GAS Tutorial - 7. Directives (2)

Young W. Lim

2016-03-12 Sat

Outline

Based on

"Using as", Dean Elsner, Jay Fenlason & friends

In short, it only meaningful to add or subtract the offsets in an address; you can only have a defined section in one of the two arguments.

7.1 .abort

- stops the assembly immediately

7.2 .ABORT (COFF)

- when producing COFF output
- the same as .abort

7.4 .altmacro (1)

Enable alternate macro mode, enabling:
LOCAL name [, . . .]

- generate a string replacement for each of the name arguments
- replace any instances of name
- The replacement string is unique in the assembly
- different for each separate macro expansion.
- LOCAL allows you to write macros that define symbols
- without conflict between separate macro expansions

7.4 .altmacro (2)

String delimiters

- 'string' delimit strings with single-quote characters

Single-character string escape

- to include any single character literally in a string
- prefix the character with '!'

Expression results as strings

- %expr to evaluate expr
- use the result as a string.

7.7 .balign[wl] abs-expr , abs-expr , abs-expr (1)

- pad the location counter to a particular storage boundary
- followed by three absolute expressions

The first expression

- alignment request in bytes
- .balign 8 advances the location counter toward multiples of 8

The second expression

- the fill value to be stored in the padding bytes
- the default fill value is zero

The third expression

- the maximum number of bytes that should be skipped
- no skipping of bytes above the specified maximum

7.7 .balign[wl] abs-expr , abs-expr , abs-expr (2)

.balignw directive

- treats the fill pattern as a two byte word value

.balignl directives

- treats the fill pattern as a four byte longword value

.balignw 4,0x368d

- align to a multiple of 4
- skipping two bytes with the value 0x368d
- skipping 1 or 3 bytes, the fill value is undefined.

7.111 .title “heading “

- use heading as the title
- when generating assembly listings
- (second line, immediately after the source file name and page number)

7.94 .sbttl “subheading “

- use subheading as the title
- when generating assembly listings.
- (third line, immediately after the title line)

7.115 .version “string “

- creates a .note section
- places into it an ELF formatted note of type $NT_{VERSION}$
- The note’sname is set to string

7.86 .print string

- print string on the standard output during assembly
- string in double quotes.

7.118 .warning “string “

Similar to the directive .error (see Section 7.47 [.error “string “], page 52), but just emits a warning.

7.46 .err

- print an error message
- normally, no object file is generated
- unless the '-Z' option was used
- to signal an error in conditionally compiled code.

7.47 .error “string “

- prints with a custom error message string

7.50 .fail expression

- generates an error or a warning
- print a warning message (expression $>=$ 500)
- print a error message (expression $<$ 500)
- print a message including the value of expression
- useful inside complex nested macros or conditional assembly

7.51 .file (1)

Default Version

- tells as that we are about to start a new logical file
- .file string
- string is the new file name

. filename is recognized whether quotes are used or not

7.51 .file (2)

DWARF2 Version

- assigns filenames to the .debug_line file name table
- .file fileno filename
- The fileno
  - must be a unique positive integer
  - index of the entry in the table
- The filename is a string literal
- filename table is shared with the .debug_info section of the DWARF2

debugging information

- the user must know the exact indices of table entries

7.62 .include “file “

- include supporting files at specified points in your source program
- control the search paths used with the ‘-I’ command-line option
- quotation marks are required around file

7.61 .incbin “file “[,skip [,count]]

- includes file verbatim at the current location
- control the search paths used with the ‘-I’command-line option
- quotation marks are required around file
- skip : skips a number of bytes from the start of the file
- count : indicates the maximum number of bytes to read
- no data alignment
- proper user alignment both before and after the incbin directive.

7.69 .line line-number

- change the logical line number
- an absolute argument for the next line
- the next line has that logical line number
- the current line has line number - 1
- associated with the a.out or b.out object-code formats
- as still recognizes it for COFF output

7.77 .macro (1)

- .macro and .endm allow you to define macros
- a macro sum example that puts a sequence of numbers into memory :

```
.macro    sum from=0, to=5
.long     \from
.if       \to-\from
sum       "(\from+1)",\to
.endif
.endm
```

7.77 .macro (2)

- With that definition, ‘SUM 0,5’ is equivalent to this assembly input:

```
.macro    sum from=0, to=5          .long    0
.long     \from                     .long    1
.if       \to-\from                 .long    2
sum       "(\from+1)",\to           .long    3
.endif                              .long    4
.endm                               .long    5
```

7.77 .macro (3)

.macro macname .macro macname macargs ...

- Begin the definition of a macro called macname
- can qualify the macro argument
  - mandate argument (':req')
  - variable argument (':vararg')
- default argument value ('=deflt')
- no redefinition is allowed
- except using .purgem

7.77 .macro (4)

valid .macro examples

```
    .macro comm
```

begin the definition of a macro called comm with no arguments

```
    .macro plus1 p, p1
    .macro plus1 p p1
```

either statement is ok. (comma or just blank)

- definition of a macro called plus1
- two arguments p and p1
- within the definition, \p and \p1 to evaluate the arguments

7.77 .macro (5)

```
.macro reserve_str p1=0 p2
```

- macro called `reserve_str` with two arguments
- p1 has a default value 0
- p2 has no default value
- `reserve_str a ,b`
  - \p1 has a and \p2 has b
- `reserve_str , b`
  - \p1 has 0 and \p2 has b

7.77 .macro (6)

```
.macro m p1:req, p2=0, p3:vararg
```

- a macro called m, with at least three arguments
- p1 is mandatory
- p2 is optional and has default value 0
- p3 is assigned all the remaining arguments

7.77 .macro (7)

When you call a macro, you can specify the argument values either

- by position
- by keyword.

For example,

- `sum 9,17`
- `sum to=17, from=9`

7.77 .macro (8)

special meanings to certain characters can cause problems

- the colon (:) is generally permitted to be part of a symbol name
- no way to differentiate with a label

```
.macro label l
\l:
.endm
```

- might not work as expected
- invoking ‘label foo’ might not create a label called ‘foo’
- instead just insert the text `\l:` into the assembler source,
- probably generating an error about an unrecognised identifier.

7.77 .macro (9)

- similar problems might occur with the period character ('.')
- often allowed inside opcode names (and hence identifier names)

```
.macro opcode base length
\base.\length
.endm
```

- invoking it as `opcode store l` will not create a `store.l` instruction
- instead generate some kind of error as the assembler tries to interpret the text `\base.\length`

7.77 .macro (10)

There are several possible ways around this problem: Insert white space

- white space characters

```
.macro label l
\textbackslash{}l :
.endm
```

- \() for separation

```
macro opcode base length
\base\().\length
.endm
```

7.77 .macro (11)

Use the alternate macro syntax mode

- & used as a separator

```
.altmacro
.macro label l
l&:
.endm
```

Note: this problem of correctly identifying string parameters to pseudo ops also applies to the identifiers used in .irp and .irpc

7.77 .macro (12)

`.endm` mark the end of a macro definition.
`.exitm` exit early from the current macro definition.
`\@`

- pseudo variable representing current number of macros
- can copy that number to your output with `\@`
- only within a macro definition.

`LOCAL name [ , ... ]`

- only available if you select
- alternate macro syntax with
  - `--alternate`
  - `.altmacro`

7.48 .exitm

- exit early from the current macro definition

7.89 .purgem name

- undefine the macro name
- later uses of the string will not be expanded

7.65 .irp symbol, values . . . (1)

- evaluate a sequence of statements assigning different values to symbol
- statement sequence is defined by
    - .irp directive
    - .endr directive
- this statement sequence is iterated over "values"
- during each iteration, `is replaced with one of iterating "values"

7.65 .irp symbol, values . . . (2)

For example, assembling

```
.irp      param,1,2,3
move      d\param,sp@-
.endr
```

is equivalent to assembling

```
move      d1,sp@-
move      d2,sp@-
move      d3,sp@-
```

7.66 .irpc symbol ,value (1)

- similar to .irp symbol, values
- instead values, one value of a character string
- in the statement sequence, `is replaced with a character in a string

7.66 .irpc symbol ,value (2)

```
For example, assembling
            .irpc      param,123
            move      d\param,sp@-
            .endr
```

is equivalent to assembling

```
            move      d1,sp@-
            move      d2,sp@-
            move      d3,sp@-
```

7.60 .if absolute expression (1)

- .if marks the beginning of a section of code which is only

considered part of the source program being assembled

- if the argument (which must be an absolute expression) is non-zero
- the end of the conditional section of code must be marked by .endif

7.60 .if absolute expression (2)

.ifdef symbol

- if the specified symbol has been defined
- a symbol which has been referenced but not yet defined is considered to be undefined.

.ifndef symbol .ifnotdef symbol

- if the specified symbol has not been defined
- both spelling variants are equivalent
- a symbol which has been referenced but not yet defined is considered to be undefined

7.60 .if absolute expression (3)

.ifb text

- if the operand is blank (empty)

.ifnb text

- Like .ifb, but the sense of the test is reversed
- if the operand is non-blank (non-empty).

7.60 .if absolute expression (4)

.ifc string1 ,string2

- if the two strings are the same
- optionally quoted with single quotes
- when not quoted
  - the first string stops at the first comma
  - the second string stops at the end of the line
- strings which contain whitespace should be quoted
- case sensitive

.ifnc string1 ,string2 .

- Like .ifc, but the sense of the test is reversed
- if the two strings are not the same.

7.60 .if absolute expression (5)

.ifeq absolute expression

- if the argument is zero

.ifne absolute expression

- if the argument is not equal to zero
- equivalent to .if

.ifeqs string1, string2

- another form of .ifc
- the strings must be quoted using double quotes

.ifnes string1 ,string2

- Like .ifeqs, but the sense of the test is reversed
- if the two strings are not the same.

7.60 .if absolute expression (6)

.ifge absolute expression

- if the argument is greater than or equal to zero

.ifgt absolute expression

- if the argument is greater than zero

.ifle absolute expression

- if the argument is less than or equal to zero.

.iflt absolute expression

- if the argument is less than zero.

7.37 .else

- part of the as support for conditional assembly
- marks the beginning of a section of code to be assembled if the condition for the preceding .if was false.

7.38 .elseif

- part of the as support for conditional assembly
- shorthand for beginning a new .if block that would otherwise fill the entire .else section

7.42 .endif

- part of the as support for conditional assembly
- marks the end of a block of code that is only assembled conditionally

7.32 .def name

- begin defining debugging information for a symbol name
- the definition extends until the .endef directive is encountered.

7.40 .endef

- flags the end of a symbol definition begun with .def.

7.54 .func name [,label]

.func emits debugging information to denote function name, and is ignored unless the file is assembled with debugging enabled. Only '–gstabs[+]' is currently supported. label is the entry point of the function and if omitted name prepended with the 'leading char' is used. 'leading char' is usually _ or nothing, depending on the target. All functions are currently defined to have void return type. The function must be terminated with .endfunc.

7.41 .endfunc

- marks the end of a function specified with .func.

7.39 .end

- marks the end of the assembly file
- does not process anything in the file past the .end directive

TTTT