Home

An Introduction to Z88DK

Z88DK is a complete development toolkit for the 8080, 8085, gbz80, z80, z180, z80n and Rabbit processors.

It contains two C compilers, an assembler / linker / librarian, data compression tools and a utility for processing the raw binaries into forms needed by specific targets.

It comes with an extensive library of functions written in assembly language that implements the C standard and many extensions. It holds the largest repository of z80 code on the internet.

Development in assembly language or C is completely integrated; projects can be 100% assembler, 100% C or any mixture of the two. The toolset treats both as first-class languages and is designed to make it very easy to mix them at will with C and asm functions being able to call each other or make use of the hand-optimized library functions.

The toolset is modern and has modern features. Most z80 assembly tools are simple in that they assume a small memory space (confined to 64k) or lack linking capability. Z88DK is able to generate ROMable code and code for multiple memory banks (bank assignment still requires human direction) and the linking capability means large bodies of code or data can be shared in libraries between projects with the linker only drawing out code and data that is actually used by a linking program.

We compare Z88DK with other commercial and non-commercial offerings using benchmarks to identify where Z88DK can be improved. Z88DK compares favourably and you can see that for yourself by investigating the benchmarks if you are interested.

Libraries

z88dk supports two different library implementations, they are being slowly merged and much of the standard C library is now shared between the implementations.

Classic library

The classic library is z88dk's traditional library and is written with in a mix of assembler and C.

Supported Platforms

The classic library supports over 100 platforms and provides many extensions (graphics, sound etc) as well as broadly supporting the C library standard.

Newlib

The new C library aims to implement as large a subset of C11 as is reasonable on an 8-bit target. The library does not confine itself to the standard and adds many non-standard functions drawn from BSD and GNU, as well as libraries aiming to support text, graphics and sound among other things. The library is slowly being merged into the classic library.

Supported Platforms

The new library supports a restricted number of targets, and notably doesn't (yet) support file I/O. The supported targets are: +cpm, +hbios, +rc2014, +scz180, +sms, +yaz180, +z80, +z180, +zx, +zxn

Tools

This is a quick overview of the tools included in Z88DK. Names without underlining do not have a Help-Text (as of 2023-02-16)

- **ZCC** is the toolchain's front end. zcc can generate an output binary out of any set of input source files.
- **SCCZ80** is z88dk's native c compiler. sccz80 is derived from small c but has seen much development to the point that it is nearly c89 compliant as well features from later standards.
- **ZSDCC** is z88dk's customization of the sdcc optimizing c compiler. Our patch makes sdcc compatible with the z88dk toolchain, gives it access to z88dk's extensive assembly language libraries and ready-made crts, addresses some of sdcc's code generation bugs and improves on sdcc's generated code. It has very good standards compliance with c89, some c99 and a little c11.
- **Z88DK-Z80ASM** (not to be confused with several external projects called z80asm) is a fully featured assembler / linker / librarian implementing sections.
- **Z88DK-Z80NM** is z80asm's companion library dumper. It can provide a listing of functions or data encoded in an object or library file.
- **Z88DK-ZOBJCOPY** allows object and library files built by **Z80ASM** to be manipulated.
- **Z88DK-APPMAKE** processes the raw binaries generated by the toolkit into a form suitable for specific target machines. For example, it can generate intel hex files, tapes, ROMs, etc.

- **Z88DK-TICKS** is a command line z80 emulator that can be used to time execution speed of code fragments.
- **Z88DK-GDB** provides the debugger interface from ticks and connects to a gdbserver to permit line-by-line debugging of software in emulators or on real hardware.
- **Z88DK-DIS** is a command line disassembler for Z80, Z180, Z80N and Rabbit 2000/3000. It can additionally read map files generated by z80asm to provide a more symbolic output.
- **Z88DK-LIB** is an installer for third party libraries. It manages installation, removal and listing of available libraries.
- **Z88DK-ZX0** and **Z88DK-ZX7** are PC side data compression tools with companion decompression functions in the z80 library.
- **Z88DK-DZX0** and **Z88DK-DZX7** are PC-side decompressor counterparts to the z88dk-zx0 and z88dk-zx7.

These tools are not normally directly invoked by the user:

- **M4** acts as z88dk's macro preprocessor and can optionally process files ahead of the c preprocessor or assembler.
- **UCPP** is the c preprocessor invoked for sccz80.
- **ZSDCPP** is the c preprocessor invoked for zsdcc.
- **Z88DK-ZPRAGMA** is used by the toolchain to process pragmas embedded in c source.
- **Z88DK-COPT** is a regular expression engine that is used as peephole optimizer for sccz80 and as a post-processing tool for both sccz80 and zsdcc.

= = = = = == =

Tool zcc

The compiler frontend: ZCC

The frontend of z88dk is called zcc, it is this that you should call if you want to do any compilations. To invoke the frontend use the command:

```
zcc [flags]  [files to be compiled/linked]
```

The files can be either C files (.c) , preprocessed C files(.i), compiled C files (.asm), optimised compiled file (.opt) or assembled files (.o), any combination of them can be mixed together and the relevant processed done on them.

Processing of a file list is done on each file in turn (i.e. preprocess, compile, optimise, assemble) at the end all files may be linked into a single executable if desired.

Options accepted by zcc can be reported using:

```
zcc -h
```

Details on the configuration of a port can be revealed with:

```
zcc +port -specs
```

Options to control the action of the frontend

```
     +[file]       Name of alternate config file
                   (must be the first argument)
     -a            Produce .asm (or .opt) file only
     -S            Produce .asm (or .opt) file only
     -c            Do not link object files
     -E            Preprocess files only, leave output in .i file
     -o            Specify the output name for the stage that compiling will
stop
                   at. For example "-a -o file.asm" will output an assembly
file.
     -bn [file]    Specify output file for binary (default is a.bas for BASIC
                   programs and a.bin for application binaries)
     -On           Optimize compiler output (to .opt file)
                   n can be either 0 (none) 1,2,3, level 2 is recommended.
                   Level 3 is suitable for large programs (includes certain
                   lib functions to reduce size of code(!))
     -v            Verbose - echo commands as they are executed
     -vn           Don't be verbose
     -clib=[lib]   Switch to the specified standard library
     -subtype=[x]  Generate output for the platform subtype x
     -compiler=X   X=sdcc or X=sccz80
```

Options to control library usage

```
     Parameters valid for the Z88 (see the platform sections for more options)

     -Lpath        Add to the library search path
     -l[name]      Link in a library - supply just the name (after placing them
                   in the correct directory)
     -lm           Link in the generic Z80 maths library

     -m            Generate .map files when assembling/linking
     -s            Generate .sym files when assembling/linking
     --list        Generate list files
```

Other libraries are available

Options to control the type code produced

```
     -unsigned        Implicitly define everything as unsigned unless explicitly
                      told otherwise (sccz80)
     -create-app      Create a file suitable for running on an emulator/machine
     -startup=        This parameter affects the resulting code in a target
                      dependent way: in example, when used with the Cambridge
Z88
                      the -startup=3 parameter instructs the compiler to produce
                      standalone code that can be run from a set
                      address from BASIC. (Use -zorg= to change the address)

     -pragma-define   Define an option,for example:
     -pragma-bytes    Dump a string of bytes zcc_opt.def
     -pragma-redirect Redirect a function, for example:
                      -pragma-redirect:fputc_cons=xyz123
```

```
                      will make xyz123 the assembler label that is used for
console
                      output
```

Miscellaneous options

```
     --c-code-in-asm Intersperse C code as comments in the assembler output,
warning:
                    this *will* clobber some optimisations for sccz80
     -Cp[option]   Pass an option through to the pre-processor
     -Ca[option]   Pass an option through to the assembler
     -Cl[option]   Pass an option through to the linker
     -Cz[option]   Pass an option through to appmake
     -Cc[option]   Pass an option through to sccz80
     -Cs[option]   Pass an option through to sdcc
```

In addition, the flags, -D, -I, -U are passed through to the preprocessor.

Any unrecognised options are passed through to the compiler (to allow for improvements in the future.)

Configuration files

In order for z88dk to work on as many platforms as possible and so that it can be easily tweaked, retargetted and generally mutilated, the frontend (zcc) consults a plain text configuration file which is in the directory pointed to be the ZCCCFG variable.

The config file is indicated by the first option to zcc which is `+[machine identifier]`

If you wish to use a config file located in the current directory or anywhere else on the system then specify the full path and filename - make sure the filename as the suffix .cfg.

The order of checking is as follows:

1. "Local" file (if exists) eg +temp.cfg
2. ZCCCFG/[name].cfg eg +z88

Tool z80asm

Z80 macro assembler / linker / librarian (z80asm)

z80asm is part of the z88dk project and is used as the back-end of the z88dk C compilers. It is not to be confused with other non-**z88dk** related projects with the same name.

z80asm is a relocatable macro assembler, linker and librarian that can assemble Intel 8080/8085 and Z80-family assembly files into a relocatable object format, can manage sets of object files in libraries and can build binary images by linking these object files together. The binary images can be defined in different sections, to match the target architecture.

Usage ...

... as assembler

```
z88dk-z80asm [options] file...
```

By default, i.e. without any options, **z80asm** assembles each of the listed files into relocatable object files with a `.o` extension. It shows a summary of all the options when called with the `-h` option.

... as linker

```
z88dk-z80asm -b [options] file...
```

When called with the `-b` option, **z80asm** links the object files together into a set of binary files.

... as librarian

```
z88dk-z80asm -xlibrary.lib [options] file...
```

When called with the `-x` option, **z80asm** builds a library containing all the object files passed as argument. That library can then be used during linking by specifying it with the `-l` option.

Contents

Copyright

The original z80asm module assembler was written by Gunther Strube. It was converted from QL SuperBASIC version 0.956, initially ported to Lattice C, and then to C68 on QDOS.

It has been maintained since 2011 by Paulo Custodio.

License

Tool z80nm

Object and Library File Dumper

Usage

```
z80nm [options] input
```

Description

z80nm reads the input library or object file generated by *z80asm* and produces a dump of its contents. It can read all the current and past object and library file versions.

Options

- -a
  Show all
- -l
  Show local symbols
- -e
  Show expression patches
- -c
  Show code dump
- -h
  Display this help

Tool zobjcopy

Object and Library File Manipulator

Usage

```
zobjopy input [options] [output]
```

Description

zobjcopy reads the input library or object file generated by *z80asm* and produces an equivalent output file, where the input file can be in any of the current or older object file formats of *z80asm* and the output file is updated to the current format.

Each option denotes one action to be done on the input file, and multiple options can be specified to execute multiple actions in the sequence given in the command line.

Options

- -l | --list
  Only reads an input file and displays the contents, does not write an output file.
- --hide-local
  In option --list, do not show local symbols.
- --hide-expr
  In option --list, do not show expressions.
- --hide-code
  In option --list, do not show code dump of sections.
- -v | --verbose
  Tells what is happening.
- -s old-name-regex=new-name | --section old-name-regex=new-name
  Renames all sections that match the old name regex (standard POSIX) to the new name. Code sections are merged, if appropriate, fixing the patch addresses of expressions and label values accordingly.
- -p name-regex,prefix | --add-prefix name-regex,prefix
  Renames all global symbols that match the given regex, adding the specified prefix. All expressions where the renamed symbols are used are corrected accordingly.
- -y old-name=new-name | --symbol old-name=new-name
  Renames a global or external symbol. All expressions where the renamed symbols is used are corrected accordingly.
- -L regex | --local regex
  Makes all global symbols that match the regex local.
- -G regex | --global regex
  Makes all local symbols that match the regex global.
- -F nn|0xhh |--filler nn|0xhh
  Change the filler byte to be used when merging sections to respect the ALIGN requirement. The default is 0xFF.
  The value can be supplied as decimal or hexadecimal with a '0x' prefix.
- -O section,nn|0xhh |--org section,nn|0xhh
  Change the ORG of the given section.
- -A section,nn|0xhh |--align section,nn|0xhh
  Change the ALIGN of the given section.

Examples

Dump the contents of a library file:

```
zobjcopy --list file.lib
```

Rename all sections starting with "text" to "rom" and all starting with "data" to "ram", writing the output in file2.lib:

```
zobjcopy file.lib --section ^text=rom --section ^data=ram file2.lib --verbose
```

Add a "lib_" prefix to all global symbols that start with "ff":

```
zobjcopy file.lib --add-prefix ^ff,lib_ file2.lib --verbose
```

Rename the global symbol "main" to "_main"

```
zobjcopy file.lib --symbol main=_main --verbose
```

Make the "main" symbol local:

```
zobjcopy file.lib --local '^main$' --verbose
```

Make the "main" symbol global:

```
zobjcopy file.lib --global '^main$' --verbose
```

Merge all sections, use 0x00 as the alignment byte:

```
zobjcopy file.lib --filler 0x00 --section .=text --verbose
```

Change section "text" to ORG 0x8000 and ALIGN 16:

```
zobjcopy file.lib --org text,0x8000 --align text,16 --verbose
```

TO-DO

- factor code into z80asm
- disassemble code sections

Tool ticks

Ticks is a command line CPU emulator that can be used for testing and debugging algorithms and code. The classic library can compile binaries that run on ticks using the +test target.

Usage

In it's simplest case, launch ticks with:

```
z88dk-ticks [binary file]
```

It will run for around 10,000,000 ticks which may not be sufficient, to increase the time it will run use the `-w` option:

```
z88dk-ticks -w X [binary file]
```

Where X counts the number of 400,000,000 cycles to wait before exiting.

To specify command line options (which are picked up main in the `argc` and `argv` parameters invoke ticks as follows:

```
z88dk-ticks [binary file] -- [argv0] [argv1]
```

CPU Features

Ticks supports emulating the z80 (default), 8080 (`-m8080`), 8085 (`-m8085`), gbz80 (`-gbz80`), z180 (`-mz180`), ZX Next z80 extensions (`-mz80-zxn`) and Rabbit processors (`-mr2k`), reporting accurate timing information for each target. *Note:* Not all Rabbit instructions are emulated.

When emulating the ZX Next cpu, ZX Next style MMU paging is available.

CP/M Emulation

Ticks supports a limit number of BDOS calls launching it to run a .COM file will enable this mode and can allow some CP/M programs to run.

Debugging

Ticks provides a command line debugger, this can be launched as follows:

```
z88dk-ticks -d -x [map file] [binary file]
```

Specify `-x [map file]` is optional, however specifying it allows symbolic debugging. Ticks will then sit at address 0 waiting for an input from you. Type `help` to view the available commands.

Hotspot detection

Ticks can also report hotspots for code execution, launch the debugger, type `hotspot on` and then `cont`, on exiting a file called `hotspots` will be written in the current directory. This file reports the number of times an address has been executed along with a disassembly of that line. To order this in terms of frequency you can use the standard sort tool. For example:

```
sort -nr hotspots
```

Will show the commonest hit addresses first of all.

```
 sort -nr -k2 hotspots
```

Will show the number of clock cycles spent at each address.

Stdio and File I/O

Ticks provides a full stdio that will output to the console, alongside this, file I/O is supported as well.

Hardware emulation

- Ticks provides an ACIA emulation that is accessed using ports 0x80 (Ctrl/status) and 0x81 (in/out) that can be used to simulate the serial port on an RC2014 board.
- An emulated AM9511 maths co-processor is available on ports 0x42 and 0x43

Tool z88dk dis

A full z80 disassembler is provided with z88dk. In addition to supporting the z80, it can also disassemble code for the other supported CPUs (8080, gbz80, z180, z80-z80n and Rabbit processors) as well as other related (but unsupported by z88dk) processors: 8085, R800, ez80.

Usage

At its simplest:

```
z88dk-dis -o [address] [binary file]
```

Will load binary file at the specified address and start disassembling from that point.

The diassembler understands the .map files generated by z80asm, and these will be read using the `-x` option:

```
z88dk-dis -o [address] -x [map file] [binary file]
```

Symbols that appear in more than one compiled source code file (module) will have the module name appended to them. e.g. a symbol named `loop` in modules `fopen_c` and `fclose_c` will become `loop_fopen_c` and `loop_fclose_c`.

Specifying the CPU

The following options control which CPU z88dk-dis will interpret the binary as.

Flag	CPU	Notes
-mz80	Z80	
-mz180	Z180	
-mez80	ez80	Short mode only
-mz80-z80n	ZX-Next	Supports the extended opcodes in the ZX Next CPU
-mr2k	Rabbit 2000	
-mr3k	Rabbit 3000	
-mr800	Ascii R800	As used in the MSX Turbo
-mgbz80	Gameboy z80	
-m8080	Intel 8080	Using Z80 Opcodes