BeebDis A Disassembler for Acorn 6502 based machines.
2018-04 Phill Harvey-Smith.

BeebDis is a disassembler that may be used to disassemble 6502 based code, it is primarily intended to complement BeebAsm by Rich Talbot-Watkins.

Why write BeebDis ?

There are several 6502 disassemblers available, but these all seem to be either old DOS based programs that will not work on modern 64 bit operating systems. Alternately they tend to be targeted at disassembling code for other 6502 based platforms such as the Commodore, Apple or Atari based machines.

BeebDis makes no assumptions about the platform apart from that it is a 6502 based machine.

BeebDis takes the philosophy that disassembly is a partly interactive process where, and that you will need to run it several times against a piece of code modifying the parameters each time as you discover the various areas of the code you are processing.

As such BeebDis relies on the creation of a disassembly control file which configures how disassembly proceeds. This control file will contain directives that define the various areas of the disassembly process.

The most basic control file will contain a load directive and the address and name of the program to be processed. More complex files will contain directives for defining program entry points, data definitions etc.

The other file that BeebDis can use is the labels file, this contains a list of label address pairs that define pre-defined labels to be used during disassembly, this allows definition of know code entry points such as OSBYTE, OSWORD etc so these can be inserted into the code at relevant points.

Invocation.

BeebDis is a command line program and should be invoked from a command prompt with the name of the control file as it's only parameter.

Control file syntax.

NOTE all numbers are assumed to be decimal, if you want to specify a hexadecimal number you should prefix it with a dollar sign.

Keywords should be separated from their parameters by spaces, optional parameters are surrounded by <>

Keyword description.

load address filename
: Loads the file to be disassembled at the specified address.

symbols filename
: Specify a file containing symbols to be added to the symbol list, this may be specified multiple times, so that you could keep all the common symbols in one file, and machine specific in another.

save filename
: Save the output to the specified filename, if not specified output goes to stdout. This is also used to generate a save line in the output assembly listing, this will have the same filename but with the '.bin' extension.

byte address <length>
: Defines an area of data bytes, this will be output as an EQUIB in the output file. If length is not specified one byte is assumed.

word address <length>
: Defines an area of data words, this will be output as an EQUIW in the output file. If length is not specified one word (two bytes) is assumed.

dword address <length>
: Defines an area of data double words, this will be output as an EQUID in the output file. If length is not specified one double word (four bytes) is assumed.

string address <length>
: Defines a string area this will be output as EQUIS in the output file. If length is specified, then length bytes will be output. If length is not specified then BeebDis will attempt to determine the length of the string by probing ahead from the start address until it finds a character not in the range $20-$7F.

stringz address

Defines a string area this will be output as EQUIS in the output file. BeebDis will attempt to determine the length of the string by probing ahead from the start address until it finds a zero byte.

stringterm address <terminator>

Defines a string area this will be output as EQUIS in the output file. BeebDis will attempt to determine the length of the string by probing ahead from the start address until it finds a byte equal to the value of the terminator, which defaults to zero.

stringhi addr

Defines a string area this will be output as EQUIS in the output file. BeebDis will attempt to determine the length of the string by probing ahead from the start address until it finds a byte with the high bit set.

entry address

Defines a code entry point, this will be loaded into the list of entry points from which code disassembly will proceed.

wordentry addr count

Defines a block of words that will be read and added to the list of entry points. This is useful for vector tables e.g. the reset, BRK/IRQ and NMI vectors can be added with wordentry $FFFA 3

hexdump filename

Output a hexdump of the loaded area in the output file, after the disassembled code. Ths can aid in working out which areas are which in the file being processed. The filename is optional, if not supplied the dump will be added to the end of the output file.

stringscan

Scans the loaded area for groups of ASCII characters longer than 2 and outputs them at the end of the output file. This can be used to initially find out the addresses of strings defined within a block of code, which can then be added explicitly using **string** and the related keywords above.

newsym filename

Output a list of just the generated symbols to the specified file.

newpc addr

Sets new value of program counter for subsequent data who’s address is specified as ‘pc’

repeat count

Repeat the following lines (up until endrepeat) count times, this is usefull for defining data structures. **Note** repeat cannot currently be nested.

endrepeat

Terminate a previous repeat block.

A note about addresses.

The directives that can take addresses, can now have the address specified as '**pc**' which in this case means the address following the previously defined item.

So if for example you had some code like this :

```
org     $1000
JSR     $2000
EQUIB   'This is a test',0
LDA     #$ff
```

Where the routine at $2000 consumes the string and then returns to the address following it, you could use the following defines to disassemble it.

```
entry $1000
stringz $1003
entry pc
```

This would add an entry point where the LDA is in the above example.

Repeat / endrepeat can be used to easily disassemble data tables, suppose you had a table of strings of 3 bytes followed by a byte offset, these can be easily disassembled with :

```
repeat 10
string pc 3
byte pc
endrepeat
```

Generally **load** should be the first keyword in a control file followed by **save**, then any **symbols** specifications and then the rest in any order.