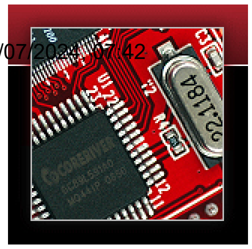

STM8 assembler

STM8

In the process of mastering STMs, I really missed the description of their assembler. This is not about the mnemonics of the instructions of the computing core of microcontrollers, but about the principles of constructing assembler programs in STVD. After the AVR core assembler, everything in the STM assembler looked outrageously illogical to me. For some time I even wanted to write my own assembler :) Now I will try to make it easier for those who will master the assembler for STM8 to run over the rake.

Beginning and end of assembly file

First of all, this assembler requires that the target microcontroller family be specified. It can be **st7** or **stm8** . In this case, the assembler will select the necessary **.TAB** file (located in the *..\st_toolset\asm* directory). The target family must be specified in the very first line of the file and begins with the very first character. The end of the file must be signaled by the keyword **END** . Moreover, this **end must not be the first** in the line . At the beginning of the line, you must put a space or a tabulation symbol. And the line with the word **END** must not be the last. There must be at least one more line after that, and you can write more than one. And you can write any nonsense there: the compiler will not pay attention to it!

Program text

The program line, except for service lines, looks like this:

```
[label[:]]<space>[opcode]<space>[operand]<space>[;comment]
```

<space> may contain spaces or tabs. The string does not necessarily contain all of these elements.

Tags

The label, if any, is written starting from the very first character of the line! At the end of the label, you can either put a colon or not. The assembler will survive both options. The label can contain:

- Upper and lower case Latin letters (**AZ** and **az**)
- Digits (**0** - **9**). The first character in the label name cannot be a digit.
- Dash sign (**_**)

Live

Comments Publications

penzet → Sprint Layout in OS X 18 → Software for electronics engineer

Vga → EmBitz 6 → Software for electronics engineer

Vga → Rail-to-rail: ideal OU or a clever marketing ploy? 1 → Theory, measurements and calculations

Flint → A little more about 1-wire + UART 56 → Hardware connection to the computer.

penzet → Cross-platform terminal - SerIO 3.x 25 → Software for electronics engineer

Gornist → PIP regulator 2 → Algorithms and software solutions

whoim → CC1101, Treatise on Tracing 89 → Blog im. khomin

OlegG → Clock on Bluetooth LE module 8 → Cypress PSoC

Technicum505SU → Nuances of PWM control of a DC motor by a microcontroller 3 → Theory, measurements and calculations

sunjob → DDS synthesizer AD9833 88 → Blog named after grand1987

DIHALT → W801, LCD screen and fly in the ointment 2 → Detail

nictrace → New Arduino-compatible board. 20 → FPGA

sunjob → Connecting the ARM GCC compiler to CLion 1 → Software for electronics engineers

Vga → CRC32: on STM32 as on PC or on PC as on STM32. 58 → STM32

dmitrij999 → Capturing images from a USB camera using STM32 6 → STM32

Gilaks → Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin. 8 → Theory, measurements and calculations

sva_omsk → Lithium ECAD - Russian PCB CAD 40 → Software for electronics engineer

x893 → W801 - budget controller with Wi-Fi 4 → Details

podkassetnik → Changing the standard instrument cluster lighting of Logan-like cars 5 → Automotive electronics

In addition, attributes can be added to the label. Label size. The fact is that the STM8 computing core supports several addressing modes, differing in the bit depth of the address set on the address bus. And the instructions associated with this, as a rule, also have several options with different address sizes.

- .B — the address has a size of **8 bits** . It can point to addresses $0000-$00FF, i.e. all senior digits are filled with zeros. Allows to reduce work with variables placed in the **RAM0** segment . In my opinion, it has no more practical use. Segments are discussed below.
- .W — the address has a size of **16 bits** . In the memory of, for example, STM8L152, this allows you to access anywhere. If the size attribute is not specified, it is considered by default that it is .W
- .L — the address has a size of **32 bits** . As far as I understand, this is necessary for the largest and thickest microcontrollers, for which 64k address space is not enough for all their wealth.

Instructions

The line cannot start with an instruction. If the line does not contain a label, it must start with a space or a tab character. You can put in this place of the line:

- Assembly instruction mnemonic
- Assembler directive
- The name of the macro to be substituted

We will not discuss instruction mnemonics, but we will talk about assembler directives and macros below.

Operands

- Numbers, including addresses
- Lines
- Current value of the program counter. The "asterisk" symbol is used for this. **JPF *** , for example. If you set the number representation format to any other than MOTOROLA, then there will be no asterisk, but a dollar sign - $
- Arithmetic expressions

Numbers are specified in decimal or hexadecimal form, and can also be in binary. By default, the STM assembler uses the MOTOROLA format to represent numbers. You can change it to the INTEL, TEXAS or ZILOG format. For this, there are the INTEL, MOTOROLA, TEXAL, ZILOG directives. The directives work until another one is declared.
In STM assembler records, if a number is an address, then the address is written without any prefixes or suffixes. And if it is a constant, then the symbol is placed before the constant. Here is a fragment of code from the previous article , copying part of the program from flash to RAM

```
;... всякие инициализации и переключения мы опускаем
        ;Переносим исполняемый код в SRAM
        LDW     X,#Main            ;По метке Main начинается тот самый ц
;обратите внимание на символ решётки перед указанием метки!
        LDW     Y,#$0100           ;По адресу 0100h (это в RAM) будет на
CopyToRAM:
        LD              A,(X)
        LD              (Y),A
        INCW    X
        INCW    Y
        CPW             X,#AfterMain
        JRULT   CopyToRAM

        JPF     $0100        ;Передаём управление в копию кода в RAM

;...

Main:
        MOV PC_ODR,#0
        MOV PC_ODR,#128
        MOV PC_ODR,#0
        MOV PC_ODR,#128
```

Vga → ROPS (Rem Object Pascal Script) - Embedded interpreter of the Pascal language. Plugin PSImport_Classes 3 → Algorithms and software solutions

Full broadcast | RSS

1-Wire Altera arduino ARM Assembler Atmel AVR C++ compel DIY enc28j60 ethernet FPGA gcc I2C IAR KEIL LaunchPad LCD led linux LPCXpresso MSP430 nxp PCB PIC pinboard2 RS-485 RTOS STM32 STM8 STM8L TI UART USB algorithm assembler ADC library power unit detail display idea tool contest competition2 LUT microcontrollers for beginners review Debug board soldering iron printed circuit board pay FPGA crafts purchases programmer programming Light-emitting diode software scheme circuit design Technologies smart House photoresist freebie crap Watch humor

Blogs

Top

AVR	38.98
STM8	37.92
Garbage truck	29.53
STM32	28.46
Detail	24.63
Connection of hardware to the computer.	24.04
Circuit design	18.15
Smart House	17.75
MSP430	17.13
LPC1xxx	14.79

All blogs

```
        MOV PC_ODR,#0
        MOV PC_ODR,#128
        MOV PC_ODR,#0
        MOV PC_ODR,#128
        jra Main            ;Тут специально команда относительного перех
AfterMain:        ;Метка стоит сразу после нужного фрагмента кода
```

Strings are enclosed in double quotes. **Single characters** are enclosed in single quotes. Single quotes can contain one character (1 byte), two characters (2 bytes) or as many as 4 characters (4 bytes), but no more. In addition, the following service characters are used, which are well known to Sei fans:

- '\0' is an empty character. ASCII code 0
- '\b' — backspace. ASCII code 8
- '\t' — horizontal tab. ASCII code 9
- '\n' — new line. ASCII code 10
- '\f' — new page. ASCII code 12
- '\r' — carriage return. ASCII code 13
- '\"' — double quote. ASCII code 34
- '\' — single quote. ASII code 39
- '\\' — fractional line. ASCII code 92

Arithmetic expressions can contain numbers, labels, brackets, and operators. Parentheses are understood as round "()" and curly "{}". Curly brackets enclose the entire arithmetic expression containing inner brackets. Up to 8 levels of nesting are allowed. Operators are executed in 4 queues.

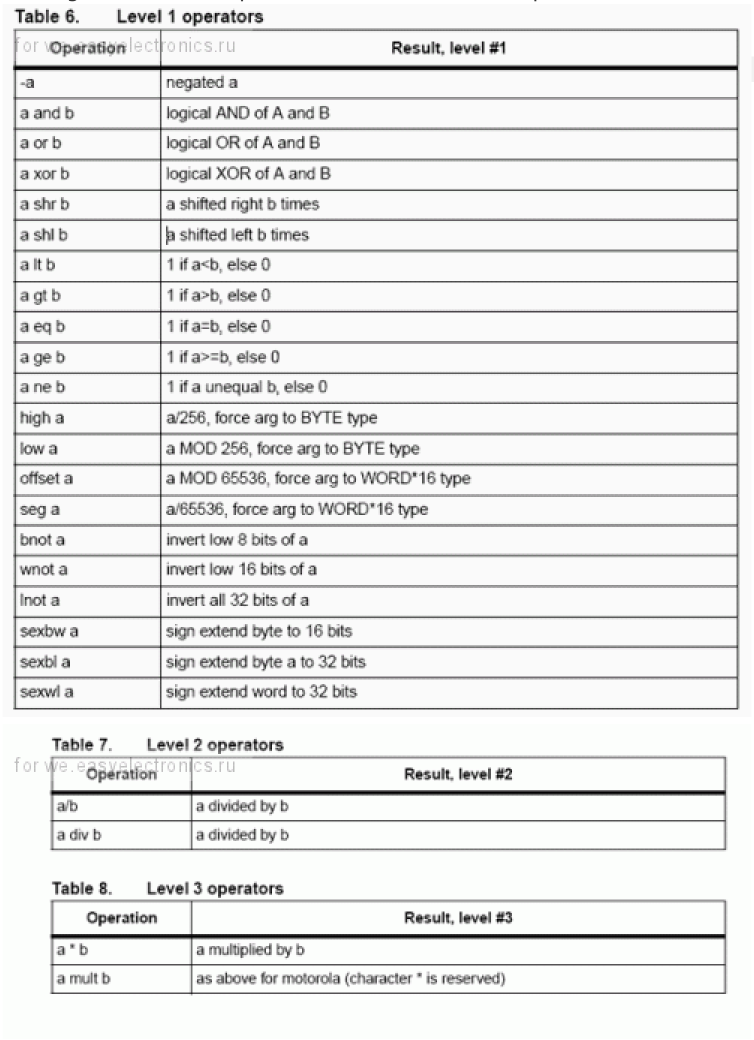

Table 6. Level 1 operators

Operation	Result, level #1
-a	negated a
a and b	logical AND of A and B
a or b	logical OR of A and B
a xor b	logical XOR of A and B
a shr b	a shifted right b times
a shl b	a shifted left b times
a lt b	1 if a<b, else 0
a gt b	1 if a>b, else 0
a eq b	1 if a=b, else 0
a ge b	1 if a>=b, else 0
a ne b	1 if a unequal b, else 0
high a	a/256, force arg to BYTE type
low a	a MOD 256, force arg to BYTE type
offset a	a MOD 65536, force arg to WORD*16 type
seg a	a/65536, force arg to WORD*16 type
bnot a	invert low 8 bits of a
wnot a	invert low 16 bits of a
lnot a	invert all 32 bits of a
sexbw a	sign extend byte to 16 bits
sexbl a	sign extend byte a to 32 bits
sexwl a	sign extend word to 32 bits

Table 7. Level 2 operators

Operation	Result, level #2
a/b	a divided by b
a div b	a divided by b

Table 8. Level 3 operators

Operation	Result, level #3
a * b	a multiplied by b
a mult b	as above for motorola (character * is reserved)

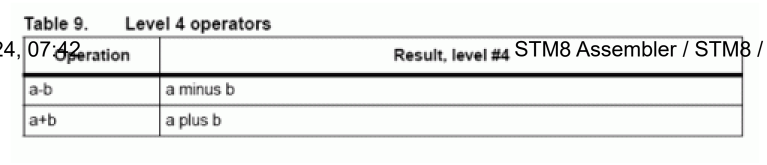

Table 9. Level 4 operators

Operation	Result, level #4
a-b	a minus b
a+b	a plus b

And it is always required to enclose arithmetic expressions in brackets.

```
#define SIZE 128
DS.W SIZE+1        ;Так выдаст ошибку
DS.W {SIZE+1}      ;А так не выдаст
```

Comments start with a semicolon. There are no other options. Does not recognize multi-line comments.

Memory segmentation

The STM8 assembler divides memory into segments. There can be up to 128 segments within a project. Individual segments contain program codes, data in RAM, and various tables in FLASH or EEPROM memory. In addition, no one forbids us to split an already declared segment into several more.

The segment declaration looks like this:

```
[<name>] SEGMENT [<align>] [<combine>] '<class>' [cod]
```

<name> is an optional parameter. It can be up to 12 characters long. When used, the linker will group segments with the same name within a class ('<class>').
<align> is an optional parameter that tells the linker how to align addresses. Options:
byte — no alignment
word — two-byte aligned ($1001 -> $1002)
long — four-byte aligned ($1001 -> $1004)
para — 16-byte paragraph aligned ($1001 -> $1010)
64 — 64-byte aligned ($1001 -> $1040)
128 — 128-byte aligned ($1001 -> $1080)
page — 256-byte page aligned ($1001 -> $1100)
1k — kilobyte aligned ($1001 -> $1400)
4k — 4-kilobyte aligned ($1001 -> $2000)
<combine> is an optional parameter that describes the location of the segment in the address space. Options:
at — used once when the segment is first declared. Specifies the address of the beginning (and optionally) of the end of the segment. Recognizes only hexadecimal notation. Moreover, it does not require any prefixes.

```
        segment at 0-FF 'ram0'
        segment at 100-7FF 'ram1'
```

common — indicates that this segment has common addresses with another. Here is an example from the author's description of this assembler:

```
dat1 segment byte at: 10 'DATA'
ds.w
com1 segment common 'DATA'
.lab1 ds.w 4
com1 segment common 'DATA'
.lab2 ds.w 2
com2 segment common 'DATA'
.lab3 ds.w
```

```
com2 segment common 'DATA'
.lab4 ds.w 2
dat2 segment 'DATA'
.lab5 ds.w 2

;The values for labels lab1, lab2, lab3, lab4, and lab5
;are 12, 12, 1A, 1A and 1E, respectively.
```

You can't write at and *common* on the same line at the same time , so first write the declaration with the keyword *AT* , and then everything with the word *COMMON* .

```
com1 segment byte at: 10 'DATA'
com1 segment common 'DATA'
;....
com1 segment common 'DATA'
```

<cod> is an optional parameter that tells the linker to write the code for this class to a separate file. It has a value from 0 to 9.

In addition , when storing code in one segment and executing it from another, you can tell the assembler that the code will be executed from a different location than it is stored. Otherwise, the first absolute jump to a label located in such code will return the program to the memory area from which the executable code was copied. Again, an example from the author's description

```
segment byte at: 0 'code'
segment byte at: 8000 'ram'
segment 'ram>code'
label1:nop
```

The point of the action is that the assembler with the linker will place this code in one place, and the label label1 will "look" at something completely different. Which will be relevant when we copy the executable code to the place from which we are going to execute it.

Macros

Macros are defined using the MACRO and **MEND** keywords .

```

<название> MACRO [список параметров, под запятую]
           <тело макроса>
           MEND

Пример:
add16   MACRO first,second,result
        ld A,first
        adc A,second
        ld result,A
        MEND
```

The LOCAL keyword indicates that the label declared with it is "local" to the macro. With each subsequent substitution, the assembler will add a number from 0000 to FFFF to the label name. An example from the author's description:

```
getio MACRO
        LOCAL loop
loop ld A,$C000
     jra loop
    MEND
```

There are compiler directives
#IFB — true if the parameter is omitted

#IFDEF — true if the parameter is defined
#IFLAB — true if the parameter is a label
#IFIDN — true if there are two string parameters (after #IFIDN there are spaces)
Also, conditional compilation directives include **#ELSE** and **#ENDIF** , as well as **#IF** , **#IF1** and **#IF2** (**#IF1** works during the first pass of the assembler, **#IF2** — during the second)

I think there is no need to describe conditional compilation here, if necessary, we will write a separate article

---

The assembler directives, as well as instruction mnemonics and their hexadecimal representation will come later. I want to format them into a HELP file, not an article. I am sure it will be more convenient to use!

STM8 , assembler

+5 January 26, 2012, 20:17 **Deer**

Comments (56)

RSS Collapse / Expand

Hmm... After AVR and MSP430 it's really strange... However, STM is a "non-assembler" architecture. They all push higher levels, as far as I can see. 0

_YS_
January 26, 2012, 21:12

Well, they used to say about AVR that their instruction system is optimized for the HVA... Only the organization is so different that in order for the HVA to be able to fully utilize all the features of the processor, it must be very smart... 0

Deer
January 26, 2012, 21:17

No, that's not what I'm talking about. The AVR instruction set *is optimized* for the HLA. But the syntax (not even the instruction set itself, but the assembler syntax) is still built in such a way that it's quite convenient for a human to write.

But in ST, apparently, they simply forgot about the convenience of developing in ASME for a human, and focused on the understandability of the HLA for translators. 0

_YS_
January 26, 2012, 21:21

Ah, well, that's true! I've been swearing for a long time here!
But that's a reason to write another extremely interesting article a little later! :) 0

Deer
January 26, 2012, 21:25

Well, come on, come on. :) 0

_YS_
January 26, 2012, 21:27

It will be, it will be!

I'm gathering material for it :) 0

Deer
January 26, 2012, 21:29

> *But in ST, as you can see, they simply ignored the convenience of developing in ASME*

0

... IMHO it would be more correct to say that ST simply gave up... and not only on assembler :) As for the command system - the story comes from the "two approaches" of Intel & Motorola and all sorts of JVUs have nothing to do with it.

ChipKiller
January 27, 2012, 19:57 ↥

The manual says that it is designed for C/C++ 0

angel5a
January 27, 2012, 08:20 ↥

It's written about AVR too. And it's very noticeable there. But, as **_YS_** and I discussed, the assembler syntax here is much crazier! If the same AVR, and not only them, has, as DI said, a simple as a shovel - you take it and dig, then here it's something more complicated than a shovel, but still far from a bulldozer :) 0

Deer
January 27, 2012, 08:32 ↥

In "PM0044" I came across the JRF and JRT instructions. Under what conditions does the transition take place? 0

coocos
January 26, 2012, 21:27

JRT and JRF? Hmm... interesting commands! Both are conditional jumps. BUT in JRT the jump will always **happen** , and in JRF it will always fail **.** 0

Deer
January 26, 2012, 21:29 ↥

Rave! 0

coocos
January 26, 2012, 21:30 ↥

Nonsense or not, but the ST Microelectronics engineers have a rich imagination, we all knew that before! :) 0

Deer
January 26, 2012, 21:32 ↥

It feels like the set of commands was generated by some tool on full autopilot, along with a description of the processor. +1

_YS_
January 27, 2012, 12:34 ↥

Or maybe (holding my breath) the entire ST Microelectronics company was generated automatically? :) 0

Deer
January 27, 2012, 20:37 ↥

In fact, STMicroelectronics is the hardware embodiment of Skynet... +1

_YS_
January 27, 2012, 10:39 PM ↥

o_O 0

_YS_
January 26, 2012, 21:31 ↥

Moreover, the JRT and JRA (relative unconditional jump) mnemonics have the same opcode: 20h, i.e., in essence, this is the same command! 0

Deer
January 26, 2012, 21:34 ↥

Makes sense. But what is JRT?

0

coocos
January 26, 2012, 21:36 ↑

JRxx is Jump Relative if xx
JRA is Jump Always
JREQ is Jump if Equal
JRNE is Jump if Not Equal
etc. By this logic,
JRT - Jump if True
JRF - Jump if False

0

Deer
January 26, 2012, 21:38 ↑

Well, yes. I saw it in the docs =).

0

coocos
January 26, 2012, 21:41 ↑

And JRF? NOP?) Or just doesn't assemble?

0

Vga
January 26, 2012, 11:32 PM ↑

But no! **The NOP** opcode is 9D, **the JRF** opcode is 21. That is, they are different!

0

Deer
January 26, 2012, 11:35 PM ↑

Funny. Does any other instruction have opcode 21?

0

Vga
January 26, 2012, 11:37 PM ↑

No. No one else has it. All JRxx start with 2, and there are no other repeating ones. Just in case, I methodically reviewed the entire description of the command system. It doesn't appear anywhere else!

0

Deer
January 26, 2012, 11:44 PM ↑

Maybe to "mute" transitions on the fly in the debugger?

0

angel5a
January 27, 2012, 08:23 ↑

Who the hell knows! Nobody forbids us from changing flags during debugging!

0

Deer
January 27, 2012, 08:40 ↑

JRF Opcode - 21h

0

N1X
January 27, 2012, 10:27 ↑

sorry, I looked in the wrong place =) JRT and JRA were compared...

0

N1X
January 27, 2012, 10:29 ↑

I was thinking and thinking too, looked at the opcodes, there is a suspicion that, firstly, there were extra bits left (for systemicity, for "prettyness"), and secondly, a reserve for the future for pipeline optimization and branch prediction (so that, for example, commands from the current branch and the one specified on the JRF are

0

loaded into the cache or onto the pipeline. For example, to prepare for a possible transition when the bus is not yet busy. That is, an additional opportunity for optimization. On the other hand, it is not logical: there were transitions for which single-byte opcodes were not enough, it would be better to allocate this opcode for them.

Pitty
December 15, 2016, 19:33 ↑

Useful article. I was too lazy to look for documentation about expressions.

+1

coocos
January 26, 2012, 21:35

Please provide another article about the STM8 core architecture and its address spaces.

0

Vga
January 26, 2012, 11:33 PM

Okay. We'll take your request into account!

0

Deer
January 26, 2012, 11:34 PM ↑

The assembler built into STVD is the most buggy piece of crap I've ever encountered. If you have a lot of extra time and a desire to find more and more bugs, then you've found what you were looking for :)... if you need to write working programs, then it's better to use assembler from Cosmik, IAR, etc.

+1

ChipKiller
January 27, 2012, 12:40

> *if there is a need to write working programs, then it is better to use*

Si

> *from Cosmik, IAR, etc.*

+2

angel5a
January 27, 2012, 12:53 PM ↑

... ST and Cosmica assemblers are not compatible, so it would be nice for the author to mention which assembler is being discussed. I can agree about C, but sometimes it is easier to write some "exotic" procedure in assembler and include it in a C program, so knowledge of assembler will not hurt IMHO...

0

ChipKiller
January 27, 2012, 1:18 PM ↑

> *but sometimes it is easier to write some "exotic" procedure in assembler and include it in a C program*

Well, they will automatically take the assembler that comes with C, i.e. IAR/Cosmic.
By the way, why does no one remember Raisonance?

0

Vga
January 27, 2012, 5:58 PM ↑

> *Well, they will automatically take the assembler that comes with C.*

will take, but "automatic" will not work due to incompatibility of assemblers. When using assembler, you often have to access special. registers (the description of which is in the folder \asm\include\*.asm). In order not to manually type the names of special. registers, I wrote a utility that converts the files \asm\include\*.asm from ST into a format understandable to Cosmic st2cosmic.rar

0

ChipKiller
January 27, 2012, 19:49 ↑

0

Why did they make so many variants? To confuse the user?

And how can you make different assemblers? These are just mnemonics assigned to opcodes that are firmly embedded in the hardware.

maxgrind
March 26, 2012, 10:19 ↑

Вот так и можно сделать, что архетиктуры у железа разные и наборы опкодов, и их доступность.
И множество компиляторов, одни под конкретную архетиктуру, другие стремились охватить как можно больше архетиктур «Чтобы не запутать пользователя» при миграции между архетиктурами.

+1

angel5a
26 марта 2012, 11:17 ↑

> *архетиктуры у железа разные и наборы опкодов, и их доступность.*

я говорю про разные варианты ассемблеров одного и того же железа. Вот ChipKiller писал:

> *Встроенный в STVD ассемблер — самая глючная х… ня, которую доводилось встречать. Если у Вас много лишнего времени и есть желание находить все новые и новые баги, то Вы нашли, что искали :)… если есть необходимость писать рабочие программы, то лучше использовать ассемблер от Cosmik,IAR и т.д.*

Не понимаю все равно. Смультиплексировать линии на такой то адрес в памяти и подать строб чтения можно одним единственным способом — на кристалле же уже все сформировано. А как мы назовем код операции уже забота ассемблера. Максимум, что может быть — это разница в написании самой мнемоники: JMP или BR, например, или требование всяких отступов, наличие лишней строчки в конце программы, директивы препроцессора. Но сам подход к общению с процом и периферией то не должен отличаться.
Или все отличия асма STVD от остальных и заключается в этих формальностях типа лишних отступов, других директив и прочей хрени?

0

maxgrind
26 марта 2012, 12:51 ↑

> *я говорю про разные варианты ассемблеров одного и того же железа.*

Мой пост состоит из 2-х частей. Первая объясняет почему появилась вторая:

> *И множество компиляторов, одни под конкретную архетиктуру, другие стремились охватить как можно больше архетиктур «Чтобы не запутать пользователя» при миграции между архетиктурами.*

0

angel5a
26 марта 2012, 13:30 ↑

> *Или все отличия асма STVD от остальных и заключается в этих формальностях*

… директивы — это основа ассемблера и вовсе не «формальности»

0

ChipKiller
26 марта 2012, 13:42 ↑

Товарищи, использующие STVD, подскажите как там объявлять переменные, и указывать их размер. И как потом узнать где он размещает переменную, что бы следить за ней во время отладки?

0

Alexl82
26 марта 2012, 08:41

STVD в «чистом» виде лучше не использовать — ибо очень кривой ассемблер. Скачайте Cosmic или что нибудь другое — избавите себя от кучи неприятных

0

неожиданностей…

ChipKiller
26 марта 2012, 13:45 ↑

Скачал Cosmic, по ассемблеру для него то же информации не много…
Полноценного мануала даже не нашел под STM8. 0

Alexl82
26 марта 2012, 14:55 ↑

после установки Cosmica, загляните в папку Docs — там есть три прекрасных pdf_a 0

ChipKiller
26 марта 2012, 17:23 ↑

Можно пару самых злостных костылей для примера? А то аж интересно стало. +1

maxgrind
26 марта 2012, 16:20 ↑

Один прикол и я могу поведать, правда не знаю откуда ноги растут у проблемы. Ставлю предделитель таймера, запуская таймер, а таймер работает без предделителя. Но досчитав до конца, снова начинает счет уже с предделителем. 0

Alexl82
26 марта 2012, 16:49

The STVD assembler incorrectly generates jump and call addresses, and as for the debugger, it can be "dropped" even by accident… 0

ChipKiller
March 26, 2012, 5:26 PM ↑

This is not a bug but a feature. There is a "caching" flag. 0

angel5a
March 26, 2012, 5:35 PM ↑

maybe so, but after working on Cosmic, I no longer wanted to use the built-in STVD assembler. The only inconvenience was with using .inc files - I wrote a converter utility and now there are no problems with this... 0

ChipKiller
March 26, 2012, 5:51 PM ↑

"I want to format them into a HELP file, not an article." A year has passed, where is the promised HELP? 0

anakost
09 April 2013, 09:18

But did anyone promise? It says "I want", and you can want forever. 0

angel5a
April 12, 2013, 11:35 ↑

The author writes "I want to format them into a HELP file, not into an article."
And before that, "... will come later." 0

Logic
April 12, 2013, 11:54 ↑

Well, it will be later. Much later :) 0

angel5a
April 12, 2013, 11:57 ↑

Only registered and authorized users can leave comments.