

Документация к классу TAsmTextParser

Версия 1.1

Дата версии файла: 11/12/2020

Изменения:

11/12/2020 – Добавлено правило Section

20/11/2020 – Добавлено правило PartsOne

Vitaly V Gorbukov

gorbukov@yandex.ru

Оглавление:

Система обработки (разбора) текста

Правила обработки текста

TOKENLIST – список токенов

TOKENONE – токен как вариант из заданных в списке

TOKENMAYBE – токен который может отсутствовать

TOKENSPLIT – объединение нескольких токенов в один

TOKENPARTS – токен состоящий из нескольких частей

PARTSONE – обязательная часть токена

PARTSMAYBE – часть токена которая может отсутствовать

EXPRESSION – числовое выражение

EXPRESSIONLIST – список числовых выражений через запятую

REGISTER – регистр

REGISTERLIST – список регистров через запятую

DEFINESYMBOL – определение символа программы

STRING – строка в кавычках

STRINGLIST – строки в кавычках через запятую

SECTION – обработка директивы ассемблера .section

Система обработки (разбора) текста

Система базируется на последовательной проверки введенного текста некоторым набором стандартных правил.

Текст на языке ассемблера разбирается построчно. Для этого строка разбивается на токены символов по следующим типам:

```
TTokenType=(  
  
ttNone,      // первоначальное значение  
ttErr,       // ошибка распознания токена (нет второй кавычки)  
ttSpace,     // пробелы и табы  
ttDelim,     // разделитель          [ , = ; _ \ ` # № ? ]  
ttLabDelim,  // разделитель меток    [ : ]  
ttRem,       // строчный комментарий [ @ ]  
ttLogic,     // лог. и арифм. операции [ ! << >> | || & && - + ~ * / % ^ ]  
ttIf,        // операции сравнения   [ == != > < <> >= <= ]  
ttGroup,     // токены группы          [ ( ) [ ] { } ]  
ttString,    // строки заключенные в кавычки [ " " ]  
ttChars,     // токен состоящий из букв и цифр  
ttOneChar,   // токен состоящий из одного символа в одинарных кавычках  
ttEdDir      // директива редактора    [ @. ]  
);
```

Дальше работа идет с токенами в зависимости от их типа и [правил](#) по следующим шагам:

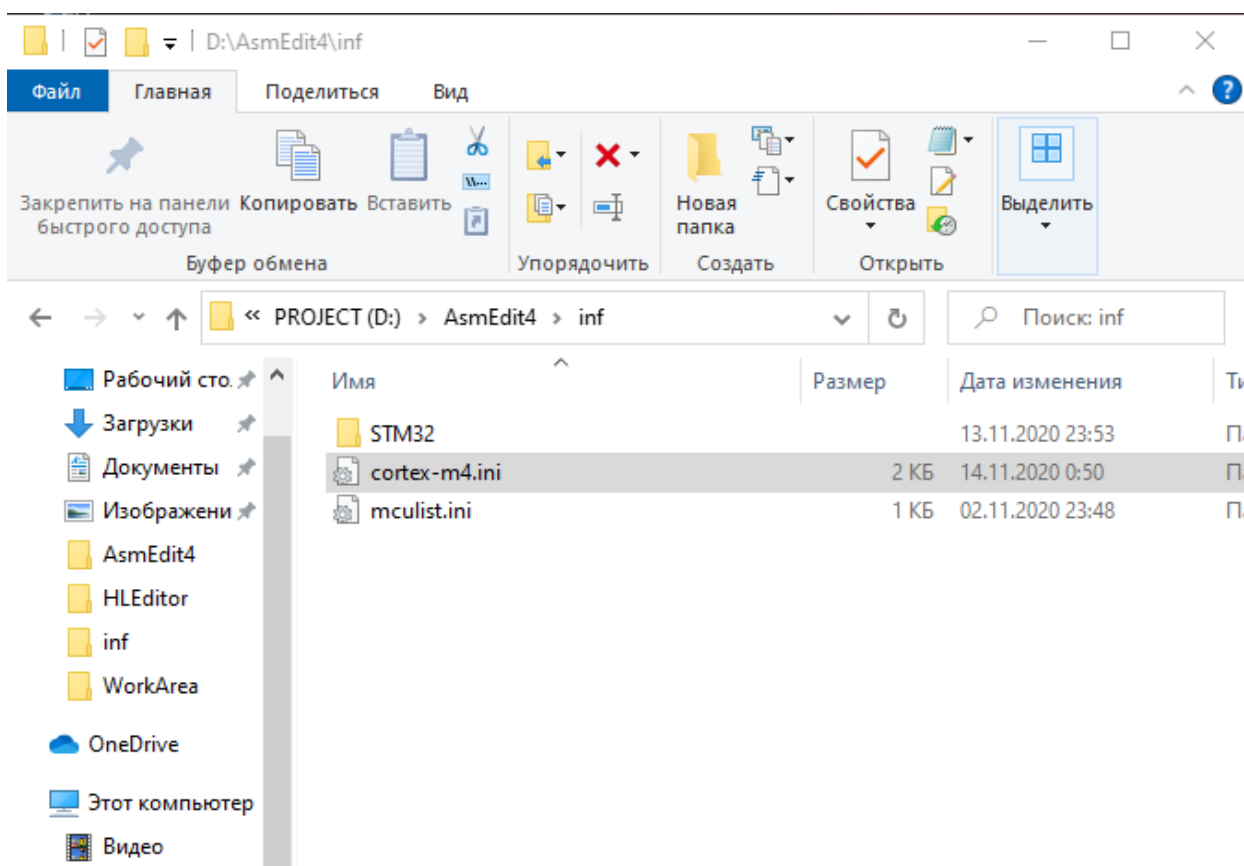
1. Выделяются метки. Меткой считается строка завершающаяся двоеточием, например: **Label:**
Метка не должна начинаться с числа, допустимые символ начала метки кроме букв: “.”, “_”, “\$” (точка, подчеркивание, знак доллара)
Строка может содержать несколько меток
2. Проверяется наличие строчного комментария, строчные комментарии в **gnu-arm-as** начинаются с **@** и действуют до конца строки
3. Проверяется наличие директив редактора. Директивы редактора начинаются символами **@.** Оставшаяся строка правее директивы редактора считается комментарием
4. Осуществляется поиск [правил](#) для токена.

Правила обработки текста

Правила обработки текста описываются в формате ini-файла

Стандартное имя файла для правил это имя ядра микроконтроллера и расширение ini, например: **cortex-m4.ini**

Файл правил размещается в каталоге **/inf** приложения



Каждое правило разбора имеет имя, которое указывается в квадратных скобках, и является заголовком секции ini-файла, например, правило **.THUMB** должно быть в файле написано как **[.THUMB]**

Регистр написания имени правила может иметь значение ! (см. ниже)

Каждое правило имеет тип которое и определяет способ проверки им текста:

Поддерживаются следующие типы правил:

TOKENLIST - список токенов
TOKENONE - токен как вариант из заданных в списке
TOKENMAYBE - токен который может отсутствовать
TOKENSPLIT - объединение нескольких токенов в один
TOKENPARTS - токен состоящий из нескольких частей
PARTSONE - обязательная часть токена
PARTSMAYBE - часть токена которая может отсутствовать

EXPRESSION - числовое выражение
EXPRESSIONLIST - список числовых выражений через запятую
REGISTER - регистр
REGISTERLIST - список регистров через запятую
DEFINESYMBOL - определение символа программы
STRING - строка в кавычках
STRINGLIST - строки в кавычках через запятую
SECTION - обработка директивы ассемблера `.section`

Тип правила указывается ключевым параметром **type**

Например,

```
[.THUMB]
Type=TokenList
```

Имя типа правила регистронезависимо для правил которые применяются к первому обрабатываемому токenu.

Далее правило может иметь параметры которые называются и описываются в зависимости от типа правила.

Обычно параметры имеют порядковый номер начинающийся с нуля, все параметры должны иметь последовательные номера !

Например,

ВЕРНО	НЕ ВЕРНО	Описание ошибки
<pre>[.THUMB] Type=TokenList item0=нулевой параметр item1=первый параметр item2=второй параметр</pre>	<pre>[.THUMB] Type=TokenList item1=нулевой параметр item2=первый параметр item3=первый параметр</pre>	Параметр имеет номер 1 при отсутствии параметра с номером 0
<pre>[.THUMB] Type=TokenList item1=нулевой параметр item2=первый параметр item0=второй параметр</pre>		
<pre>[.THUMB] Type=TokenList item2=нулевой параметр item0=первый параметр item1=второй параметр</pre>		

Чаще всего все правила имеют параметр **itemNN**, где NN порядковый номер параметра

Параметр **itemNN** может быть строкой заключенной в кавычки (двойные!) или указывать на другое правило, которое должно начинаться с символа \$ (доллар), например:

Текст в файле правил	Описание
<code>[.THUMB]</code>	Имя правила
<code>Type=TokenList</code>	Тип правила
<code>item0=".THUMB"</code>	Параметр указывает на строку
<code>item1=\$RuleName</code>	Параметр указывает на правило \$RuleName

Для каждого правила itemNN обычно существует второй параметр **itemNNtype** который может принимать следующие значения:

REM - примечание
ASMDIR - директива ассемблера
EDDIR - директива редактора
PARAM - параметры директив редактора или ассемблера
ASMCOM - команда ассемблера
DELIM - разделитель
LABSYMB - символ / метка
REG - регистр
NUM - число
STRING - строка

Фактически тип **itemNNtype** указывает как тот или иной токен текста после обработки будет отображаться редактором (стиль текста: цвет и стиль шрифта, цвет фона)

Алгоритм выбора правила для обработки токенов строки следующий:

В токене для разбора (то есть после проверки токена на то, что он не является: меткой, комментарием, директивой редактора) ищется вхождение имени правила с первого символа

Например для правила **.THUMB** это вхождение для следующих токенов: **".THUMB"**, **".THUMBA"**, **".THUMBHGG"**, а также для всех токенов начинающихся с **".thumb"** (нижний регистр или *pA3ноРегистр* символов). То есть вхождение регистронезависимо.

Для токенов **"A.THUMB"**, **"kA.THUMB"**, **"0A.THUMB"** – вхождения не будет !

После обнаружения вхождения правила в токен он обрабатывается внутри правила в соответствии с определенными в нем алгоритмами

Пример строки для разбора:

.label: .Thumb @ remark

Строка будет разбита на токены: **".label"**, **":"**, **".Thumb"**, **"@ remark"**

Далее в тексте строки произойдет обработка имени метки **.label** и для токена **.Thumb** будет произведен поиск правила **[.THUMB]** из ini-файла Правила обработки текста **cortex-m4.ini**

Дальнейший разбор текста будет передан в обработчик правила **[.THUMB]** и после его обработки будут обрабатываться оставшиеся токены строки, в нашем случае это токен комментария **"@ remark"**

TOKENLIST – список токенов

Правило задает список токенов, которые должны быть найдены.

Общий формат правила

```
[Имя_правила]  
Type=TOKENLIST  
item0="Строка сравнения"  
item0type=ASMCOM  
item1=$вложенное_правило  
..  
itemNN="СтрокаNN"
```

например, для обработки директивы компилятора **.thumb** правило должно быть описано следующим образом:

```
[.THUMB]  
type=TOKENLIST  
item0=".THUMB"  
item0type=ASMDIR
```

Логика обработки правила:

у правила **.THUMB** только один токен равный **".THUMB"** (регистронезависимо!), при совпадении для этого токена применяется стиль [ASMDIR](#) – директива редактора

Поскольку иных токенов правило не обрабатывает, происходит возврат в вызывающую процедуру для последующей обработки последующих токенов строки (если они еще есть)

TOKENONE – токен как вариант из заданных в списке

Правило задает список вариантов написания токена. Применяется при существовании вариантов описания токенов правила.

Общий формат правила

```
[Имя_правила]
Type=TOKENONE
item0="Строка1"
item1="Строка2"
item2="Строка3"
..
itemNN="СтрокаNN"
```

Использование правила проще показать на примере правила обрабатывающего директиву ассемблера с параметром: ***.syntax divided/unified***

У директивы ***.syntax*** параметр может принимать два значения **divided** или **unified**

Соответственно, чтобы описать эту директиву нам необходимо создать правило, которое задаст несколько токенов текста строки:

Правила обработки	Описание
<pre>[.SYNTAX] type=TOKENLIST item0=".SYNTAX" item0type=ASMDIR item1=\$syntax_param item1type=PARAM</pre>	Обрабатываем список токенов по правилам типа TOKENLIST Первым токеном должен быть токен с текстом ".SYNTAX" (регистронезависимо!), которому при совпадении присваивается стиль ASMDIR Второй токен обрабатывается правилом <i>\$syntax_param</i> и в случае успеха токеноу присваивается стиль PARAM
<pre>[\$syntax_param] type=TOKENONE item0="UNIFIED" item1="DIVIDED"</pre>	Обрабатываем вариант токена, им может быть строка UNIFIED или DIVIDED

ВАЖНО !

Правила обработки текста не могут содержать ссылки на правила которые описаны ниже их вызова !! в примере выше, в файле правил, сначала должно быть описано правило [\$syntax_param], и только потом [.SYNTAX]

TOKENMAYBE – токен который может отсутствовать

Правило задает токен или правило которое может отсутствовать и будет пропущено при анализе без генерации ошибки разбора.

Общий формат правила

```
[Имя_правила]  
Type=TOKENMAYBE  
item0="Строка1"
```

или

```
[Имя_правила]  
Type=TOKENMAYBE  
item0="$правило"
```

ВАЖНО !

У данного правила может быть задан только один параметр **item0**.

TOKENSPLIT – объединение нескольких токенов в один

Токен позволяет обрабатывать сложные токены которые [на этапе токенизации строки](#) были разделены на несколько последовательно идущих токенов.

Общий формат правила

```
[Имя_правила]
Type=TOKENSPLIT
item0="Строка1"
item1="Строка2"
item2="Строка3"
..
itemNN="строкаNN"
```

Пример использования правила

При описании директивы компилятора **.cpu** необходимо описание параметра **cortex-m4**.

Однако по заложенным [правилам токенизации](#) эта подстрока будет разбита на 3 отдельных токена: **"cortex"**, **"-"**, **"m4"**. Соответственно при попытке описания этого параметра правилом [TOKENLIST](#) мы получим разрешенное написание в тексте программы с пробелами между частями **"cortex - m4"** что однако будет воспринято компилятором как ошибка, так как он ожидает только параметр написанный слитно.

Для того чтобы избежать неправильной трансляции необходимо использовать правило **TOKENSPLIT** который проверит последовательное написание всех трех токенов без пробелов между ними, после этого объединит их в один, случае же наличие где либо между токенами пробела – будет возвращена ошибка

```
[$cpu_param]
Type=TOKENSPLIT
item0="cortex"
item1="-"
item2="m4"
```

TOKENPARTS – токен состоящий из нескольких частей

Тип правила позволяющий описать сложный токен состоящих из различных подстрок

Фактически тип правила TOKENPARTS это правило TOKENLIST только действующее внутри одного токена

Общий формат правила

```
[Имя_правила]
Type=TOKENPARTS
item0="ПодСтрока1"
item1="ПодСтрока2"
item2="ПодСтрока3"
..
itemNN="ПодСтрокаNN"
```

Применяется для описания таких команд ассемблера ARM в которых в текст команды входит указание на размерность операции и/или флаги исполнения

Например, для команды LDR

LDR and STR, immediate offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

Syntax

```
op{type}{cond} Rt, [Rn {, #offset}]; immediate offset
op{type}{cond} Rt, [Rn, #offset]!; pre-indexed
op{type}{cond} Rt, [Rn], #offset; post-indexed
```

Как видно команда LDR может иметь вариативное написание к которому добавляются символы {type} и {cond} которые определены следующим образом:

- 'type' is one of the following:
 - B: Unsigned byte, zero extends to 32 bits on loads
 - SB: Signed byte, sign extends to 32 bits (LDR only)
 - H: Unsigned halfword, zero extends to 32 bits on loads
 - SH: Signed halfword, sign extends to 32 bits (LDR only)
 - : Omit, for word

То есть команда как минимум может имеет вид **LDRB** - загрузка одного байта или например **LDRH** – загрузка полуслова (16 бит) с заполнением нулями слева

Для символа команды {cond} список вариантов еще более внушительен:

Table 24. Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned \geq
CC or LO	C = 0	Lower, unsigned $<$
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned $>$
LS	C = 0 or Z = 1	Lower or same, unsigned \leq
GE	N = V	Greater than or equal, signed \geq
LT	N \neq V	Less than, signed $<$
GT	Z = 0 and N = V	Greater than, signed $>$
LE	Z = 1 and N \neq V	Less than or equal, signed \leq
AL	Can have any value	Always. This is the default when no suffix is specified.

Правило **TOKENPARTS** позволяет проводить обработку подстрок именно внутри токена

Текст правила	Описание
<i>[\$ldr_oper]</i>	Имя обрабатываемого правила
<i>Type=TOKENPARTS</i>	Тип правила
<i>item0="LDR"</i>	Токен должен начинаться с подстроки LDR
<i>item1=\$type</i>	обрабатывается указание на размер операции в правиле \$type
<i>item2=\$cond</i>	Обрабатывается условие исполнение операции в правиле \$cond

В данном примере обработка размера операции и обработка условия исполнения вынесены в отдельное правило, так как указание на размер или условие могут иметь варианты или могут отсутствовать одно или оба. В данном случае для обработки будет применено правило типа [**PARTSMAYBE**](#)

PARTSONE – обязательная часть токена

Тип правила позволяющий описать вариант части сложного токена состоящего из различных подстрок

Фактически тип правила PARTSONE это правило TOKENONE только действующее внутри одного токена

Общий формат правила

```
[Имя_правила]  
Type=PARTSMAYBE  
item0="Вариант1"  
item1="Вариант2"  
item2="Вариант3"  
..  
itemNN="ВариантNN"
```

PARTSMAYBE – часть токена которая может отсутствовать

Тип правила позволяющий описать вариант части сложного токена состоящего из различных подстрок

Фактически тип правила PARTSMAYBE это правило TOKENMAYBE только действующее внутри одного токена

Общий формат правила

```
[Имя_правила]
Type=PARTSMAYBE
item0="Вариант1"
item1="Вариант2"
item2="Вариант3"
..
itemNN="ВариантNN"
```

Для примера опишем [символ {type} команды LDR](#) из примера к правилу [TOKENPARTS](#)

```
[$type]
Type=PARTSMAYBE
item0="B"
item1="SB"
item2="H"
item3="SH"
```

И [символ {cond} команды LDR](#)

```
[$cond]
type=PartsMayBe
item0="EQ"
item1="NE"
item2="CS"
item3="HS"
item4="CC"
item5="LO"
item6="MI"
item7="PL"
item8="VS"
item9="VC"
item10="HI"
item11="LS"
item12="GE"
item13="LT"
item14="GT"
item15="LE"
item16="AL"
```

Такое задание частей токена совместно с правилом TOKENPARTS позволяет описывать сложные составные токены в которых части могут отсутствовать все или частично

Описание инструкции LDR команды с дополнительными условиями выглядит следующим образом:

```
[$type]  
Type=PARTSMAYBE  
item0="B"  
item1="SB"  
item2="H"  
item3="SH"
```

```
[$cond]  
type=PARTSMAYBE  
item0="EQ"  
item1="NE"  
item2="CS"  
item3="HS"  
item4="CC"  
item5="LO"  
item6="MI"  
item7="PL"  
item8="VS"  
item9="VC"  
item10="HI"  
item11="LS"  
item12="GE"  
item13="LT"  
item14="GT"  
item15="LE"  
item16="AL"
```

```
[$ldr_oper]  
Type=TOKENPARTS  
item0="LDR"  
item1=$type  
item2=$cond
```

ВАЖНО !!

В описании правил TOKENPARTS и PARTSMAYBE не используются параметры itemNNtype так как выделение частей токена не осуществляется. Стиль обрабатываемого токена должен быть определен в вызывающем правиле.

EXPRESSION – числовое выражение

Правило осуществляет анализ числового выражения

Общий формат правила

```
[Имя_правила]  
Type=EXPRESSION  
min=MinValue  
max=MaxValue  
size=BitCount
```

Значение проверяется на нахождение в диапазоне от minValue до maxValue (оба крайних значения включаются), либо на размер значения в количестве бит (8,16,32)

ПРИМЕЧАНИЕ!

В текущей версии программы проверка диапазона не выполнена.

Обрабатываются выражения включающие в себя как непосредственно числовые значения, так и значения символов. Значение меток вычисляются на этапе компиляции поэтому при их наличии в выражении проверка не производится

EXPRESSIONLIST – список числовых выражений через запятую

Правило описывает числовые выражения идущие через запятую, как это обычно происходит в директивах **.word .hword .byte**

Общий формат правила

```
[Имя_правила]  
Type=EXPRESSIONLIST  
min=MinValue  
max=MaxValue  
size=BitCount
```

Параметры аналогичны EXPRESSION.

REGISTER - регистр

Правило определяет регистр. В списке itemNN определяются базовые регистры, которые **НЕ ДОЛЖНЫ** быть в обрабатываемом тексте

Имена регистров: R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 LR SP PC

Общий формат правила:

```
[имя_правила]  
Type=REGISTER  
itemNN="excludeRegister"
```

Пример, из списка возможных вариантов буду исключены регистры PC (R15) и SP (R13):

```
[$adr_registers]  
Type=REGISTER  
item0="PC"  
item1="R15"  
item2="R13"  
item3="SP"
```

REGISTERLIST - список регистров через запятую

Правило определяет список регистров через запятую, обычно подобное указание встречается в командах PUSH / POP, в параметрах itemNN указываются регистры исключаемые из списка допустимых.

Пример использования:

```
[$adr_registers]  
Type=REGISTERLIST  
item0="PC"  
item1="R15"  
item2="R13"  
item3="SP"
```

DEFINESYMBOL – определение символа программы

Правило задает символ определенный в программе

Общий формат правила:

[Имя_правила]

Type=DEFINESYMBOL

Используется для описания директивы ассемблера **.equ**

[\$symb_define]

type=DefineSymbol

[\$Expression]

type=Expression

[.EQU]

type=TokenList

item0=".EQU"

item0type=ASMDIR

item1=\$symb_define

item2=", "

item3=\$Expression

STRING – строка в кавычках

Правило определяет строку заключенную в двойные кавычки

Общий формат правила:

[имя_правила]
type=String

STRINGLIST – строки в кавычках через запятую

Правило определяет одну или несколько строк заключенных в кавычки. Используется для директив ассемблера .ascii и .asciz

Общий формат правила:

[имя_правила]
type=StringList

SECTION – обработка директивы ассемблера .section

Правило обрабатывает список доступных секций для директивы .section ассемблера. Сами секции определяются в файле .ld проекта, таким образом вне проекта правило не работает и будет возвращать ошибку.
Параметров правило не имеет

Общий формат правила:

[имя_правила]
type=Section