CHAPTER 7

DEBUGGING SUPPORT

This chapter describes the debugging support available with the cross compiler targeting the **Z80/HD64180**. There are two levels of debugging support available, so you can use either **cxdb**, the C source level cross debugger or your own debugger or in-circuit emulator to debug your application.

Generating Debugging Information

The compiler generates debugging information in response to command line options you pass to the **c** compiler driver as described below. The compiler can generate two kinds of debugging information:

1. line number information that allows **cxdb** or another debugger or emulator to locate the address of the code that a particular C source line (or set of lines) generates, and

2. information about the name, type, storage class and address (absolute or relative to a stack offset) of program static data objects, function arguments, and automatic data objects that functions declare.

You may put line number information into the object module in either of the two formats, or you can generate both line number information and information about program data and function arguments, as described below.

3. information about what source files produced which relocatable or executable files. This information may be localized by address (where the output file resides in memory). It may be written to a file, sorted by address or alphabetical order, or it may be output to a printer in paginated or unpaginated format.

Generating Line Number Information

The compiler puts line number information into a special debug symbol table. The debug symbol table is part of the relocatable object file produced by a compilation. It is also part of the output of the **lnk80** linker. You can therefore obtain line number information about a single file, or about all the files making up an executable program. However, the compiler can produce line number information only for files that are fewer than 65,535 lines in length.

You can direct the compiler to generate one of two different types of line number information. "Type 1" specifies a function name and a line number that is relative to the first line of the source file. "Type 2" specifies a file name and a line number relative to the first line of the source file.

To generate type 1 line number information, specify the **-ddl1** option to the compiler driver. To generate type 2 line number information, specify the **-ddl2** option to the compiler driver.

You can request either type 1 or type 2 line number information. Do not specify **-ddl1** and **-ddl2** together.

You use the **-ddl1** and **-ddl2** command line options to direct the compiler to generate line number information into object files. You can then use the **lines** utility to extract that information, as described below.

The compiler produces line number information for multi-line instructions, as well as multi-instruction lines. For example, when compiling a program containing a **for** statement, the line number information produced references the address of each component of the **for** statement. Given that the following C source line

```
for (i = 0, j = 2; i < MAX; ++i, ++j)
```

is on line 25 of the file **test.c** and that **test.c** has been compiled with the **-ddl2** option enabled, the component

```
i = 0, j = 2
```

will be referred to as "line 25." The test

```
i < MAX;
```

will be referred to as "line 25(1)," since it is the first "subline" of line 25. Similarly, the component

++i, ++j

will be referred to as "line 25(2)" as it is the second subline of line 25. **lines** can extract subline or range information, as described below.

Generating Line Number and Data Object Information

The **-dxdebug** option directs the compiler to generate both line number information and information about data objects and function arguments and return types. The line number information is type 2, as if you had specified the **-ddl2** option. The information produced about data objects includes their name, scope, type and address. The address can be either absolute or relative to a stack offset.

The debugging information the compiler generates when you specify **-dxdebug** is the information used by the **cxdb** C source level cross debugger.

As with line number information alone, you can generate debugging information about a single file or about all the files making up an executable program.

lines and **prdbg** may be used to extract the debugging information from files compiled with the **-dxdebug** option, as described below.

Generating Source Listings for Object Files

The lines Utility

lines extracts line number information from object modules and from executable images produced by the linker. The information **lines** extracts consists of the starting address of the code produced by the original C source line, the size of the code generated in bytes, and, where applicable, a subline number or a range of line numbers for multi-line instructions.

lines accepts the following command line options, each of which is described in detail below:

lines -[l# n* r s v] <file>

where <file> is an object file or executable image compiled from C source with one of the compiler command line

options **-dxdebug**, **-ddl1** or **-ddl2** enabled.

-l# print information about an executable line or range of executable lines. # may specify a single line number or a range of line numbers of the form #:#, where the number preceding the colon ':' specifies the low end of the range.

-n* print information about the executable lines in the function or file *. If <file> has been compiled with the **-ddl1** option, * implicitly specifies a function name. If <file> has been compiled with the **-ddl2** option or with **-dxdebug**, * implicitly specifies a file name. Only those lines belonging to the specified <file> are displayed. You specify the format used to print the information using the **-r**, **-s** or **-v** options in combination with **-n***.

-r Print "range" information. When this option is enabled, both the start and end lines of instructions spanning multiple lines are displayed. This option displays line number information of the form

<address> <file> <line number> **:<non_xeq_line>**

where <address> is the address of the executable line in memory, <file> is the object file name, <line number> is the number of the line in the object file, and <non xeq line> represents the inclusion of nonexecutable lines after the executable line preceding it.

-s print "size" information of the form

<file> <address> <size>

where <file> is the name of the object file, <address> is the start address of the code generated, and <size> is the size of the code the line produced in bytes.

-v suppress the display of the start address of the source lines. This option prints line number information of the form

<file> <line number>

where <file> is the name of the object file and <line number> is the number of the line in the object file.

By default, **lines** prints the start address, object file name and the line number of each executable line in <file>.

The prdbg Utility

prdbg extracts information about functions and data objects from an object module or executable image that has been compiled with the **-dxdebug** option enabled. **prdbg** extracts and prints information on the name, type, storage class and address (absolute or offset) of program static data objects, function arguments, and automatic data objects that functions declare. For automatic data, the address provided is an offset from the frame pointer. For function arguments, the address provided is an offset from the stack pointer.

prdbg is designed to be used in combination with **lines** to extract all relevant debugging information from a file.

prdbg accepts the following command line options:

```
prdbg -[fc* fl* o*] <file>
```

where <file> is an object file compiled from C source with the compiler command line option **-dxdebug** enabled.

-fc* print debugging information only about the function *. By default, **prdbg** prints debugging information on all functions in <file>. Note that information about global data objects is always displayed when available.

-fl* print debugging information only about the file *. By default, **prdbg** prints debugging information on all C source files.

-o* print debugging information to file *. Debugging information is written to your terminal screen by default.

By default, **prdbg** prints debugging information about all functions and global data objects in <file>.

Examples

The following examples show sample output generated by running the **lines** and **prdbg** utilities on an object file created by compiling the program **acia.c** with the compiler options **-ddll** and **-dxdebug** enabled.

```
lines acia.80
    0x102c acia.c 33
    0x103c acia.c 35
    0x103c acia.c 34
    0x1049 acia.c 36
    0x1056 acia.c 37
    0x105c acia.c 38
    0x1069 acia.c 46
    0x106e acia.c 47
    0x10a6 acia.c 67
    0x10af acia.c 68
    0x10b0 acia.c 69(1)
    0x10b0 acia.c 71
    0x10b6 acia.c 72


prdbg acia.80
 Information extracted from acia.80
SOURCE FILE : acia.c
  FILE VARIABLES :
      extern unsigned char buffer[512]  at 0x8005
      extern unsigned char *ptlec  at 0x8003
      extern unsigned char *ptecr  at 0x8001
  FUNCTION : extern unsigned char getch() lines 29 to 39 at 0x1024-0x1066
    VARIABLES:
      auto unsigned char c at -1 from frame pointer
  FUNCTION : extern int outch() lines 43 to 47 at 0x1066-0x1071
    VARIABLES:
      argument int c at 0x4 from frame pointer
  FUNCTION : extern int recept() lines 53 to 58 at 0x1071-0x109e
  FUNCTION : extern int main() lines 63 to 74 at 0x109e-0x10c0
    VARIABLES:
      auto unsigned char c at -1 from frame pointer
```

The clist utility

The clist utility takes relocatable or executable files as arguments, and creates listings showing the C source files that were compiled or linked to obtain those relocatable or executable files. It is a convenient utility for finding where the source statements are implemented.

To use **clist** efficiently, its argument files must have been compiled with either the **ddl2** or **dxdebug** flags set.

clist can be instructed to limit its display to files occupying memory in a particular range of addresses, facilitating debugging by excluding extraneous data. **clist** will display the entire content of any files located between the endpoints of its specified address range.

The syntax for calling **clist** is as follows:

clist -[a d* l# o* p r*] <files>

---

clist Options

clist takes the following flags when selecting options:

-a when set, causes **clist** to list files in alphabetical order. The default is that they are listed by increasing addresses.

-d* read string * to locate the source file in a specific directory. Source files will first be searched for in the current directory, then in the specified directories in the order they were given to **clist**. Up to ten such specifications may be given.

Note that the concatenation of the directory specification and file name must constitute a valid pathname. For example, under UNIX, a directory specification should include a terminating **/**. Under MS-DOS, a directory specification should end with **** or **/**, depending which convention is in use.

l# when paginating output, make the listings # lines long. By default, listings are paginated at 66 lines per page.

o* redirect output from **clist** to file #. You can achieve a similar effect by redirecting output in the command line;

clist -o li.dat is equivalent to

clist > li.dat

-p Suppress pagination. No page breaks will be output.

-r* * is a range specification. It must be of the form <number>:<number>. When this flag is specified, only those source files occupying memory in the specified range will be listed. If part of a file occupies memory in the specified range, that file will be listed in its entirety.