BeebDis A Disassembler for 6502 / 6809 based machines.
2019-05 Phill Harvey-Smith.

Now supports Motorola 6809 too!

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 its only parameter.

Control file syntax.

The control file should be a text file that consists of lines starting with the keywords as listed below (in bold). Any blank line, or a line not starting with a valid keyword is ignored. This allows you to put comments in your control file e.g. by starting a line with a semicolon or hash or similar character.

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 descriptions

load address filename

Loads the file to be disassembled at the specified address.

e.g. : load $8000 basic.rom

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.

The symbols file should be of the format :
symbol = address **OR** symbol <EQU_TOKEN> address
EQU_TOKEN can be specified using the **Equate** option (see settable options section below).

e.g. : symbols mos.asm

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.

e.g. : save results.asm

cpu cpuname

Set the CPU variant you will be disassembling for :

cpuname	Variant
6502	Plain old MOS 6502.
65c02	CMOS 65c02, with extra instructions.
wd65c02	WDC variant, supports more instructions.
6512	6512 used in BBC B+
6809	Motorola 6809 / 6809E compatible. (inc 6309 in 6809 mode).

e.g. CPU 6502

byte address <count>

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.

e.g. : byte $C000 2 2 bytes starting at $c000

word address <count>

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.

e.g. : word $C000 2 2 words starting at $c000

dword address <count>

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.

e.g. : dword $C000 2 2 dwords starting at $c000

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.

e.g. : string $a000 $10

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.

e.g. : stringterm $a000 $FF

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.

e.g. : stringz $a000

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.

e.g. : stringhi $a000

stringhiz

A combination of stringz and stringhi, the string is terminated if a zero byte **OR** a byte with the high bit set is found.

e.g. : stringhiz $a000

entry address

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

e.g. entry $c002

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

e.g. : wordentry $FFFA 3

hexdump filename

Output a hexdump of the loaded area in the output file, after the disassembled code. This 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.

e.g. hexdump output.hex

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.

e.g. : stringscan

Inlinescan addr

Scans the loaded area for a subroutine jump to the specified address. If found outputs a stringhiz + entry pc for the address following the jump. This is used to search for inline print sequences such as are common in the Acorn System & Atom programs. These will generally be of the form :

```
JSR   $Fxxx
EQU   “string to print”
NOP
(more code)

JSR   $Fxxx
EQU   “string to print”
BRK
(traps back to OS).
```

e.g. : inlinescan $a000

newsym filename

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

e.g. : newsym symbols.asm

newpc addr

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

e.g. newpc $a000

repeat count

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

endrepeat

Terminate a previous repeat block.

e.g. : repeat 10
byte pc
word pc
endrepeat

would disassemble a table consisting of 10 byte, word pairs.

option name value

Set the value for a disassembler option, see section on option values below.

e.g. option equate equ would set the value of the ‘equate’ option to equ.

verbose level

Set the verbosity level :

0 = quiet, no output unless a fatal error is encountered
1 = normal, print signon message and file load / save messages.
2 = verbose, also print additional information as working.
3 = debug, also print internal debugging information.

e.g. verbose 1

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.

Settable Options and their values.

BeebDis 1.20 and above have a set of configuration options that are settable at runtime (in the control file) using the option keyword. These options control the behaviour and output of the disassembler making it more flexible and not tied to having its output code in one format.

Some of the options have defaults, which may be overridden others will only be present if defined in the control file (and so will default to not set or empty string).

The option names are compared in a case insensitive way so it doesn't matter what case they are in in the control file.

Some of these are defaulted by the Disassembler modules these are :

LabelPrefix <string>
This defines the prefix that will be prepended to all generated labels, this will default to '.' for the 6502 disassembler and nothing for the 6809.

LabelSuffix <string>
This defines the suffix that will be appended to generated labels, and defaults to nothing. Some assemblers that the output code may be fed to may require this to be defined as ':'.

DefineByte <string>
How to define byte storage in the output code, defaults to 'equb' for 6502 and 'fcb' for 6809.

DefineWord <string>
How to define word storage (16 bits) in the output code, defaults to 'equw' for 6502 and 'fdb' for 6809.

DefineDWord <string>
How to define double word storage (32 bits) in the output code, defaults to 'equd' for 6502 and 'fqb' for 6809.

DefineString <string>
How to define string storage in the output code, defaults to 'equs' for 6502 and 'fcc' for 6809.

DefineStringh <string>
How to define string storage, for strings that are delimited by having the top bit of the last character set. Defaults to 'equs' for 6502 and 'fcs' for 6809.

Origin <string>
How to define strings in the output code, defaults to 'org' for 6502 and 'org' for 6809.

Equate <string>
How to define constants in the output code, defaults to '=' for 6502 and 'equ' for 6809.

BeginIgnore <string>
Defines the beginning of a block that will be ignored in the output code, used to wrap found strings etc. Defaults to 'if (0)' for 6502 and 'ifeq 1' for 6809.

EndIgnore <string>
Defines the end of a block that will be ignored in the output code, used to wrap found strings etc. Defaults to 'endif' for 6502 and 'endc' for 6809.

SaveCommand <string>
The save command that will be inserted into the output file, currently only beebasm needs this. Defaults to 'SAVE' for 6502 and '; ' for 6809.

CommentChar <string>
The character used to comment a line in the output file defaults to ';' for 6502 and ';' for the 6809.

CommentCol <numeric>
The minimum column number that comments should start at in the output code defaults to 40.

The following options are only set if required :

ExecFormat <string>
Set the executable format of the input file, currently understands :

'os9' : OS9 module format.
'dragondos' : Dragon DOS or DragonMMC exec format.