а там и был ноль?

Dghost

17 Сентябрь 2008 в 11:16

о понял! т.е. если сдвиг влево то, как бы справа входит 0, а самый левый бит уходит в C , так?
А что значит «сканировать через флаг C»? Все время смотреть, что в C?

★ DI HALT

17 Сентябрь 2008 в 11:42

А это, надо, например, тебе байт превратить в последовательность битов. Например, чтобы послать по последовательной шине. Ты загружаешь байт в регистр, сдвигаешь его через C, а то что попадает в C отправляешь куда нибудь.
Таким же образом можно принимать байт побитно. Т.е. следишь за ножкой порта если она 1, то мы ставим C в 1 и сдвигаем байт. Если 0, то мы ставим C в 0 и сдвигаем байт (только там уже другая команда — сдвиг через C) в результате, через восемь сдвигов у тебя в регистре соберется байт который пришел по одной линии.

WPanda

12 Октябрь 2008 в 2:07

А вот если без ijmp… я в свое время на 8080 (да, я мастодонт ;)) делал вот как — вычислял адрес перехода, загонял его пушами в стек и делал ret:

```
case: ;пусть в R16 у нас номер кейса

;загружаем начало таблицы переходов
ldi zh ,high(qqqq)
ldi zl ,low(qqqq)

;вычисляем адрес вектора. На самом деле нельзя просто так
;прибавлять к zl, надо вообще грамотно складывать-то.. но лень

add zl, r16

;раз уж нельзя писАть в $PC напрямую — загоняем в стек

push zl
push zh

;и прыгаем

ret

;а тут уже пишем таблицу векторов
qqqq:
rjmp qqqq_1
rjmp qqqq_2
rjmp qqqq_3
```

qqqq_1:

nop
nop
nop

qqqq_2:

nop
nop
nop

qqqq_3:

nop
nop
nop

Иногда бывает очень полезно.

А на tiny точно нельзя напрямую в $PC писАть?

★ DI HALT

12 Октябрь 2008 в 2:45

Кстати да!!! Совсем забыл. Офигенный метод! Кстати, он и попроще будет :)))) Не обязательно юзать индексные регистры, достаточно сунуть любое дерьмо в стек.

Wan-Derer

21 Октябрь 2013 в 9:23

А на Z80 уже были индексные регистры, правда уже не помню можно ли было по ним делать переходы или они были только для адресации данных. Но мне тогда в голову не пришла идея такого элегантного CASE (учил ASM сам и толковых книжек что-то в руки не попалось), и я делал опрос клавы последовательными проверками. К счастью, писал я игрушку, поэтому опросить надо было всего-то десяток кноп :)

Art

14 Октябрь 2008 в 15:28

А зачем table умножаем на 2? Разве table не является базовым адресом таблицы переходов?

★ DI HALT

14 Октябрь 2008 в 17:11

По факту это так, но тут есть одна тонкость. Все дело в разнице представления.

Дело в том, что тут же память программ, а значит адресация заточена под переходы — а они адресуются в словах (т.к. минимальная длина команды 2 байта и незачем увеличивать число смещения и тратить дополнительные биты для расширения адресного диапазона если можно взять и адресовать сразу двубайтные слова). Соответственно адрес метки это адрес в словах. А команды работы с данными в памяти оперируют адресами байтов. Вот и выходит, что байтный адрес вдвое больше адреса словесного.

выглядит это так — справа адрес в байтах, слева в словах. X посредине это ячейка памяти

00-x-00
—x-01
01-x-02
—x-03

02-x-04
—x-05
03-x-06

Понтяно?

Art

23 Октябрь 2008 в 2:55

вроде бы да :)
то есть table — это адрес в байтах из которого нужно получить адрес в словах?

---

testicq

3 Февраль 2009 в 3:34

косячекс:
«Поскольку поскольку стандартных механизмов вроде If-Then-Else в ассемблере нет…»
Лишнее «поскольку «

---

testicq

10 Февраль 2009 в 18:28

Хочу сохранить байт из регистра R25 в память данных (.DSEG) по адресу Command и смещению в регистре R16. Можно ли сделать так:

```
LDI     ZL,Low(Command*2)       ; Заносим младший байт адреса, в регистровую пару Z
LDI     ZH,High(Command*2)      ; Заносим старший байт адреса, в регистровую пару Z
ADD     ZL, R16                 ; Добавляем смещение
CLR     R16
```

```
ADC     ZH, R16          ; Учитываем возможный перенос
ST      Z, R25           ; Сохраняем байт
```

просто пишу обработчик прерывания и хочу сэкономить регистры, чтобы не тратить на них стек и машинное время. Посмотрел по даташиту команда CLR Rd (она же EOR Rd, Rd) не меняет флаг переноса, который учитывается командой ADC Rd, Rr.
И на 2 наверно не надо домножать…

---

http://maddev.livejournal.com/

10 Февраль 2009 в 18:40

Все правильно, только умножать адрес на 2 не надо — в SRAM (.dseg) и EEPROM (.eseg) с метками связывается адреса байтов, а не слов.

---

bdp

11 Июнь 2009 в 17:20

Застрял:
Никак не могу разобратся с флагами, и так:
ldi r16, 254
ldi r17, 2
cp r16, r17
Результат положительний однозначно, но флаги «S» «N» = 1 типа результат отрицательное число. Что не так, кроме моих мозгов :) В даташитах влияние виполнения команди на флаги расписано очень детально, но на непонятном мне язике, если есть возможность хотябы направте в правильном направлении, особенно об этом непоняном языке (дефаултный путь мне известен).

---

http://maddev.livejournal.com/

11 Июнь 2009 в 17:48

Сравнение — это вычитание:
254 — 2 = 252 = 0b11111100

Смотрим в мануал по системе команд:
N: R7 — т. е. N равно седьмому биту результата (он у нас 1)
V: Rd7 • Rd7 • R7 + Rd7 • Rr7 • R7 = 0
S: N [+] V = 1

+ — ИЛИ
• — И
[+] — исключающее ИЛИ

Получили N = 1, S = 1.

> типа результат отрицательное число.
Если рассмотреть операнды как числа со знаком, получаем:
-2 — 2 = -4
Флаги не врут — результат действительно отрицательный! Дело в том, что для операций сравнения важно оговорить, знаковыми или беззнаковыми являются операнды и использовать соответствующую группу команд условного перехода: brlt, brge — для знаковых; brlo, brsh — для беззнаковых. Разница — в проверяемых флагах.

---

bdp

11 Июнь 2009 в 18:03

Большое СПАСИБО, за исчерпывающий ответ.

---

http://maddev.livejournal.com/

11 Июнь 2009 в 17:59

Более того, нет необходимости знать наизусть, какие флаги устанавливает команда сравнения: мнемоника команд условного перехода достаточно хорошо абстрагирует флаги до знакомых человеческих понятий «больше», «меньше», «ниже», «плюс» и т. д.

---

bdp

11 Июнь 2009 в 18:17

В чем то Вы правы, но без глубоких базовых знаний потом в такую задницу можна влезть. А если разобрается сначала и нормально, то както увереней потом себя чувствую, типа контроль на работой чтоли.

ReArt

24 Июль 2009 в 0:46

Вопрос: а если сохранять в памяти программ байты — нужно ли умножать на 2 адрес метки Tabl, т.е. адрес метки — это адрес в словах или байтах???

```
.cseg // SRAM

Tabl: db 0AH, 0BH, 0CH, 0DH, 0EH
…
LDI ZL,Low(Tabl) // Заносим младший байт адреса, в регистровую пару Z
LDI ZH,High(Tabl) // Заносим старший байт адреса, в регистровую пару Z
LPM R0,Z // В R0 лежит 0AH?
```

★ DI HALT

24 Июль 2009 в 0:56

cseg это ROM — флешка. SRAM это dseg

На флеше адресация флеша в ассемблере только по словам — т.е. метки указывают только на целые слова. Но реальная адресация все же побайтная, поэтому, чтобы загрузить 0AH

нужно умножать метку на 2:
Tabl: db 0AH, 0BH, 0CH, 0DH, 0EH
…
LDI ZL,Low(Tabl*2) // Заносим младший байт адреса, в регистровую пару Z
LDI ZH,High(Tabl*2) // Заносим старший байт адреса, в регистровую пару Z
LPM R0,Z // В R0 лежит 0AH?

Khroft

18 Август 2009 в 16:31

хм, странно, не нашел в книге по Тини и Мега команду сравнения РОН с константой с учетом переноса. Неужто ее нет?!

★ DI HALT

18 Август 2009 в 16:46

Вроде нет такой, а нафига нужно ваще сравнение с константой+перенос?

Khroft

18 Август 2009 в 18:04

делаю динамическую индикацию чисел (старший порядок — тысячи). Такая команда пригодилась бы при разбитии его из шестнадцатиричной формы на тысячи, десятки и т.д. Впринципе, проблему легко обошел, ведь можно сравнивать с РОНами, а их у меня вследствие фатальной простоты девайса завались:)

;r18:r17 — число, которое надо показать
ldi r19,low(1000) ;вычисление тысяч и в r0 их

```
ldi r20,high(1000)
clr r0

TH: cp r17,r19 ;если бы была комманда сравнения с
cpc r18,r20 ;конст. тут было бы немного короче
BRGE TH1
RJMP TH2

TH1: sub16 r18,r17,r20,r19 ;это простой макрос вычитания 16разрядных
inc r0
RJMP TH
TH2: сотни, десятки аналогично
```

это моя первая прога, (еще не дописанная) на ассемблере, так что не судите строго, но оно работает!:))

---

marsden

10 Май 2010 в 16:46

DI HALT, как насчет

; Позже освещу тему
; Математики на ассемблере.
?
Или я плохо по сайту рылся?

★ DI HALT

10 Май 2010 в 16:49

да все некогда.

dima_m

11 Июнь 2010 в 16:20

Не могу понять почему при исполнении кода вылазит сообщение на строчку
TABLE: .dw Way0, Way1 ,Way2,Way3, Way4 —> AVR Simulator: Invalid opcode 0x0002 at address 0x00000c
Проект скопирован целиком из этой статьи.

★ DI HALT

11 Июнь 2010 в 16:25

Дело в том, что эту таблицу нужно перепрыгивать джампами или распологать так, чтобы она не выполнилась никогда. Ведь там не опкоды, а адреса, но контроллеру все равно и он будет пытаться выполнить ее как код.

★ DI HALT

11 Июнь 2010 в 16:34

Немного подправил статью, чтобы явно показать этот момент.

dima_m

11 Июнь 2010 в 17:04

Ах вот оно что, а я уже часа 4 парюсь, с этой строкой, и так и эдак ее кручу верчу, не знаю куда засунуть. Триста раз уже эту строку переписывал в разных вариантах. Теперь, все работает пучком.

dima_m

11 Июнь 2010 в 17:28

А темы про математику на ассемблере, реально не хватает.

dpochechuev

9 Июль 2010 в 10:49

Делал такой алгоритм раньше. Но у меня получилось немного по другому.
Выполняется немного быстрее рассмотренного т.к. меньше команд с Z регистром
Но имеется ограничения на размер кода т.к. rjmp не может перебросить дальше 2к команд, но я думаю это ничего. Вот мой код. Таблица адресов переходов заменена на список rjmp объем кода тот же, что адреса что rjmp имеют объем 2 байта.

```
ldi r16,3; наш номер перехода (case)

ldi r31,high(startcase); В Z регистр положили адрес первого rjmp
ldi r30,low(startcase);

clr r17;
add r30,r16;
adc r31,r17; сложили Z и смещение

ijmp; Ушли куда нужно

startcase:
rjmp m0; Таблица переходов подряд адреса
rjmp m1;
rjmp m2;
rjmp m3;

rjmp m251;
rjmp m252;
rjmp m253;
rjmp m254;
```

```
;-----------------------
nop;
rjmp endcase; код для 255
;-----------------------
m0: nop;
; обработчики case
rjmp endcase;
;-----------------------
m1: nop;
;
rjmp endcase;
;-----------------------
m2: nop;
;
rjmp endcase;
;-----------------------
m3: nop;
;
rjmp endcase;
;-----------------------

;-----------------------
m251: nop;
;
rjmp endcase;
;-----------------------
m252: nop;
;
rjmp endcase;
```

```
;————————————————-
m253: nop;
;
rjmp endcase;
;————————————————-
m254:nop;
;
rjmp endcase;
;————————————————-

endcase: nop;
```

Прогонял в отладчике от начала case до обработчика всего 10 тактов.
Ваши коментарии…

---

grigoriy

20 Июль 2010 в 11:44

ОЧЕНЬ ЗАИНТЕРЕСОВАЛА ПРОГРАММКА таблицы переходов
типа LSL R20

```
LDI ZL, low(Table*2)
LDI ZH, High(Table*2)
CLR R21
ADD ZL, R20
ADC ZH, R21
LPM R20,Z+
LPM R21,Z
MOVW ZH:ZL,r21:r20
CLR TEMP
```

IJMP
Никак не врублюсь:
— будет ли эта программка работать в цикле, т.е. при многократном обращении к ней, по окончании одного раза сразу же повторно еще и еще?

★ DI HALT

20 Июль 2010 в 14:04

Если заново загрузить индексный регистр адресом начала таблицы, то почему нет?

---

grigoriy

20 Июль 2010 в 23:48

На базе Вашей информации возникла идея простой наглядной программы управления LED-индикаторами. Программа нагляная и легко настраивается на количество индикаторов.
Управление LED-индикаторами количеством N, выполняется по прерыванию от отдельного счетчика по типичным очень схожим блокам (кускам) программ. яркость регулируется частотой счетчика. В последней метке таблицы дополнительно команды загрузки индексного регистра началом таблицы.
Там же сформулирован неясный мне вопрос. Ниже приводится фрагмент программы вызываемой по прерыванию.

Table: .dw Indik0, Indik1, Indik2, Indik3, Indik4

LDI ZL, low(Table*2)
LDI ZH, High(Table*2)
CLR R21
ADD ZL, R20
ADC ZH, R21
LPM R20,Z+
LPM R21,Z

MOVW ZH:ZL,r21:r20
IJMP

Indik0:
Управление выбором индикатора 0
Управление выбором цифры на месте 0
RETI
Indik1:
Управление выбором индикатора 1
Управление выбором цифры на месте 1
RETI
Indik2:
Управление выбором индикатора 2
Управление выбором цифры на месте 2
RETI
…..
…..
Indik N:
Управление выбором индикатора N
Управление выбором цифры на месте N
LDI ZL-, low(Table*2) (Вопрос: ZL- или ZL?)
LDI ZH, High(Table*2)
RETI

★ DI HALT

21 Июль 2010 в 0:07

ZL конечно, а ZL- у тебя даже не скомпилируется, даст ошибку.

★ DI HALT

21 Июль 2010 в 0:09

А еще нельзя так выходить из прерывания. При входе в прерывание надо сохранять в стеке
1) SREG
2) Все используемые регистры

А при выходе их из стека доставать. Почитай у меня раздел про прерывания и подпрограммы. Там это четко расписано.

★ DI HALT

21 Июль 2010 в 0:10

И неясно зачем тебе загружать заново Z перед выходом из прерывания?

---

dima_m

30 Сентябрь 2010 в 11:56

Интересно, а ведь можно еще sreg и используемые регистры сохранять не только в стеке но и прямой загрузкой в озу используя команды sts и lds. Количество тактов то же. Тогда стек вообще можно не трогать, и обратно вынимать можно в любом порядке. Ну и все это в макрос закрутить и забыть. Как такая идея?

Но у меня вопрос в другом, попробовал я портянку из 30 way спрятать в макрос, чтоб в главном тексте программы не мешала, а компилятор выдает ошибки. Засада, пришлось обратно эту беду на экран вывести. Мешает дико. О… а если попробовать, это дело засунуть просто в отдельный файл типа .include «WAY.ASM». Попробовал, получилось. Жаль нельзя в макрос так красивее было бы.

★ DI HALT

30 Сентябрь 2010 в 14:16

А где ты адреса возвратов хранить будешь? И под каждое прерывание создавать кучу статичных ячееек в ОЗУ, где они будут тупо простаивать пока нет прерывания? Неэффективно.

В макрос нельзя, т.к. у тебя там абсолютные адреса переходов. Макрос их не понимает.

---

dima_m

25 Октябрь 2010 в 8:56

Получается наш данный case of работает от 0 до 255. А можно ли как то его расширить от 0 до 65535 тоесть до двухбайтной величины? Думаю что можно, только как надо подумать. А что думает DI?

В моей самой первой программе на ассемблере нужен был case до 600 гдето. Но тогда я не знал еще про индексные переходы. Тогда я вообще еще мало что знал. Вот и лепил как выше было сказано с помощью SPI/BREQ, сожрал половину памяти кристала. Но прикольно что эта плата до сих пор работает как год уже и хоть бы что.

★ DI HALT

25 Октябрь 2010 в 13:08

Да ради бога. Только складывать с базой придется два байта. Вот и все. Ну и сам параметр индекса будет двубайтным.

---

shiva

2 Январь 2011 в 19:38

Цитата:
«Таблица переходов это всего лишь строка данных в памяти, содержащая адреса Way0…Way4. Обратите внимание на то, что данные у нас двубайтные слова dw!!! Если же мы хотим адресовать расположенные данные, например вложенную таблицу адресов переходов, то адреса нужно умножать на два.

Вообще тут проще при написании по быстрому скомпилить кусок, открыть дамп с map файлом и посмотреть что лежит в памяти, куда что ссылается. Сразу станет понятно надо умножать на два или нет»

Как это сделать?

Плюс еще вопрос. Поясните поподробнее, плиз,строку вида:
MOVW ZH:ZL,r21:r20

★ DI HALT

2 Январь 2011 в 23:35

Что именно сделать?

Команда MOVW копирует сразу два байта. Тк что мы полностью загружаем Z из регистров R21:R20 за одну команду. Можно и за две ,если контроллер не поддерживает команду MOVW (тини вроде бы не умеют ее)

---

v.m.s.

13 Февраль 2011 в 16:25

Приветствую.
Идея алгоритма работы с Case с помощью ветвлений на индексных переходах понятна. Но возникли сложности с пониманием отдельных моментов. Вчасности:
ADD ZL, R20
ADC ZH, R21
LPM R20,Z+ ; Загрузили в R20 адрес из таблицы
LPM R21,Z ; Старший и младший байт
Раньше я считал, что по команде LPM в R20 мы загружаем содержимое R30. В P30, например, у меня $2C ($2C=$22+$0A, где $22=table*2, а $0A это Входное содержимое R20 умноженное на 2). После исполненения команды LPM R20,Z+ , содержимое R20 становится не $2C, а каким-то чудным образом становится равным $07 — адрес моего Way5.
Так как же всетаки работает LPM R20,Z+. Просветите, пожалуйста, новичка. :)

★ DI HALT

13 Февраль 2011 в 18:01

Загрузить в R20 число из адреса в Z и увеличить Z на 1

---

v.m.s.

13 Февраль 2011 в 19:37

Благодарю за оперативный ответ. Но всеравно пока не сходится — в Z хранится адрес
памяти программ 0x002c, но у меня по этому адресу — FF FF. Так откуда берется в R20 правильное значение Way5 — 07.
Подскажите, пжлста, куда смотреть (где искать) 07. :)

ЗЫ: а'м сорри за нуубские вопросы. просто, я в самом начале пути освоения асма.

★ DI HALT

13 Февраль 2011 в 20:35

Раскладка у нас какая? Есть адреса программ WayXX есть таблица Table в ячейках которой лежат адреса WayXX

Мы берем адрес этой таблицы, сразу умноженый на два, чтобы в байтах

```
1
2           LDI     ZL, low(Table*2)        ; Загружаем адрес нашей таблицы. Компилятор сам посчитает
3           LDI     ZH, High(Table*2)       ; Умножение и запишет уже результат.Старший и младший байты.
```

И к нему прибавляем смещение, чтобы найти адрес нужной ячейки, где лежит WayXX

```
1           CLR     R21             ; Сбрасываем регистр R21 - нам нужен ноль.
2           ADD     ZL, R20         ; Складываем младший байт адреса. Если возникнет переполнение
```

```
3           ADC     ZH, R21         ; То вылезет флаг переноса. Вторая команда складывает
4                                   ; с учетом переноса. Поскольку у нас R21=0, то по сути
5                                   ; мы прибавляем только флаг переноса. Вопросы - в комменты!
6                                   ; Таким образом складываются многобайтные числа.
7                                   ; Позже освещу тему
8                                   ; Математики на ассемблере.
```

И потом берем из этого адреса значения перехода (WayXX)
LPM R20,Z+ ; Загрузили в R20 адрес из таблицы
LPM R21,Z ; Старший и младший байт

Потом забрасываем в Z и делаем переход.

Так что у тебя в памяти, по адресу начиная с Table должены лежать пары байт равные адресам WayXX.

Смотреть надо в памяти программ (не путать с Data) и я не помню в чем там адресация в словах или в байтах.

---

v.m.s.

14 Февраль 2011 в 0:07

Например:
Память программ

```
1   Table:    000011      Way0   02 00
2             000012      Way1   03 00
3             000013      Way2   04 00
```

```
4                   000014      Way3   05 00
5                   ------      ----   -- --
```

При входном значении 0 в R20 перед выполнением команды
LPM R20,Z+ в Z будет 22 (table*2+смещение по case(0)*2).
Вопросы: 1) Как по адресу 0022 (в Z) осуществляется переход на слово с адресом 000011;
2) После выполнения LPM R20,Z+ в R20 будет 02. Это связано с тем что младший бит регистра Z (ZLSB)=0 (грузится младший байт слова по адресу 000011)?

★ DI HALT

14 Февраль 2011 в 0:18

Таблица бред и к действительности отношения не имеет. дай мне реальную раскладку по адресам что у тебя получились. Ну или проект замыль на dihalt@dihalt.ru я сам запущу и покажу что там и как.

★ DI HALT

14 Февраль 2011 в 0:24

1) По адресу Z лежит адрес перехода в частности 02 00

2) Да, мы по Z = 0011 (адрес таблицы) берем в R20:R21 число из 0011 (нулевая ячейка таблицы), а там лежит 02 00 (содержимое ячейки таблицы, искомый адрес перехода)

И потом на 02 00 (адрес метки Way) делаем переход.

★ DI HALT

14 Февраль 2011 в 1:31

Ну вот вижу твой проект:

```
.include "m32M1def.inc"   ; Используем ATMega16

;= Start macro.inc ================================

; Макросы тут

;= End  macro.inc ================================


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


; FLASH ===========================================
                .CSEG           ; Кодовый сегмент
                JMP START
Way0:   NOP
Way1:   NOP
Way2:   NOP
Way3:   NOP
Way4:   NOP
Way5:   NOP
Way6:   NOP
Way7:   NOP
Way8:   NOP
Way9:   NOP
```

```
Way10:  NOP
Way11:  NOP
Way12:  NOP

        JMP START

TABLE:  .dw Way0, Way1, Way2, Way3, Way4, Way5, Way6, Way7, Way8, Way9, Way10, Way11, Way12

START:  LDI R20,0
        LSL     R20             ; Сдвигом влево умножаем содержимое R20 на два.
                                ; Было, например, 011=3  стало 110=6 Круто да? ;)
                                ; опять же изза того, что у нас адреса двубайтные
                                ; НЕ ПУТАТЬ С ТЕМ ЧТО КОМПИЛЕР ОБСЧИТЫВАЕТ
                                ; ПАМЯТЬ В СЛОВАХ, тут несколько иное. Если непонятно
                                ; то в комменты пишите, обьясню.

        LDI     ZL, low(Table*2)        ; Загружаем адрес нашей таблицы. Компилятор сам посчитает
        LDI     ZH, High(Table*2)       ; Умножение и запишет уже результат.Старший и младший байты.

        CLR     R21             ; Сбрасываем регистр R21 - нам нужен ноль.
        ADD     ZL, R20         ; Складываем младший байт адреса. Если возникнет переполнение
        ADC     ZH, R21         ; То вылезет флаг переноса. Вторая команда складывает
                                ; с учетом переноса. Поскольку у нас R21=0, то по сути
                                ; мы прибавляем только флаг переноса. Вопросы - в комменты!
                                ; Таким образом складываются многобайтные числа.
                                ; Позже освещу тему
                                ; Математики на ассемблере.

        LPM     R20,Z+          ; Загрузили в R20 адрес из таблицы
```

```
56          LPM     R21,Z           ; Старший и младший байт
57
58
59          MOVW     ZH:ZL,r21:r20          ; забросили адрес в Z
60
61          IJMP                    ; Обана и переход на наш выбранный Way. Понравилось?
62                                  ; То ли еще будет :)
63
64  ; EEPROM =================
65                  .ESEG           ; Сегмент EEPROM
```

Студия показывает, что метка Table численно равна 0x00011 это в словах. Умножается на два сразу же и Т.е. в байтах у нас будет 0x22 это число оказывается в Z. Теперь смотрим в память, что лежит по адресу 0x22 память там указана тоже в словах. По этому смотрим на адрес 0x11 (0x22 деленый на два) там 0200 значение (Way0 в словах). Но это всего лишь отображение в редакторе студии (в словах)

Реальная же адресация для команд работы с памятью байтовая (потому и умножаем на два чтобы адрес из словесного перевести в байтовый). И адрес 0x22 указывает на первый байт (02), а адрес 0x23 указываает на второй байт (00). Из студии же мы можем меткой адресоваться без ухищрений только по четным байтам (границам слов).

Эти два байта 02 00 из ячеек 0x22:0x23 (или слово с адресом 0x11) мы загребаем в наши регистры R20:R21 на временное хранение. После копируем в Z — и это уже готовый адрес Way0 лежащий в той таблице. ПО нему уже делаем Ijmp

Если посмотришь на адрес в таблице, то дальше после 02 00 идет 03 00 потом 04 00 — это Way1 и так далее. Каждый адрес двубайтный, поэтому и ячейка под него двубайтная и чтобы выбрать следующую ячейку мы смещение перед прибавлением умножаем на два.

v.m.s.

14 Февраль 2011 в 2:11

Понял. Спасибо.)

---

https://profiles.google.com/zagzag2008

27 Март 2011 в 18:07

Ув., DI HALT
Реально ли сэкономить память для ATTiny26 за счёт использования только ZL?
[CODE].CSEG ; Code
M1:
LDI R20, 15 ; Индекс элемента в Table для перехода
LSL R20 ; Умножаем на 2
LDI ZL, Table * 2
ADD ZL, R20 ; Складываем в ZL = Table * 2 + 15
LPM R20, Z ; Читаем значение из Table[15]
MOV ZL, r20 ; Индекс перехода в ZL
IJMP ; Прыгаем
RJMP M1 ; Возврат в начало

Table: .dw Way0, Way1, Way2, Way3, Way4, Way5, Way6, Way7, Way8, Way9, Way10, Way11, Way12, Way13, Way14, Way15, Way16, Way17, Way18, Way19, Way20, Way21, Way22, Way23, Way24, Way25, Way26, Way27, Way28, Way29, Way30, Way31, Way32, Way33, Way34, Way35, Way36, Way37, Way38, Way39

Way0:
RET

Way1:
RET

…
[/CODE]

★ DI HALT

27 Март 2011 в 18:14

Если хватит длины одного байта. Но в этом случае ZH должна быть равна нулю, а иначе проц ускачет куда угодно.

salpingot

22 Апрель 2011 в 2:58

; Способ первый с помощью возврата из подпрограммы (создания таблицы с векторами, если не правильно назвал, то прошу поправить), значения по просьбам трудящихся двух байтное 0xFFFF, что обеспечивает 65 535 вариантов.

```
1                in              r16, pinA; low младший байт 16 битного значения;
2                in              r17, pinb; high старший байт, 2 байтного значния;
3                add             r16, r16; умножаем на два с помощью сложения, т.к. команда jmp занимает 2 байта
4    + 2 байта
5                adc             r17, r17; адрес перехода во флеше, т.е нам надо сместиться относительно таблицы
6    в два раза.
7                ldi             zh, high (tablica); можно использовать любые другие регистры
8                ldi             zl, low (tablica); для загрузки адреса таблицы
9                add             zl, r16; нулевое значение таблицы + умноженное в двое
10               adc             zh, r17; входящего значения в порт pinA and pinB
11               push            zl; запихиваем младший байт в стек
12               push            zh; запихиваем старший байт в стек
13               ret; а теперь загрузка адреса возврата из стека и переход по нему
14   tablica:
```

```
15      jmp     a0;
16      jmp     a1;
17      jmp     a2;
18      jmp     a3;
19      jmp     a4;
20      jmp     a5;
21      jmp     a6;
22      nop;
23      a0: nop;
24      a1: nop;
25      a2: nop;
26      a3: nop;
27      a4: nop;
28      a5: nop;
29      a6: nop;
30      nop;
31
32      ; Способ второй с помощью команд, косвенного перехода (вызова), поспешу напомнить, что LMP считывает значение
33      ячеек памяти по адресу указанном регистровой паре «Z» т.е. ячейки памяти (кластеры) в AVR хранятся в словах и
34      составляют слово=DW=dw=2 байтам= 16 битам, в связи с этим происходит умножение на два, для тех кто
35      экспериментирует, прошу обратить внимание на reset собственно адрес 0x0000, а дальше вся нумерация флеша идет
36      через один, т.е. 0x0000, 0x0002, 0x0004, 0x0006 и т.д.
37                              in              zl, pinA; low младший байт 16 битного значения;
38                              in              zh, pinB; high старший байт, 2 байтного значния;
39                              clc; очищаем флаг «C» перед умножением на два, для чистоты операции;
40                              lsl             zl; как и в примере выше умножаем входящее значение на два,
41                              rol             zh; с помощью операции сдвига с учетом флага переноса «C»
42                              ldi             yh, High (tablica*2); грузим в регистровую пару значение
43                              ldi             yl, Low  (tablica*2); адреса таблицы переходов
```

```
                        clc; очищаем флаг «C» перед сложением  регистровых пар,
                        add             zl, yl; складываем младшие значения регистров
                        adc             zh, yh; с учетом «C» старшие теперь сложили
                        lpm             xl, z+; загрузили в регистровую пару значение указанное по адресу в
«Z», младший байт
                        lpm             xh, z;  а после знака + следующую ячейку, т.е. старший байт
                        movw            zh:zl, xh:xl; скопировали результат в «Z», т.е. адрес перехода
                        icall; ну и собственно осуществили переход
Tablica:        .dw E_0, E_1, E_2, E_3, E_4;
E_0: nop;
E_1: nop;
E_2: nop;
E_3: nop;
E_4: nop;
;Ди Халт, есть еще способы? Или их только два? Может еще есть, какие ни будь варианты кроме ret and icall?
;Так и не разобрался с листингом, это где ссылочка есть типа большой кусок кода Постоянная ссылка, по которой
он размещён:
```

http://easyelectronics.ru/repository.php?act=view&id=48 , так что напишу я по старинке, (чуть позже попытаюсь освоить этот листинг) -)) (Если есть вариант Ди поправь то что я написал пожалуйста и если не тяжело, напиши где можно посмотреть как вставлять листинг в форум, что бы другие тоже могли видеть), заранее благодарю.

★ DI HALT

22 Апрель 2011 в 8:18

В первом случае у тебя таблица неправильная. Там не нужны jmp a0 достаточно только адресов. Как в варианте два. Ты же в стек пихаешь не команду перехода, а адрес возврата. А Так у тебя там будет каша — по два слова команд с адресами.

На AVR вариантов пожалуй два. Первый — переход через модификацию стека и RET, второй через icall и ijmp команды (более правильный, так как более явный и читаемый)

tokiw

19 Сентябрь 2011 в 18:29

Поясните, что делают команды LPM и MOVW? Я так понял, после команд ADD и ADC в регистре Z уже будет храниться адрес нужного перехода? Что происходит потом?

★ DI HALT

20 Сентябрь 2011 в 8:35

Lpm грузит в регистр r0 значение из ячейки флеш памяти с адресом z. A movw просто копирует две регистровые пары из пары в пару

tokiw

20 Сентябрь 2011 в 17:13

То есть после ADC в регистре Z будет храниться адрес ячейки WayX, а после LPM в регистрах r20:r21 будет храниться содержимое ячейки WayX, так? А зачем потом копировать это значение в Z?
Кстати, Z+ это такое указание на старший байт регистра?

★ DI HALT

20 Сентябрь 2011 в 19:21

Да так. Z+ это операция с последействием Т.е. мы читаем по адресу Z, а потом Z+1. Таким образом при следующем LPM будет чтение с адреса Z+1.

А копируем мы все в Z мы для команды IJMP т.к. она делает прыжок по адресу лежащему в Z

В статье же написано:
>Делается это все круче. Помнишь я тебе рассказывал в прошлых уроках о таких командах как ICALL и IJMP. Нет, это не новомодная яблочная истерия, а индексный переход. Прикол в том, что переход (или вызов подпрограммы, не важно) осуществляется тут не по метке, а по адресу в регистре Z (о том что Z это пара R30:R31 я пожалуй больше напоминать не буду, пора бы запомнить).

---

salpingot

20 Сентябрь 2011 в 2:11

http://musmu.narod.ru/atmel/avrasm_rus.html
http://www.mymcu.ru/Articles/Atmel11.htm

это набор команд. там вроде коротко достаточно и понятно. у меня
есть сборка команд можно скачать здесь Последняя на сегодня версия.doc.html

---

Shinigami_Neko

30 Март 2012 в 12:46

А что если у меня в r20 не числа идущие по порядку как п примере 0,1,2,3… а произвольные например 150,70,50,200 и в зависимости от того какое число мне нужно уйти по заданному адресу/метке указанному в таблеце (table: .dw way1,way2,way…). этоже получается лепить 200 переходов?! помоему не гуманно. и если число DI HALT Можешь подсказать?
ЗЫ надеюсь кто-нибуть подскажет…

★ DI HALT

30 Март 2012 в 12:56

Тогда только делать N сравнений для каждого варианта. Других способов нет.

Еще можно сделать табличное приведение по числу. Т.е. числу 200 соответствует первая ячейка таблицы. числу 150 вторая. И так далее. А ты вначале находишь какое у тебя число, пробегая по таблице и на основании этого делаешь переход. Но это уже дольше по времени.

Shinigami_Neko

30 Март 2012 в 14:50

А этот варит будет практичнее чем юзать cpi с brne? если да. можно махонький примерчик?

★ DI HALT

30 Март 2012 в 16:16

Зависит от количества сравнений. По скорости примерно одинаково, но таблицу с вариантами можно выбирать в цикле. А значит выйграем по обьему кода. Т.е. брать значение и последовательно сравнивать его с каждым вариантом таблицы. Щелкая ячейками — номер ячейки будет индексом к таблице переходов.

★ DI HALT

30 Март 2012 в 17:01

Есть еще один изврат вариант.

Делаешь таблицу переходов ныкая ее прямо в код. саму таблиц делаешь на абсолютных адресах. А дырки в пустотах заполняешь кодом, обходя безусловными переходами куски таблицы.

т.е. как то так:
NOP
NOP
NOP
RJMP M1

```
ORG xxxx
tabel_1: .dw 1,2,3
M1: NOP
NOP
NOP
NOP
RJMP M2
ORG xxxx+150
tabel_2 .dw 150,151
M2: NOP
NOP
NOP
```

Конечно тут придется считать байты. чтобы все влезло и тщательно все контроллировать. И код получается просто адовый (особенно для тех кто его будет дизассеблить :)). Но при этом мы получаем высокую скорость обработки вариантов и не треяем места под таблицу.

Разумеется вместо NOP NOP NOP пишем свой полезный код.

Shinigami_Neko

30 Март 2012 в 16:41

О как, Спасибо огромное!! Даже не представляю, чтоб без вас делал!

★ DI HALT

30 Март 2012 в 16:56

Че делал, че делал. Хреначил бы на CPI-BREQ :)

---

warezzzok

8 Апрель 2012 в 1:59

Здравствуйте. Спасибо за курс, очень интересно и понятно. Почти все :)
Разбираюсь сейчас с ветвлением, в общем, понятно все, за исключением одного момента с Z+. Понимаю, что обсуждалось, прочитал все комменты, нашел кое-что еще интересное, но так и не понял смысл эти хстрочек.

lpm r20, Z+ ; Так понял, что тут в регистр R20 (который в последствии будет присвоен ZL) падает адрес метки, на которую надо перескочить, потом адрес этой метки увеличивается на 1.
lpm r21, Z ; В R21 падает значение чего?

Почему мы пишем Z и Z+, а не оперируем ZH и ZL?
Возможно ли эти 2 строчки записать как-то иначе?
Если все сделать, как написано, то работает, но хочется разобраться до конца.
Спасибо.

★ DI HALT

8 Апрель 2012 в 2:55

Просто в AVR есть удобные команды которые сразу оперируют регистровой парой ZH:ZL при загрузке значения. Можно и по отдельности складывать, а потом учитывать переносы (прибавлять то надо будет к двубайтному числу).
Скажем так

```
lpm R20, Z
LDI R0,1
ADD ZL,R0
CLR R0
ADC ZH,R0
LPM R21, Z
```

Но зачем если для этого есть спец команда которая сработает быстрей и займет меньше места?

Адрес метки лежит в памяти. Т.к. адрес двухбайтный, то занимает он две ячейки памяти. У каждой ячейки памяти тоже двубайтный адрес, он у нас загружен в Z

Берем первый байт адреса метки
LPM r20, Z+
Грузит в R20 число на которое ссылается регистрвоая пара Z, а после загрузки Z увлеичивается на 1. Выбирая следующую ячейку, где лежит второй байт адреса метки.

В результате у нас в R20:R21 оказывается полный адрес метки.

Вообще LPM Rx, Z+ можно использовать и просто если надо увеличить Z на единицу. Пофиг что какое то значение будет загружено в Rx. Побочное явление :)

---

warezzzok

9 Апрель 2012 в 12:08

T.e при записи lpm r20, Z в r20 будет записана только часть адреса, вторая часть будет просто отсекаться? Почему бы тогда не записать lpm r20, ZL?
Сейчас на работе, студии под рукой нет, проверить будет ли работать не смогу.

★ DI HALT

9 Апрель 2012 в 13:17

Адрес ДВУХБАЙТНЫЙ. и в R20 записыватся не Z, а байт который лежит по адресу ячейки адрес которой записан в Z.

---

warezzzok

9 Апрель 2012 в 14:03

Аааааа! Сообразил теперь. Спасибо за помощ:)

Senbad

13 Май 2012 в 21:07

Не понимаю зачем грузить в ZH и ZL адресс WayX,
LPM R20,Z+ ; Загрузили в R20 адрес из таблицы
LPM R21,Z ; Старший и младший байт

потом опять грузить из R20,R21 В ZH,ZL
MOVW ZH:ZL,r21:r20 ; забросили адрес в Z

Как я понимаю ZH тоже самое что и Z, а ZL тоже самое что Z+1?
У нас получается цепочка Z->R20_>Z. Почему сразу после ADC нельзя поставить IJMP?

★ DI HALT

13 Май 2012 в 21:20

Нет. ZH это регистр R31, а ZL это регистр R30. Потому и грузим их отдельно.

Нельзя, т.к после ADC в Z нахоодится не адрес перехода, а адрес ячейки где лежит адрес перехгода. Потом мы черзе LPM берем по этому адресу адрес перехода, кладем его в Z и переходим.

A Z+1 это адрес лежащий в Z +1

Senbad

14 Май 2012 в 19:53

Позвольте уточнить ещё:
1)Как я понял условно Z=ZL, а Z+1=ZH? так ли это? и почему тогда с LPM нельзя использовать ZL и ZH?
2)Почитал комментарии и совсем запутался где адресация словами, а где байтами? Если я правильно понимаю, то в реальном контроллере вся адрессация побайтовая, а слова-это уже прикол компиляторов? т.е. в SRAM и ROM аресация в словах? В каких случаях нужно умножать на 2?

★ DI HALT

17 Май 2012 в 16:28

Нет Z это ZL:ZH почитайте про систему команд чтоль.
Z+1 это +1 к адресу в ZL:ZH

С LPM можно использовать только Z т.к. адрес 16т разрядный.

Адресация словами это чисто прикол компилятора и только когда речь идет о флеше. Т.е. если берем адрес по метке в .cseg сегменте, то его надо умножать на два. Для RAM и EEPROM это не актуально.

Senbad

17 Май 2012 в 16:10

Позвольте уточнить ещё:
1)Как я понял условно Z=ZL, а Z+1=ZH? так ли это? и почему тогда с LPM нельзя использовать ZL и ZH?
2)Почитал комментарии и совсем запутался где адресация словами, а где байтами? Если я правильно понимаю, то в реальном контроллере вся адрессация побайтовая, а слова-это уже прикол компиляторов? т.е. в SRAM и ROM аресация в словах? В каких случаях нужно умножать на 2?

---

ilya123

22 Май 2012 в 1:04

приветствую всех
DI HALT,во первых спасибо за сайт, учусь практически по нему..
затеял я тут сделать частотник трехфазный для управления АС двигателем,нужно трехфазный синус нарисовать,решил на Attiny261 в режиме 6PWM, есть таблица синусов (255 точек) ,в один регистр сравнения пишется все хорошо,в два других регистра надо писать со смещением в 120 градусов(смещение на 85 и 170 по таблице),вот тут появились проблемы…

я сильно был не прав ? при моделировании в авр-студии происходит переполнение регистров A и D
нет перехода к началу таблицы,с регистром A все в порядке,в протеусе моделирование как то идет,только сдвига на 120 градусов почему то не видно..
в чем я накосячил ?
пробовал еще команду ADIW использовать, чтоб регисты лишние не задействовать, компиляция вообще не пошла…
еще хотелось спросить, если таблица не 255 значений,а меньше, как запустить процесс по кольцу,как отслеживать что таблица закончилась ?

---

ilya123

22 Май 2012 в 1:06

как то коряво все вставилось…

---

ilya123

22 Май 2012 в 12:21

вот, переписал, вставил как листинг, сохранилась как ссылка
http://easyelectronics.ru/repository.php?act=view&id=88

★ DI HALT

23 Май 2012 в 22:49

НУ перенос там закономерный. Ты же при
add ZL, r17 можешь получить C, а следующая команда у тебя уже adc
adc ZH, r18 которая эту C учтет. Так что все верно. Правда как то ты странно складываешь. У тебя R18 чему равен? По идее должен быть равен нулю. Чтобы не мешать. Тогда ты просто прибавляешь C если оно того требует. У тебя же смещение небольшое. Не более 255 за раз. А где у тебя R18 забивается нулем? По идее надо перед вычислением его зачистить
CLR R18, например.

С ADIW все верно. Только оно прибавляет сразу же пару. Т.е. ты указываешь ему в качестве аргумента младший регистр пары, а старший будет следующим по списку. Т.е. ты этой командой прибавишь к паре r30:r31 число 85.

А вот регистры ты в стеке сохраняешь правильно, но не до конца. Мало сохранить SREG ,надо еще последовательно сныкать и достать все регистры которые ты используешь в прерывании. Иначе, если они применяются в фоновой программе получишь полный пердимоноколь.

Чтобы в таблице было меньше значений, то есть три варианта:

Медленный, но компактный — мы при увеличении индекса сравниваем его с переполнением по текущему размеру массива и корреткируем так, чтобы он вышел в начало. Чистая арифметика и сравнения.

Быстрый, но жирный. Не мудря делаем таблицу с шагом, скажем, через два значения. Тогда тебе надо лишь будет выбирать кратные смещения, чтобы при переполеннии оно само вылезало на другую сторону массива.

Есть еще третий вариант, с масками. Тут у тебя массив должен быть кратный степени двойки. Скажем не 256 значений, а 64 или 32 или 128, а свое смещение ты как обычно инкрементируешь словно оно у тебя до 256, а потом отрезаешь старшие биты по маске через AND. в результате смещение у тебя никогда не превысит диапазон больше указанного в маске. Т.е. 32, 64, и тыды. Сколько отрежешь старших битов. Быстро и компактно.

---

ilya123

24 Май 2012 в 18:30

спасибо большое за ответ.
многое прояснилось,особенно по поводу таблицы.
в регистре r18 у меня ноль и лежит,тут все вроде верно,регистры не сохраняю пока,потому как они
пока ни где более не используются (это пока только черновик),пока ни как таймер не могу заставить
генерить синус со смещением на три выхода,в авр-студии вроде что то идет,а в протеусе все три синуса идут синхронно..
вот по поводу add не совсем понял,я думал что эта команда складывает без переноса,значит переполнение в следующем регистре не учитывается…
тогда мне надо наверно вместо adc тоже использовать add , таблица то маленькая, в один байт умещается..?мне перенос то совсем не нужен..

★ DI HALT

24 Май 2012 в 21:29

add не учтет имеющийся перенос в СВОЕЙ операции, на младшем байте. Но флаг переноса встанет обязательно. А ADC прибалвяет уже с учетом переноса который может быть.

Не не не! Перенос нужно учитывать обязательно! Ты ведь прибавляешь смещение к двубайтому АДРЕСУ, а откуда ты знаешь какой он будет в итоге? Он же динамически компилятором вычисляется. И будет он 00FF, и прибавишь ты к нему число, пусть даже 1, и будет у тебя C на младшем байте, а результат вычисления будет как

ZH:ZL = 00FF
R18:R17 = 0001

ADD ZL,R17 = FF + 1 = 00 + C
ADC ZH,R18 = 00 + 0 + C = 01

ZH:ZL = 0100

Вот так вот.

---

ilya123

25 Май 2012 в 11:36

так…
все правильно… с чего я решил,что в начальном адресе массива в старшем байте будут нули ??
1) тогда как же мне быть, я прибавляя смещение по адресу (свои 85 и170), вылечу за пределы массива ??
придется ,что ли, писать в таблицу свое значение для каждого регистра…заодно и количество точек аппроксимации уменьшится в три раза,если массив оставить того же размера..
2) или как то все таки можно остаться в своем адресном пространстве ? (прошу прощения за глупый вопрос, я конечно понимаю,что можно,только до меня не доходит как…)
3) дальше еще возникает куча вопросов, допустим ,я получил свой трехфазный синус,дальше ,как плавно регулировать частоту ? Просто изменяя TOP я искажу форму сигнала… а мне еще надо сохранить постоянным соотношение V/F ,пытался разобраться в апноутах (там есть примеры управления двигателем),но я в СИ,мягко говоря не силен, понял так , что там берут табличное значение и пересчитывают ,в зависимости от частоты,как то сложно все… мне вроде такая точность не нужна , нужно крутить асинхронник мощностью 0,8кВт, можно даже ступенчато изменять частоту в пределах от 50 до 500 Гц,думал операциями сдвига пересчитывать,тогда дискретность получается совсем большая,в геометрической прогрессии.. если просто прибавлять дискретные значения,так функции все нелинейные…или уж написать кучу таблиц,благо место вроде позволяет,и перебирать их ??

Alex0720

31 Май 2012 в 0:49

Доброго времени суток. Цитата: «Вообще тут проще при написании по быстрому скомпилить кусок, открыть дамп с map файлом и посмотреть что лежит в памяти, куда что ссылается. Сразу станет понятно надо умножать на два или нет.»
Открыл map файл, не хрена не понятно, сплошные EQU+имена регистров+чего то еще… =(Гугль and Хуюгль как понимать это, толком ни чего не сказали. Если можно хотя бы пару слов о этом файле.
начало файла примерно такое:
AVRASM ver. 2.1.30 E:\Coding\AVR\2\atmega16\atmega16.asm Wed May 30 21:52:15 2012

EQU SIGNATURE_000 0000001e
EQU SIGNATURE_001 00000094
EQU SIGNATURE_002 00000003
EQU SREG 0000003f
EQU SPL 0000003d
EQU SPH 0000003e
EQU OCR0 0000003c
EQU GICR 0000003b
EQU GIFR 0000003a
EQU TIMSK 00000039
EQU TIFR 00000038
EQU SPMCSR 00000037

★ DI HALT

31 Май 2012 в 6:51

А что там непонятного? Сразу же видно что означает та или иная символическая метка. Сначала указывается адресное пространство, потом сама метка и чему она в итоге равна.

EQU SIGNATURE_000 0000001e, например значит, что компилятор заменит SIGNATURE_000 на 0000001e при итоговой сборке. А вот если смотреть дальше, то будет интересней. Будут адреса меток, переменных и тыды, с указанием памяти. Например, так:

CSEG RAM_Flush 00000074

Указывает, что метка RAM_Flush находится в флеше по адресу 00000074

Или:

DSEG TimersPool 00000089, что переменная TimersPool лежит в ОЗУ по адресу 00000089.

А зная адреса символьных имен для текущей компиляции можно погонять кусок кода в симуляторе и посмотреть те ли цифры записываются в регистры. А если не те, то почему они не те.

Alex0720

31 Май 2012 в 12:25

Спасиб за ответ..но не доехал…туплю дальше… %)
ИМХО данный файл представляет перечень всех регистров проца и тогда вот что непонятно:
EQU SPL 0000003d
EQU SPH 0000003e
Компилятор сам определил стек??? «EQU SPL 0000003d» обозначает что при компиляции сегмент SPL находится по адресу 0000003d? Или в SPL пихает 0000003d? Но тогда по какому адресу?

Alex0720

4 Июнь 2012 в 21:55

ура!!! кажется дошло… %)
после сигнатуры проца, идут АДРЕСА РЕГИСТРОВ. Сразу после них, биты и их номера в регистре. Вроде как то так…

★ DI HALT

5 Июнь 2012 в 9:17

Загляни еще в дефайны на твой контроллер. Это тот файл с определениями, что подключается в самом начале программы, где ты указываешь компилятору с каким контроллером работаешь. Там все адреса на данный контроллер. Естественно они, как есть, попадают и в map файл.

★ DI HALT

5 Июнь 2012 в 9:15

Это не сегмент SPL а младший байт регистра указателя стека, точнее его адрес. У каждого AVR Контроллера этот адрес свой и прописан он в inc файле на контроллер.

---

limburan

9 Июль 2013 в 0:01

Хм. Мне кажется, статья была бы более понятно, если бы сначала был подробно описан алгоритм происходящего с помощью блок-схем или просто слов, а то вкупе с не сразу понимаемой ассемблерной математикой не оч понятно :)

Kolobok13

11 Январь 2014 в 18:56

Асм конечно тру=) Но ИХМО многим будет интересно как сделать это на си.
Сам узнал недавно, вдруг кому нужно будет.
Для затравки кусок кода простой, но самый наглядный (спасибо камраду ReAl):

```
ISR(INT0_vect)
{
static void *next_state = &&start;

goto *next_state;

start:
next_state = &&idle;
return;

idle:
if (PINB & 0x01) {
PORTB |= 0x80;
```

```
next_state = &&drain;
}
return;

drain:
if (PINB & 0x02) {
PORTB &= ~0x80;
next_state = &&idle;
}
return;
}
```

и несколько ссылок:

★ DI HALT

11 Январь 2014 в 19:04

Обычно достаточно последовательного CASE. Компилятору (особенно если это IAR или ICC) хватает мозгов замутить индексный переход самостоятельно.

Kolobok13

11 Январь 2014 в 18:58

и несколько ссылок:

http://electronix.ru/forum/index.php?showtopic=101163

http://electronix.ru/forum/index.php?showtopic=61802&st=0

http://electronix.ru/forum/index.php?showtopic=56739&st=0

Kolobok13

11 Январь 2014 в 21:00

DI HALT, ты прав=). Спасибо за оперативный ответ.
Только что попробовал нагородить switch с кучей case на gcc, оптимизация O1.
Просмотрел дисассемблер.
Железная логика компилятора не подкачала и начала использовать индексные переходы.
Только индексный переход производился ненапрямую а через дополнительный rjmp.
Такчто switch-case рулит!!!
P.S. Спасибо за easyelectronics!!!

Adler

13 Январь 2014 в 15:47

Очень полезная статья,спасибо автору,как раз
то что я искал,но мне надо к этому прикрутить
работу с AT-командам, например:
по uart приходит фраза AT+openport и выполняется
какое то действие,если другая команда соответственно
другое действие,с описанным в статье case теперь все ясно ,но
как быстренько проверять на соответствие принятые данные,(у меня правда есть один муторный способ)…
помогите пожалуйста.

Stark

22 Сентябрь 2015 в 0:11

Итак, вначале пишем дофига вариантов наших действий. Те самые сто путей, у меня будет не сотня, а всего пять, но это не важно.

1 Way0: NOP
2 Way1: NOP
3 Way2: NOP
4 Way3: NOP
5 Way4: NOP

А эти варианты прописывать в отделе подпрограмм?????

1 Way0: NOP RET
2 Way1: NOP RET
3 Way2: NOP RET
4 Way3: NOP RET
5 Way4: NOP RET
Наверно так?

★ DI HALT

22 Сентябрь 2015 в 2:18

Нет, RET там не надо. Ведь мы уходим по IJMP. В случае RET ты получишь срыв стека, ведь в стеке адреса возврата нет. Но ты можешь сделать уход по ICALL и тогда RET тебя вернет обратно. Тогда, можно сделать 100500 подпрограмм.

---

kaki-malaki

16 Октябрь 2015 в 4:49

Здравствуйте DI HALT и команда))
Заранее скажу, что коменты просмотрел, но все таки осталось парочку вопросов:
1)LDI ZL, low(Table*2) — делая вот так мы не боимася, что у нас может байт вылезти за рамки и складывая потом старший байт мы это не учтем, что приведет к потере информации(адресации не туда, куда хотели)?
2)LSL R20 — сюда грузится кейс, но почему мы его на два не умножаем, ведь таблица у нас в словах?

3)Ну и для закрепления информации) про это заезденное умножение на 2. Насколько я понял, программа во флеше адресует себя словами в 2 байта, ибо это выгодно, так как является минимальным размером команды, но от нас она ждет адреса в байтах поэтому и делаем умножение, так?
Заранее спасибо.

★ DI HALT

20 Октябрь 2015 в 1:32

1) Конечно может. И это надо учитывать в больших МК, где адресация больше двух байт. Тут же 16 разрядов за глаза хватает на адресацию, так что вся память ПЗУ укладывается 65535 слов.

2) LSL это не Load, а SHIF :)))) LSL это и есть умножение на 2 :) Грузится он где то раньше. Т.е. в примере у нас уже в R20 есть кейс. И мы его умножаем сдвигом (LSL) на два.

3) Ситуация тут такая:

Команды загрузки данных оперируют ТОЛЬКО Байтами. Так что если тебе надо LOAD сделать, то ищи байтовый адрес.
Команды передачи управления, оперируют ТОЛЬКО Словами. Так что если тебе надо JMP сделать или CALL чего бы то ни было, то адрес должен быть в словах. То же касается и PC если ты к нему через стек обращаешься (модификация адресов возврата в стеке)
Компилятор все метки по умолчанию считает как метками для команд передачи управления, т.е. считает их в словах, поэтому если мы метками размечаем данные чтобы грузить оттуда, то их надо умножать на два.

При этом LSL R20 не относится к данной адресной путанице. Это лишь чтобы нашу таблицу смасштабировать на то, что у нас адреса двухбайтные, а значит одна ячейка таблицы занимает 2 байта. Был бы адрес 4х байтный (больше 65кбайт памяти), то умножать пришлось бы на 4.

---

bondyara

3 Май 2016 в 11:30

Не понял одну вещь в переходе по таблице.
Загружаем адрес таблицы в Z:
LDI ZL, low(Table*2)
LDI ZH, High(Table*2)

Прибавляем нужное смещение:
CLR R21
ADD ZL, R20
ADC ZH, R21
Теперь у нас в Z должен быть адрес перехода WayN. Почему его нельзя сразу использовать, а нужно прогнать через R21:R20?

Опять копируем Z в R21:R20
LPM R20,Z+
LPM R21,Z

И достаем обратно:
MOVW ZH:ZL,r21:r20

Каков смысл в последних двух действиях?

★ DI HALT

3 Май 2016 в 18:42

«Опять копируем Z в R21:R20
LPM R20,Z+
LPM R21,Z»

Вот только LPM это ни разу не копирование. Это загрузка из ПЗУ.