NAME

cul − consult file format for computer intelligence disassembler 386

DESCRIPTION

cul files are consulted by **cidis**. They contain a prescription of how the file that has been fetched (henceto-forth called *binary image*) is to be disassembled.

On top of the user commands described in this page, the full assembler is available after the command **ASSEMBLER** and the full Forth language is available after the command **FORTH** . The Forth library is available after suitable installation.

SYNTAX

These command may reside in the cul file, but they can be used interactively too.

COMMENT

A backslash \ followed by a space ignores the remainder of a line. An open bracket **(** followed by a space ignores the remainder of the file until a closing bracket (so don't use in expressions!). Comment signs are not recognized within strings. The string symbol **"** is not recognized within comment.

EXPRESSIONS

An atom can be a number, a character, a label or a flag. The default number base is hexadecimal, but this can be overruled by the **#** prefix for decimal.

A character is denoted by a **&** prefix.

A label must be previously defined by an **EQU**, **LABELED** or **LABEL** statement.

Hexadecimal numbers can start in a letter, and labels can be fully numeric, but both practices are discouraged.

A flag contains either all 0's or all 1' in number base 2.

A string is delimited by double quotes: **"** . It may contain embedded new lines. It can contain double quotes, by doubling them. The representation is an address and the number of characters. Those two atoms can be manipulated individually, but some commands operate on strings.

Atoms are untyped.

A simple expression consists of one or two atoms followed by a unary or binary operator, in a *postfix* fashion. A compound expression results from replacing one or both of the atoms by an expression.

Brackets are not allowed. They are not needed, because an unlimited number of atoms can be retained, e.g. "*12<((4+5)*(8+9))*" becomes " 12 4 5 + 8 9 + * <"*.

All operators, despite suggestive naming, are allowed on all data. Inappropriate use of operators may yield unwanted results.

An arithmetic expression combines atom's or other expressions by the binary operators **+ - / * MOD** and the unary operator **NEGATE**.

Arithmetic expressions can be combined using the binary operators **= <> < > <= >=** yielding a flag.

A logical expression combines atom's by the bitwise binary operators **OR AND XOR** and the unary operator **INVERT** , again in postfix fashion.

Equality of strings is established by **$=** working on two strings (4 atoms) and yielding a flag.

A range consist of two atoms, representing addressing in the host space, i.e. such as seen during execution. Ranges are exclusive, but they are always expanded to contain at least one byte.

A name cannot contains spaces.

SECTIONS

The word **section** is used in the sense traditional for assembly programming: a range of memory that is addressed in a contiguous way. Normally the data of a section is specified in the binary file that is being analysed. If so, it must occupy a contiguous range there too, starting at the **file-offset** in the file. Sections may overlap, in particular if ranges of memory are moved around. In those cases more than one memory address is associated with the same data, at different times during execution. A part of a section that is contiguous and of the same type (e.g. all code, or all string) is called **range**. So a range can be disassembled by a single command.

KEYWORDS

Keywords (or words in Forth parlance) apply to one or more preceeding expressions. They may also scan ahead, mostly for a name that is hence-to-forth known, e.g. as a label.

FETCH <*name*>
Fetch the binary image <*name*> to the code buffer.

CONSULT <*name*>
Get the information from file <*name*>. It presumedly refers to the current codebuffer and is used to analyse its content.

<*expr*> **LABEL** <*name*>
Add a label to the plain labels with name <*name*> with value <*expr*>.

<*expr*> <*string*> **LABELED**
Add a label to the plain labels with value <*expr*> and the name in <*string*>. This is useful if the string is extracted from the binary image.

<*expr*> **EQU** <*name*>
Add <*name*> to the plain labels with value <*expr*>.

<*expr*> **.LABEL/.**
If *expr* is known as a plain label print the label, otherwise print its value in hex.

<*expr1*> <*expr2*> **-dn:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as uninitialised space.

<*expr1*> <*expr2*> **-db:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as byte values.

<*expr1*> <*expr2*> **-dw:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as word (16-bit) values.

<*expr1*> <*expr2*> **-dl:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as long (32-bit) values. They are printed as symbolic names, if they occur as a plain label.

<*expr1*> <*expr2*> **-d$:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as strings. This means it will be shown (in order of preference):

in the form *"<letters>"* for consequitive ASCII characters

in the form *&<char>* for isolated ASCII characters

in the form ^<*char*> for some expected control characters

in the form *[<hex>]<hex>* for other numbers.
All this is acceptable by the **d$** directive of the assembler.

<*expr1*> <*expr2*> **-dc:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is to be disassembled as a normal code range. For the Intel 80386 this means a 32-bit code range.

<*expr1*> <*expr2*> <*string*> **[-dn|-db|-bw|-dl|-dc|-d$]**
These commands are equivalent to **-dn: -db: -dw: -dl: -dc: -d$:** but the ranges get their names from <*string*>.

<*expr1*> <*expr2*> **[-dn-|-db-|-bw-|-dl-|-dc-|-d$-]**
These commands are equivalent to **-dn: -db: -dw: -dl: -dc: -d$:** but the ranges are anonymous.

<*expr*> **ORG**
Clear the code buffer and associate its start with the target address <*expr*>.

<*expr*> **-ORG-**
The start of the code buffer is associated with the target address <*expr*> without affecting its content. (**Note:** this is messy and will probably be replaced by assembly sections and one **-ORG-** per section.)

<*string*> <*expr*> **DIRECTIVE**
The string is a directive associated with the target address <*expr*>. It will be a separate line or lines in front of the disassembly.
Note: This can be a multiple-line-comment, provided it is laid out as comment.

<*expr*> **COMMENT:** "comment"
The remainder of the line is a comment associated with the target address <*expr*>. It will be printed after the disassembly on the same line.

DISASSEMBLE-ALL
Disassemble the code buffer using all available information.

SORT-ALL
Sort all available information on the addresses it applies to. This is mandatory for **DISASSEMBLE-ALL** and recommended for **MAKE-CUL**.

MAKE-CUL
Output all available information to standard output. This includes all information added interactively.

INCLUDE <*name*>
Read in the file named <*name*> and execute all commands there in.

[PLAIN-LABELS|RANGE-LABELS] REMOVE: <*name*>
Select the plain labels or the ranges labels class and remove the label <*name*> from it. Removal of several labels in the same class need not repeat the selection.
When redefining a label is intended, the old label must be removed first.

[PLAIN-LABELS|SECTION-LABELS] <*expr*> **REMOVED**
Like REMOVE: but the label is identified by its address in <*expr*>.

<*expr*> **CRAWL**
Use the information that <*expr*> is a target code address. Heuristically find as much code as possible by disassembling from this address up till an unconditional transfer of control, and assuming jumps refer to more code addresses. Add new knowledge to the labeled ranges, then combine any anonymous ranges.

<*expr*> **CRAWL!**
Add the information that <*expr*> is a target code address. It will be taken into account at the next invocation of CRAWL .

FETCHING FROM BINARY

Extracting label names from the binary is a vital capability. Also especially in headers, there are addresses to be fetched from the binary. Note that the keywords in this paragraph are operators, in the sense that they leave a result for further processing. The string operators work on addresses in the host space (unlike **L@** e.a.), so they are normally preceeded by **TARGET>HOST**.

<*addr*> **[B@|W@|L@]**
Get an 8 bit, 16 bit or 32 bit value from target address <*addr*>, in a big-endian (Intel) fashion.
Note that in 4.x lina there is a conflict with the built-in **L@** . The old **L@** is available under the name **FAR@**. (No more in 5.x lina)

<*addr*> **TARGET>HOST**
Transform the target address to a host address. It may be abbreviated to **th**.

<*addr*> **COUNT**
Get a string expression from address <*addr*>, assuming its first byte is the character count.

<*addr*> **$@**
Get a string expression from address <*addr*>, assuming its first long-word (32 bits) is the character count.

<*addr*> **Z$@**
Get a string expression from address <*addr*>, assuming it ends in an ASCII zero (c-style).

ADVANCED

A modest skill in the Forth language can increase the usefulness of **cidis** considerably.
You can get pretty far by making a customized script. The source contains many commands that are occasionally useful. All commands in the source are documented using the Stallman convention.
With the Forth commands **DUP SWAP OVER 2DUP 2SWAP 2OVER** writing down the same expression repeatedly can be avoided. See lina(1) if installed.
A sequence of commands can be combined into a macro in the following fashion (regular Forth practice):

: <*name*> <*sequence*> **;**
Using <*name*> will result in the execution of the commands in <*sequence*>. If <*sequence*> contains commands that scan ahead (e.g. *-db:*) the scanning will be done when <*name*> is invoked; this can be confusing for novices.

LABEL-STRUCT
This command can be used to add a new class of labels. All classes of labels are registered automatically. See the source **labeldis.frt**.

SHOW-REGISTER
List the names of all registered classes of labels. A class can be made current by typing its name and then its content can be printed using **.LABELS**.

<*expr*> <*string*> **?ABORT**
If <*expr*> is not zero, output the string on the error channel and exit **cidis** with an error code of 2.

INTEL 386 SPECIFIC

<*expr1*> <*expr2*> **-dc16:** <*name*>
The range <*expr1*> <*expr2*> is hence-to-forth known by <*name*> and is a range to be disassembled as a 16-bit code range. This command is specific to the Intel 80386. As are the corresponding **[-dc16|-dc16-]** commands.

<*expr*> **CRAWL16**
This command is like **CRAWL16** but applies to 16 bits code sections and generates **-dc16** family directives. **CRAWL!** is recognized for start addresses.

COMMAND

After the command **ASSEMBLER** , all assembler commands can be tried out interactively (see lina(1)).

After the command **FORTH** you have a full Forth environment available (see lina(1))

A **BYE** command ends an interactive session.

AVAILABILITY

cias / cdis is based on **ciforth**.
The underlying Forth system can be fetched from

http://home.hccnet.nl/a.w.m.van.der.horst/ciforth.html

The binary distribution of **cias / cdis** is for Intel-Linux, so not for the MS-DOS, "windows" , stand alone and Alpha Linux versions of **ciforth**.

EXAMPLE

A typical consult file to disassemble a c-program could contain:

```
100 148 -   -ORG-
0 148 -db: header
148 COMMENT: entry point
148 2008 -db: text
"Data area" 2008 COMMENT
2008 4804 -dc: data
DISASSEMBLE-ALL
BYE
```

The actual command to disassemble is:

```
cidis freecell.exe freecell.cul > freecell.asm
```

A reusable file to be included if disassembling MS-DOS **.exe** files could contain:

```
...
0
 DUP LABEL exSignature       2 +
exSignature 2 "MZ" $=
 0 = "Fatal, not an exe header!" ?ABORT
DUP LABEL exExtrabytes       2 +
DUP LABEL exPagesture        2 +
...
```

The **DUP** leaves a duplicate of the labels value and **2 +** turns it into the next label, a technique similar to that used in assembler files:

```
exSignature     EQU 0
exExtrabytes    EQU exSignature + 2
```

SEE ALSO

cias(1) computer_intelligence_assembler_386
cidis(1) computer_intelligence_disassembler_386
lina(1) Linux Native version of ciforth.

CAVEAT

Mistakes in Forth mode can easily crash **cias / cidis**.

cias / cdis is case sensitive.

AUTHOR