Z8000 C cross compiler and assembler

for the Olivetti M20 running PCOS

Christian Groessler, chris@groessler.org

Table of Contents

1 Introduction

This manual contains documentation for a Z8000 C and assembler development toolchain targetting the Olivetti M20 personal computer running the PCOS operating system.

The M20 was a Z8001 based personal computer sold by Olivetti during the early eighties. It had 128kB to 512kB of RAM, monochrome or color (4 or 8 colors) displays, one or two 5.25" floppy drives, and optionally a hard drive. See Davide Bucci's excellent web page at http://www.z80ne.com/m20/index.php for more information about the M20.

The toolchain is based on a Z8000 port of the GNU C compiler (the one which comes with the eCos tools[1]), the GNU binutils[2], and newlib[3].

This document describes the January 19, 2009 release of the tools.

[1] ftp://sources.redhat.com/pub/ecos/releases/ecos-1.2.1/ecosSWtools-990319-src.tar.bz2
[2] http://www.gnu.org/software/binutils
[3] http://sourceware.org/newlib

2 Getting and installing the toolchain

2.1 Getting the toolchain

The main distribution site is
ftp://ftp.groessler.org/pub/chris/olivetti/m20/misc/z8kgcc.

Source code and precompiled versions for selected architectures are provided.

2.2 Installing the toolchain

Tarballs with precompiled versions for Linux/x86[1], MacOS-X 10.5 (ppc and x86)[2], FreeBSD/x86[3] or NetBSD/ppc[4] are available.

Extract the tarball into the root directory of your system. It will create a directory hierarchy under `/opt/z8kgcc-jan-19-2009`. Add `/opt/z8kgcc-jan-19-2009/bin` to your PATH environment variable, e.g. with these commands at the shell:

```
$ PATH=/opt/z8kgcc-jan-19-2009/bin:$PATH
$ export PATH
```

That's all. Now you can invoke e.g. the C compiler with `z8k-pcos-gcc`.

If you need to build from source (probably because you use a different operating system than the ones where precompiled versions are available, or you want to make changes to the source code), you'll need to build the toolchain from source. Please refer to Chapter 9 [Building from source], page 40 of this manual.

2.3 Upgrading

Older versions had installed, and newer versions will install, into a different directory than `/opt/z8kgcc-jan-19-2009`. Typically it will be a directory like `/opt/z8kgcc-<date-of-release>`. Therefore it's possible to have different versions installed on the system at the same time. This is true for the precompiled versions. If you build from source you can place the installations at any location you like.
Select the version you want to use by putting its `bin` directory in the PATH environment variable as described above.
Alternatively you can execute the version you want by invoking it with its absolute path, like

```
$ /opt/z8kgcc-jan-19-2009/bin/z8k-pcos-gcc <parameters>
```

This way you can quickly switch between versions.

---

[1] `z8kgcc-jan-19-2009-linux-fc9.tar.bz2`
[2] `z8kgcc-jan-19-2009-darwin-9-ub.tar.bz2`
[3] `z8kgcc-jan-19-2009-freebsd-7-0-x86.tar.bz2`
[4] `z8kgcc-jan-19-2009-netbsd-4-0-ppc.tar.bz2`

3 The C compiler

This chapter describes the C compiler.

3.1 Overview

The precompiled releases come with 2 compilers, `z8k-pcos-gcc` and `z8k-coff-gcc`. The former creates executable files ready to run under PCOS, while the latter creates COFF[1] files which can be run under a simulator (`z8k-coff-run`).
This simulator is a generic Z8000 CPU simulator, it doesn't know about M20 specifics.

Object files and library files (`*.o` and `*.a`) and the executables created by `z8k-coff-gcc` are in COFF format. When building a PCOS program, the PCOS linker (Chapter 6 [`ldpcos` - the PCOS linker], page 20) constructs a PCOS execuable out of the COFF input files.

3.1.1 Included Tools

Here's an overview over the tools included in the release:

`gcov` Coverage testing tool (untested).

`ldpcos` PCOS linker, see Chapter 6 [`ldpcos` - the PCOS linker], page 20.

`m20stub.sav`
: GDB debugging stub to be run on the M20

`protoize`
`unprotoize`
: Automatically add or remove function prototypes.

`z8k-coff-addr2line`
`z8k-pcos-addr2line`
: Convert addresses into file names and line numbers.

`z8k-coff-ar`
`z8k-pcos-ar`
: Tool to create and manipulate libraries (a.k.a. archive files).

`z8k-coff-as`
`z8k-pcos-as`
: The assembler.

`z8k-coff-c++`
`z8k-pcos-c++`
`z8k-coff-c++filt`
`z8k-pcos-c++filt`
`z8k-coff-g++`
`z8k-pcos-g++`
: C++ compiler and symbol demangler (`c++filt`). They are built as part of the build process, but aren't tested and probably don't work.

[1] http://en.wikipedia.org/wiki/COFF

`z8k-coff-gcc`
`z8k-pcos-gcc`
: The C compilers for COFF and PCOS.

`z8k-coff-gdb`
`z8k-pcos-gdb`
: Debugger, see (Chapter 8 [The debugger], page 39).

`z8k-coff-gprof`
`z8k-pcos-gprof`
: Display call graph profile data (untested).

`z8k-coff-ld`
`z8k-pcos-ld`
: COFF linker.

`z8k-coff-nm`
`z8k-pcos-nm`
: Lists symbols from object files.

`z8k-coff-objcopy`
`z8k-pcos-objcopy`
: Copy object files.

`z8k-coff-objdump`
`z8k-pcos-objdump`
: Display information from object files.

`z8k-coff-ranlib`
`z8k-pcos-ranlib`
: Generate archive (`*.a` files) index.

`z8k-coff-readelf`
`z8k-pcos-readelf`
: Displays information about ELF files.

`z8k-coff-run`
`z8k-pcos-run`
: Simulator.

`z8k-coff-size`
`z8k-pcos-size`
: List sections sizes of object files.

`z8k-coff-strings`
`z8k-pcos-strings`
: Print the strings of printable characters in files.

`z8k-coff-strip`
`z8k-pcos-strip`
: Discard symbols from object files.

3.2 Basic usage

If you followed the instructions in Section 2.2 [Installing the toolchain], page 2, you can invoke the C compiler by issuing `z8k-pcos-gcc` at the command prompt.

Let's do a simple example, like this little C program, `hello.c`:

```
#include <stdio.h>

int main(void)
{
        printf("Hello from the M20!\n");
        return 0;
}
```

Compile it with `z8k-pcos-gcc -o hello.cmd hello.c`:

```
$ z8k-pcos-gcc -o hello.cmd hello.c
$ ls -l hello.cmd
-rw-r--r-- 1 chris chris 16209 Mar  5 23:01 hello.cmd
$
```

`hello.cmd` is the executable generated by the compiler. You'll need to transfer it to the M20 in order to run it. See http://www.z80ne.com/m20/index.php?argument=sections/transfer/transfer.inc for ways to transfer the program to the M20.

3.3 Compiler Switches

The compiler is an rather old version of gcc (2.9). It was never an official release from the FSF[2], but came with the eCos tools from Cygnus Solutions[3].
You can refer to gcc documentation about the available command line switches. For example, refer to `/opt/z8kgcc-jan-19-2009/man/man1/z8k-pcos-gcc.1`[4] for an exhaustive list of command line switches.
Here's an overview of some useful switches when compiling for the Z8000:

`-O`
`-O1`
`-O2` Optimize for speed. `-O1` optimizes more and `-O2` optimizes even more.

`-Os` Optimize for size.

`-o` *output file*
Name of output file.

[2] http://www.fsf.org

[3] http://en.wikipedia.org/wiki/Cygnus_Solutions

[4] `z8k-pcos-gcc.1` is in ROFF format, use e.g. "`groff -Tascii -man z8k-pcos-gcc.1`" to view it in a human readable form.

`-c` Compile only, don't link.

`-S` Create assembler output file instead of object file or executable.

`-mstd` Use "standard call" calling convention for functions. This generates larger code and is slower than the default. It's used primarily for debugging with the simulator.
"Standard call" means passing all parameters to functions over the stack instead of using registers as much as possible.

`-mz8001` Generate code for segmented mode (only available on the Z8001, pointers are 32 bits, 23 bits of them are actually used). This is the default for `z8k-pcos-gcc`.

`-mz8002` Generate code for non-segmented mode (available both on the Z8001 and the Z8002, pointers are 16 bits). This is the default for `z8k-coff-gcc`. The PCOS runtime library does not support non-segmented mode.

`-mint16` Integers (`int` type) are 16 bits. This is the default.

`-mint32` Integers (`int` type) are 32 bits. Don't use it. It's not supported by the runtime library.

`-Wl,`*`linker options`*
Pass *`linker options`*, separated by commas, to the linker.
E.g. `-Wl,-stack,0x1000,-multi`

`-Wno-cpg` Disable *CPG* warnings. See Section 3.4 [*CPG* warnings], page 6.

3.4 *CPG* warnings

"CPG" are my initials.
I've fixed some problems in the compiler where I'm not 100% sure that they are correct. (I'm not really a `gcc` hacker.) Therefore, in order to keep users from trying to fix bugs in their programs which in fact might be introduced by my `gcc` changes, the compiler issues warnings when these changes are used.

These warnings look something like

```
<source_file>:<linenum>:warning: cpg machine description change #num is
being used, program may not work (disable warning with '-Wno-cpg')
```

`num` is in the range 1..4, and indicates which change was utilized. If you encounter such a warning and your program doesn't work, please contact me[5], provide your program and I'll check whether your program's defect comes from my compiler changes.

In order to get rid of the warning, use the `-Wno-cpg` command line switch.

[5] email address: `chris@groessler.org`

3.5 Predefined macros

The compiler predefines a "__Z8000__" macro. Depending on the compilation target (segmented or non-segmented) it also defines a "__Z8001__" or "__Z8002__" macro. With these macros the program's source code can adapt to different machines, e.g.

```
void function(void)
{
#ifdef __Z8000__
       ...  /* do some Z8000 specific stuff */
#else
       ...  /* do other stuff when not compiling for Z8000 */
#endif
}
```

If you compile with the `-mstd` switch, the macro `__STD_CALL__` is predefined.

> **Hint:** In order to see all predefined macros of the compiler, issue the command "`z8k-pcos-gcc -E -dM - < /dev/null`". You can add additional command line arguments like `-mstd`, `-mz8002`, or `-mint32` in order to see the effect of these switches to the macros.

Currently there is no predefined macro to distinguish between compilation for plain Z8000 (for COFF with `z8k-coff-gcc`) and the M20 (for PCOS with `z8k-pcos-gcc`).

3.6 Inline assembly

A basic introduction to gcc inline assembly can be found at e.g. http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html. The explanation there is x86 specific, but the basic syntax is the same as for the Z8000. One can specify the assembler code, a list of output operands, a list of input operands, and a clobber list. The clobber list is the list of registers whose values are modified by the assembler code.

Basic syntax:

```
__asm__ ( "assembler code"
          : output operands                  /* optional */
          : input operands                   /* optional */
          : list of clobbered registers      /* optional */
          );
```

The following operand modifiers are available with the Z8000 port:

`X` stack pointer name

Registers:

`Q`	byte sized register name
`U`	high byte of word register
`V`	low byte of word register
`H`	word register name
`I`	next word register name
`S`	
`B`	long register name
`T`	next long register name
`D`	quad register name
`P`	register name in size of pointer

Integers:

`O`	log two of value
`P`	inverted log two
`H`	bottom 16 bits
`I`	top 16 bits
`N`	negative
`B`	high 32 bits of 32bit number

Memory:

`I`	adjusted upwards by two
`T`	adjusted upwards by four

Address:

`H`	low 16 bits
`I`	high 16 bits
`A`	as long constant
`S`	same as A but with #

Misc:

`C`	conditional name
`D`	reverse conditional name
`F`	clear v flag if necessary

Here's a simple example of a function which reads a byte from an I/O port by means of the `inb` opcode:

```
unsigned char in(unsigned int portaddr)
{
    unsigned char retval;

    __asm__ volatile (
        "inb %Q0,@%H1 \n\t" : "=r" (retval) : "r" (portaddr));

    return retval;
}
```

In this example the `Q` and `H` modifiers are used to specify the sizes of the register operands.

The `volatile` keyword is in fact not needed here, but is included anyway to show it's use. It prevents the compiler to remove the `__asm__` statement when optimizing because it doesn't change anything the compiler knows about. In this example we have an output value (`retval`) which is used afterwards, therefore the assembler code cannot be skipped. But in other cases, where there is no output from the assembler (think a delay loop), `volatile` is required.

In Chapter 7 [Examples], page 25 there are some programs which demonstrate the usage of the inline assembler.

> **Hint:** If you want to know how the registers are assigned for an inline assembly block, compile the C program with the `-S` parameter and look at the generated assembly code to check the register assignments.

4 The C runtime library

This chapter describes the PCOS runtime library of `z8k-pcos-gcc`. The `z8k-coff-gcc` runtime library is an unmodified version of newlib[1]. The PCOS runtime library implements most functions of `newlib` with the notably exception of `opendir`, `readdir`, and `closedir`. The disk directory can nevertheless be accessed by using native PCOS functions[2].
In order to use the functions and defines described in this chapter, the header file `sys/pcos.h` has to be included in the C file. This header file is located at `<instdir>/z8k-pcos/include/sys/pcos.h`. Refer to Section 9.1.1.1 [Building GNU toolchain], page 40 for an explanation of `<instdir>`.

4.1 Floating Point

The C compiler itself does support floating point variables (`float` and `double`), but the `printf` and `scanf` function families of the runtime library don't support them. You can still print the integer part of a floating point variable by casting it to an `int`:

```
float f = 123.123;
printf("value of f: %d\n", (int)f);
```

This results in the following output:

```
value of f: 123
```

4.2 PCOS system functions

The runtime library provides access to most of the PCOS system functions. See chapter 8 ("THE M20 SYSTEM CALLS") of the PCOS assembler language user guide (Section A.2 [ASSEMBLER Language User Guide], page 45) for a description of the PCOS system functions and PCOS programming environment.

`sys/pcos.h` has to be included in order to get access to the definitions of the system functions.

List of supported PCOS functions:

```
int _pcos_dgetlen(int did, unsigned long *length);
int _pcos_dgetposition(int did, unsigned long *length);
int _pcos_dseek(int did, unsigned long offset);
int _pcos_resetbyte(int did);
int _pcos_eof(int did, unsigned int *status);
int _pcos_putbyte(int did, unsigned char byte);
int _pcos_getbyte(int did, unsigned char *byte);
int _pcos_writebytes(int did, const void *buffer,
                     unsigned int nbytes, unsigned int *retbytes);
```

[1] Version 1.12, http://sourceware.org/newlib

[2] An example is provided in Section 7.9 [Accessing the disk directory], page 36

```
int _pcos_readbytes(int did, const void *buffer,
                    unsigned int nbytes, unsigned int *retbytes);
int _pcos_readline(int did, const void *buffer,
                   unsigned int nbytes, unsigned int *retbytes);
int _pcos_new(unsigned short length, void **memory);
int _pcos_newsamesegment(unsigned short length, void **memory);
int _pcos_dispose(int length, void **memory);
int _pcos_drename(const char *from, int fromlen, const char *to,
                  int tolen);
int _pcos_dremove(const char *name, int namelen);
int _pcos_openfile(int did, const char *name, int namelen, int mode,
                   int extent_len);
int _pcos_close(int did);
int _pcos_ddirectory(const char *name, int namelen);
int _pcos_maxsize(unsigned short *maxsize);
int _pcos_search(int drive, int search_mode, int *length,
                 char **file_pointer, char *name_pointer);
void _pcos_selectcur(int mode);
void _pcos_cls(void);
int _pcos_crlf(void);
void _pcos_grfinit(int *color, void **pointer);
int _pcos_cleartext(unsigned int column, unsigned int row,
                    unsigned int xlen, unsigned int ylen);
int _pcos_scrolltext(unsigned int plane_mask, unsigned int function,
                     unsigned int src_x, unsigned int src_y,
                     unsigned int dst_x, unsigned int dst_y,
                     unsigned int xlen, unsigned int ylen);
int _pcos_bset(void *dest, unsigned char val, unsigned int len);
int _pcos_bwset(void *dest, unsigned short val, unsigned int len);
int _pcos_bclear(void *dest, unsigned int len);
int _pcos_bmove(void *dest, const void *src, unsigned int len);
int _pcos_dstring(char *string);
int _pcos_dhex(unsigned int word);
int _pcos_dhexbyte(unsigned char byte);
int _pcos_dhexlong(unsigned long byte);
int _pcos_dlong(unsigned long byte);
int _pcos_dnumw(unsigned int word, unsigned int field_width);
int _pcos_gettime(char *buf, unsigned int buflen);
int _pcos_getdate(char *buf, unsigned int buflen);
int _pcos_settime(char *buf, unsigned int buflen);
int _pcos_setdate(char *buf, unsigned int buflen);
int _pcos_lookbyte(int did, unsigned char *byte,
                   unsigned char *buffer_status);
int _pcos_chgwindow(unsigned int fgcolor, unsigned int bgcolor);
int _pcos_readcur0(cursor_shape *shape, unsigned int *blinkrate,
                   unsigned int *column, unsigned int *row);
int _pcos_readcur1(cursor_shape *shape, unsigned int *blinkrate,
```

```
                    unsigned int *x_pos, unsigned int *y_pos);
int _pcos_chgcur0(unsigned int column, unsigned int row);
int _pcos_chgcur1(unsigned int x_pos, unsigned int y_pos);
void _pcos_chgcur2(unsigned int blinkrate);
void _pcos_chgcur3(unsigned int blinkrate);
void _pcos_chgcur4(cursor_shape new_shape);
void _pcos_chgcur5(cursor_shape new_shape);
int _pcos_setcontrolbyte(int did, unsigned int word_number,
                         unsigned int word);
int _pcos_getstatusbyte(int did, unsigned int word_number,
                         unsigned int *word);
int _pcos_checkvolume(void);
```

4.3 PCOS status codes

The status codes are taken from appendix 'E' ("SYSTEM ERRORS") of the PCOS assembler language user guide (see Section A.2 [ASSEMBLER Language User Guide], page 45). The descriptions of the codes are a verbatim copy from this document.
These status codes are returned by the PCOS system functions (see Section 4.2 [PCOS system functions], page 10).

`sys/pcos.h` has to be included in order to get access to the definitions of the status codes.

Name	Value	Description
`PCOS_ERR_OK`	0	success
`PCOS_ERR_XXX`	3	invalid termination of input byte stream
`PCOS_ERR_MEM`	7	out of memory
`PCOS_ERR_INVADR`	9	invalid listener or talker address
`PCOS_ERR_NOIEEE`	10	no IEEE board
`PCOS_ERR_TO`	11	time out error
`PCOS_ERR_DATATYPE`	13	bad data type
`PCOS_ERR_NOWIN`	35	window does not exist
`PCOS_ERR_WINCREAT`	36	window create error
`PCOS_ERR_NOENT`	53	file not found

PCOS_ERR_MODE	54	bad file open mode
PCOS_ERR_ALOPN	55	file already open
PCOS_ERR_EIO	57	disk i/o
PCOS_ERR_EEXIST	58	file aready exists
PCOS_ERR_NOTINIT	60	disk not initialized
PCOS_ERR_NOSPC	61	disk filled
PCOS_ERR_EOF	62	end of file
PCOS_ERR_REC	63	bad record number
PCOS_ERR_NAME	64	bad file name
PCOS_ERR_VNOENT	71	volume name not found
PCOS_ERR_INVVOL	73	invalid volume number
PCOS_ERR_VOLNOTEN	75	volume not enabled
PCOS_ERR_PASSWD	76	password not valid
PCOS_ERR_DCHG	77	illegal disk change
PCOS_ERR_WRPROT	78	write protected file
PCOS_ERR_CPPROT	79	copy protected file
PCOS_ERR_PARM	90	error in parameter
PCOS_ERR_TOOMPARM	91	too many parameters
PCOS_ERR_NOTFND	92	command not found
PCOS_ERR_NOTOPM	96	file not open
PCOS_ERR_BADLOAD	99	bad load file
PCOS_ERR_TIMDAT	101	time or date
PCOS_ERR_EXFN	106	function key already exists

PCOS_ERR_CALLUSR	108	call-user
PCOS_ERR_TO2	110	time-out
PCOS_ERR_INVDEV	111	invalid device

4.4 PCOS open modes

The open modes are taken from page 8.15 of the PCOS assembler language user guide (see Section A.2 [ASSEMBLER Language User Guide], page 45). They are passed to the `_pcos_openfile` function as `mode` parameter.

`sys/pcos.h` has to be included in order to get access to the definitions of the open modes.

Name	Value
PCOS_OPEN_READ	0
PCOS_OPEN_WRITE	1
PCOS_OPEN_RDWRITE	2
PCOS_OPEN_APPEND	3

4.5 PCOS DID defines

DID stands for "Device ID". It's passed to many PCOS system functions to specify the device or file to operate on. See the `did` parameter in the function prototypes.
The DID codes are taken from appendix 'D' ("DEVICE ID (DID) ASSIGNMENTS") of the PCOS assembler language user guide (see Section A.2 [ASSEMBLER Language User Guide], page 45).

`sys/pcos.h` has to be included in order to get access to the definitions of the DID defines.

Name	Value
DID_CONSOLE	17
DID_PRINTER	18
DID_COM	19
DID_COM1	25
DID_COM2	26

4.6 Special characters

`sys/pcos.h` has to be included in order to get access to the definitions of the special characters.

Name	Value	Key
`PCOS_CH_CURS_DOWN`	154	Shift + keypad 2
`PCOS_CH_CURS_UP`	158	Shift + keypad 8
`PCOS_CH_CURS_LEFT`	155	Shift + keypad 4
`PCOS_CH_CURS_RIGHT`	157	Shift + keypad 6
`PCOS_CH_DEL`	8	Control + H
`PCOS_CH_TAB`	9	Control + I
`PCOS_CH_DELCHR`	4	Control + D
`PCOS_CH_ESC`	221	
`PCOS_CH_STOP`	3	Control + C
`PCOS_CH_EOL`	13	
`PCOS_CH_ENTER`	13	

4.7 Creating files with `open()`

When a file is created in PCOS (with the `_pcos_openfile` system function), a parameter (`extend_len`) is given which specifies how many sectors to preallocate for the file. The `open()` call doesn't have such a parameter, therefore the PCOS runtime library uses the value of a global variable for the numbers of sectors to preallocate. This variable is initialized to 4, but can be set by the user program prior to the `open()` call or by overriding it with its own define.

The `sys/pcos.h` file provides a definition of this variable:

```
extern unsigned short _pcos_extent_length;
```

To override it globally within your program, the suggested method is to provide an initialized variable `_pcos_extent_length` in your program, e.g. like

```
unsigned short _pcos_extent_length = value;
```

```
int main(void)
{
        ...
}
```

You can also set it before each call to `open` (or `fopen`):

```
_pcos_extent_length = other_value;
fd = open(...);
```

Keep in mind that `_pcos_extent_length` is a global variable, therefore after an assignment to it all subsequent calls to `open` will use the last value assigned to it.

5 The assembler

The assembler is the one from GNU binutils (see http://www.gnu.org/software/binutils). Please refer to its documentation for detailed information. This section will only outline the most important differences compared to the Zilog or Olivetti assemblers. Some examples of assembly language programs can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/misc/asm-snippets/binutils and in the runtime library source code (Section 9.1.1.3 [PCOS specific parts of the runtime library], page 42).

> **Note:** The assemblers (both the PCOS and COFF versions) generate object files in COFF format. At link time the PCOS linker creates PCOS compatible executable files from the COFF input object file(s).

5.1 Radix representation

Binary values are prefixed by "0b", octal values are prefixed by "0", and hexadecimal values are prefixed by "0x". For example this source file, `x.s`:

```
.z8001
.text
ld      r0,#12
ld      r0,#0b0110
ld      r0,#0x12
ld      r0,#012
.end
```

Assembling it with

```
$ z8k-pcos-as -o x.o x.s
```

results in this object file:

```
x.o:     file format coff-z8k


Disassembly of section .text:

00000000 <.text>:
   0:   2100 000c       ld      r0,#0xc
   4:   2100 0006       ld      r0,#0x6
   8:   2100 0012       ld      r0,#0x12
   c:   2100 000a       ld      r0,#0xa
```

(Use "`z8k-pcos-objdump -d x.o`" to view the disassembly.)

5.2 Segment notation

The assembler doesn't know about the `<<segment>>` notation to indicate a segmented address. Segmented addresses are expressed as 32bit values (where the highest bit and the second byte of the address are ignored by the processor).

So in order to load the address of segment 2, offset 0x10 into the register RR2, use the following statement

```
ldl     rr2,#0x02000010
```

instead of

```
ldl     rr2,#<<2>>%10
```

(which is the equivalent syntax of the Olivetti assembler).

5.3 Comments

Comments are prefixed by an exclamation mark ("`!`"), instead of an asterisk ("`*`"). Comments after an assembly statement in the same line in contrast to the Olivetti assembler also need a preceeding "`!`" character.

5.4 Mixing C and assembly

This chapter describes how parameters are passed from C to assembly subroutines and how the results are returned.

> **Hint:** If you are not sure about how the parameters of a given function are passed, compile the C program with the `-S` parameter and look at the generated assembly code to determine the exact locations of the parameters.

Another way to mix C and assembly is the inline assembler of the C compiler, see Section 3.6 [Inline assembly], page 7.

5.4.1 Z8001

5.4.1.1 Z8001 default calling convention

Registers R2 to R7 are used for parameter passing. The first argument to a function is passed in R7, the second in R6, and so on until R2. If more parameters are present than available registers, the remaining parameters are passed on the stack. `char` parameters consume a whole register (the lower part), therefore a function which has 2 `char` parameters uses R7 and R6 as input registers. `long` parameters consume 2 registers, RR6, RR4, or RR2. If the first parameter is a `char`, `short`, or `int`, and the second a `long`, R7 will be used for the first parameter and RR4 for the second. R6 will be unallocated in this case.

The return value of a function is passed in the R2 (`char` or `int` or `short`) or RR2 (`long` or pointers) register.

Registers R8 to R13 must be preserved by the called function.

5.4.1.2 Z8001 "Standard" calling convention

The stack is used to pass parameters. The parameters are pushed on the stack starting from the rightmost parameter until the leftmost parameter. `char` parameters will be pushed as a word (16bit).

The return value of a function is passed in the R7 (`char` or `int` or `short`) or RR6 (`long` or pointers) register.

Registers R8 to R13 must be preserved by the called function.

5.4.2 Z8002

This chapter will be provided in a future revision of this document.

6 ldpcos - the PCOS linker

ldpcos is the only program of the toolchain which knows about the PCOS executable file format. All other programs (assembler, linker, archiver) operate on COFF format files. ldpcos uses the COFF linker (z8k-pcos-ld) and other tools (z8k-pcos-objdump and z8k-pcos-size) to build a COFF executable where the .text, .data, and .bss sections are adjacent[1]. From this COFF executable it then creates the PCOS executable by copying the sections and adding relocation information[2].

6.1 Command line switches

Start ldpcos without any parameters to get a list of available command line switches:

```
$ ldpcos
$Id: ldpcos.c,v 1.44 2006-11-30 23:09:20 chris Exp $
(c) Copyright 2001-2006 Christian Groessler, GPL license
Compiled at Jan 20 2009
ldpcos: usage: ldpcos <options> <object files>
        options are:
                -v              be verbose
                                (use up to three times for more verbosity)
                -f              fill bss with zeroes
                -stack value    reserve additional stack space
                -farentry       entry point is more than 256 bytes away
                -o outfile      set output file name
                -c configfile   specify config file name
                -map mapfile    set map file name
                -l linker       specify linker to use
                                (z8k-pcos-ld)
                -a assembler    specify assembler to use
                                (z8k-pcos-as)
                -b objcopy      specify objcopy program to use
                                (z8k-pcos-objcopy)
                -s size         specify size program to use
                                (z8k-pcos-size)
                -data value     specify start of data section
                                for section size test (0xa000)
                -bss value      specify start of bss section
                                for section size test (0x4000000)
                -save-temps     do not delete intermediate files
                -sav            make a .sav file
                -multi          create multiple memory load chunks
                                (.text, .data, .bss)
```

[1] This is done for non -multi links only. It's not needed for -multi links.

[2] ldpcos requires the linker (z8k-pcos-ld) from this distribution. A z8k-pcos-ld from the generic binutils release doesn't work, since it doesn't support the --pcos-relocs command line parameter to write out the relocation information.

```
                    -raw            don't create default PCOS program prologue
    $
```

Description of the individual command line switches:

`-v` Displays information about the linking process. You can give it more than once in order to get increasingly more information.

`-f` The `.bss` section is normally not part of the executable file. With this switch `ldpcos` will include it in the executable file. This switch is mainly useful for debugging purposes.

`-stack` The stack size reserved for the program is 0x1DE (PCOS default), unless you specify a different size with this parameter. The C compiler's `specs` file will reserve 0x800 bytes for the stack using this switch. See Section 9.1.1.2 [Building the PCOS linker], page 41.

`-farentry`
`ldpcos` by default creates a PCOS conforming program prologue[3]. This prologue requires the program's entry point to be within the first 256 bytes of the program. If this isn't the case for the program at hand you can overcome this restriction with this command line switch. See Section 6.5 [Default PCOS program prologue], page 23.

`-o` Specifies the name of the output file. Typically something like *`prog`*`.cmd` or *`prog`*`.sav`.

`-c` Specify config file for linking. See Section 6.2 [Config file], page 22.

`-map` Create a map file.

`-l`
`-a`
`-b`
`-s` With these switches you can override the backend tools `ldpcos` is going to use. They default to `z8k-pcos-ld`, `z8k-pcos-as`, `z8k-pcos-objcopy`, and `z8k-pcos-size`. These switches are primarily useful for debugging.

`-data` Only used for non "`-multi`" operation: When creating the initial executable which is used to size the different sections, use this value for the start of the `.data` section. The default value is 0xA000.

`-bss` Only used for non "`-multi`" operation: When creating the initial executable which is used to size the different sections, use this value for the start of the `.bss` section. The default value is 0x4000000.

`-save-temps`
This switch tells `ldpcos` not to delete intermediate files which are created during the section size tests. Useful for debugging `ldpcos`.

[3] See pg. 2-28ff of the assembler language user guide (Section A.2 [ASSEMBLER Language User Guide], page 45).

`-sav` Create a `.sav` file instead of a `.cmd` file. See Section 6.3 [.sav files], page 22.

`-multi` Create multiple memory load chunks (`.text`, `.data`, `.bss`). Use this for programs which are bigger than 64K. See Section 6.4 [Big Programs], page 22.

`-raw` Don't create default PCOS program prologue. See Section 6.5 [Default PCOS program prologue], page 23.

6.2 Config file

The backend tools to be used and the program's description can be specified in a config file. The backend tools can also be specified with the `-a`, `-b`, `-l`, and `-s` command line switches. If one of these switches appears on the command line together with a config file which also overrides the same backend tool, the order in the command line is important. The last occurrence is the one which will finally be used.

```
# Comment lines start with a "#"
programid = "Hello World Rev. 1.0"
objcopy = /bla/z8k-pcos-objcopy
linker = z8k-pcos-ld.new
getsize = /bla/z8k-pcos-getsize
assembler = my-special-asm
```

Empty lines are ignored. All lines are optional, so a typical config file might look like

```
# config file for hello world program
programid = "Hello World Rev. 1.0"
```

The program description (the string specified by `programid`) is displayed when the program is loaded resident by use of the `PLOAD` PCOS command[4]. The program description is ignored when using the `-raw` switch. If there is no config file specified or no `programid` line in the config file, the description of the program defaults to "Executable generated by ldpcos $Revision: *x.y* $", where *x* and *y* denote the major and minor version of `ldpcos`.

6.3 .sav files

`.sav` executable files are kept in memory after the first run. `ldpcos` creates such a file if you give the `-sav` command line parameter. `.sav` files, after having been loaded, cannot be unloaded again. The system has to be rebooted in order to get rid of them. Therefore they are normally only used for device drivers or other low level system code. Regular (`.cmd`) programs can also be made memory resident by the use of the PCOS `PLOAD` command. But, different compared to `.sav` files, they can be unloaded again with the `PUNLOAD` command.

6.4 Big Programs

By default `ldpcos` places all sections of a program (`.text`, `.data`, `.bss`) into one load segment. Since the Z8000 has 64K segments, this limits the total program size to 64K[5].

[4] See section 6 of the PCOS User Guide (Section A.1 [PCOS User Guide], page 45).

[5] A bit less than 64K since PCOS requires some management data inside the segment.

The -multi switch (for "multi"ple segments) puts each section into its own load segment. PCOS takes care of the loading and if the program size is less than 64K, the sections still may end up in the same Z8000 segment. But it allows programs to be as big as 192K, if each of the sections are 64K.
When using the -multi switch, the -data and -bss switches are ignored (with a warning). Reason is, that with multiple load segments ldpcos doesn't know the relative location of the .text, .data, and .bss sections (since they are allocated at program load time). Therefore PC relative addressing of items in the .data and .bss sections is not possible. The -data and -bss switches are used to fine tune the test link step of ldpcos in which it finds out the sizes of each section when doing a non-"multi" link.

6.5 Default PCOS program prologue

ldpcos creates a program header as described at pages 2-31ff in the assembler language user guide (see Section A.2 [ASSEMBLER Language User Guide], page 45). The program header's source code looks like this:

```
                .z8001
                .text
                .globl  __entry

                .word   0
__entry:        jr      t,_start
__program_id:   .asciz  "program identification string\r"
                .end
```

This code snippet is compiled in the background and then linked as the first object file. The file starts with a 16 bit zero value indicating the type of the program. The next location (__entry) gets called by PCOS after loading the program. It jumps to a label called _start. This label is the start of the user program, see Section 7.1 [Assembler version of "Hello World"], page 25 for an example.

Note the jr opcode in the first instruction at _entry. It requires the _start label to be not farer away than 256 bytes. If for some reason the entry point of the program is farer away, the -farentry command line switch to ldpcos lets it generate a slighty different prologue, as shown here:

```
                .z8001
                .text
                .globl  __entry

                .word   0
__entry:        jr      t,__start
__program_id:   .asciz  "program identification string\r"
                .even
__start:        jp      t,_start
                .end
```

This one jumps over the program id string and then jumps with `jp` to the real entry point.

The program id string at `__program_id` comes from the `programid` line of the config file. If no config file or a config file with no `programid` line is specified, it's set to a default string (see Section 6.2 [Config file], page 22).

You can tell `ldpcos` to not generate a default program prologue by passing it the `-raw` command line switch. Then your program has to provide the program header itself. See Section 7.2 [Assembler version of "Hello World" (`-raw` version)], page 27 for an example.

7 Examples

This chapter contains some examples.

7.1 Assembler version of "Hello World"

This is a simple assembler "Hello World" program:

```
!
! simple hello world test by CPG
!
                .z8001
                .data

msg:            .asciz  "simple \"Hello World\" by CPG\r"

                .text
                .even
                .globl  _start

! entered from PCOS (in fact, from the default program prologue)
_start:         pop     r0,@sp            ! get # of command line args

! throw cmd line args from stack (see p.2-37 asm manual)
                clr     r2
                ld      r3,r0
                sll     r3,#2
                addl    sp,rr2

                ldar    rr12,msg           ! address of message string
                sc      #0x59              ! PCOS: DString

                ret

                .end
```

Please note that we have a `.data` and a `.text` section in this program. Also, the string to display (`msg`) ends with a `\r` character (end-of-line for PCOS). And you can include a '"' character in a string by "backslashing" it.

Let's compile it:

```
$ z8k-pcos-as hello.s -o hello.o
$ ldpcos -o hello.cmd hello.o
hello.o:fake:(.text+0xe): relocation truncated to fit: r_rel16 against ‘msg’
$
```

Oops, we've got an error. The problem is the `ldar` opcode which loads the address of `msg` into `rr12`. If we would write `lda` instead of `ldar`, it would work (try it!).
The reason is that `ldpcos` does a test link in order to find out the sizes of the different program sections (`.text`, `.data`, `.bss`). For this test it assumes the `.data` section to start at 0xA000 (`.text` starts at 0). Since the `ldar` opcode can only access data in the range of -0x8000..0x7FFF, the address of `msg` somewhere at 0xA000 is out of range.
But we know that in *this* program the size of the `.text` section is definitely nowhere near 0xA000. So we can tune `ldpcos`'s size check by telling it that `.data` should start at e.g. 0x3000:

```
$ z8k-pcos-as hello.s -o hello.o
$ ldpcos -data 0x3000 -o hello.cmd hello.o
$
```

No error, we have now a `hello.cmd` executable for the M20.

7.1.1 Setting the program id string

When we load the previous program with the PCOS `PLOAD`[1] command, we get

```
1> pl hello.cmd
 Disk file name = hello.cmd
 Program name   = Executable generated by ldpcos $Revision: 1.33 $
 Operation Mode = Segmented / System
 Main entry = <0A>%D474; Init entry =  --None--
 Memory allocated:
    Block No. %0A; Starting address = <0A>%D472; Size = %0068
1>
```

In order to have a more descriptive "Program name", use the following config file (`hello.cfg`):

```
ProgramID = "Simple \"Hello World\""
```

Compile with

```
$ z8k-pcos-as hello.s -o hello.o
$ ldpcos -data 0x3000 -c hello.cfg -o hello.cmd hello.o
$
```

Loading it with `PLOAD` shows our new "program name":

[1] See section 6 of the PCOS User Guide (Section A.1 [PCOS User Guide], page 45).

```
1> pl hello.cmd
 Disk file name = hello.cmd
 Program name   = Simple "Hello World"
 Operation Mode = Segmented / System
 Main entry = <0A>%D490; Init entry =  --None--
 Memory allocated:
    Block No. %0A; Starting address = <0A>%D48E; Size = %004C
1>
```

7.2 Assembler version of "Hello World" (-raw version)

This is a modified version of the previous example, which doesn't use `ldpcos`' default program prologue, but provides its own:

```
!
! simple hello world test by CPG (raw version)
!
                .z8001
                .data

msg:            .asciz  "simple \"Hello World\" by CPG\r"

                .text
                .even

! *** prologue start
                .word   0
! entered from PCOS
                jr      t,mystart
                .asciz  "Simple \"Hello World\"\r" ! prog id string
                .even
! *** prologue end

mystart:        pop     r0,@sp     ! get # of command line args

! throw cmd line args from stack (see p.2-37 asm manual)
                clr     r2
                ld      r3,r0
                sll     r3,#2
                addl    sp,rr2

                ldar    rr12,msg    ! address of message string
                sc      #0x59       ! PCOS: DString

                ret

                .end
```

Compile with

```
$ z8k-pcos-as helloraw.s -o helloraw.o
$ ldpcos -raw -data 0x3000 -o hellor.cmd helloraw.o
$
```

7.3 Direct screen access (assembler)

The screen memory in the M20 is located at segment #3. This example "flickers" the screen by repeatedly writing all 0s and 1s to the screen memory bits. It assumes a monochrome display.

```
!
! "flicker" the screen ten times
!

        .z8001
        .text
        .even
        .globl  _start

! entered from PCOS (in fact, from the default program prologue)
_start: pop     r0,@sp          ! get # of command line args

! throw cmd line args from stack (see p.2-37 asm manual)
        clr     r2
        ld      r3,r0
        sll     r3,#2
        addl    sp,rr2

! now the program's guts:
        ldl     rr6,#0x03000000 ! setup pointer to screen memory
                                ! screen memory is in segment #3
        ldk     r4,#10          ! ten times

loop:   clr     r0              ! fill with 0 (black)
        calr    fillscr
        calr    delay           ! short delay
        dec     r0,#1           ! fill with 255 (white)
        calr    fillscr
        calr    delay           ! short delay
        djnz    r4,loop

        ret

! subroutine: fill screen memory with value of rl0
```

```
fillscr:ld      r5,#0x2000-1    ! screen memory (monochrome)
                                ! is 16k in size, count in words

        ld      @rr6,r0         ! fill first word
        ldl     rr8,rr6
        inc     r7,#2           ! rr8 points to 2nd word of
                                ! screen memory
        ldir    @rr6,@rr8,r5    ! fill the complete memory
        ret

! subroutine: small busy loop delay routine
delay:  ldl     rr8,#0x20000
deloop: djnz    r7,deloop
        djnz    r8,deloop
        ret

        .end
```

Compile with

```
$ z8k-pcos-as flicker.s -o flicker.o
$ ldpcos -o flicker.cmd flicker.o
$
```

7.4 Direct screen access (C)

This program does the same as the previous example, but now it's written in C (`flicker.c`):

```
/*
 * "flicker" the screen ten times
 */

/* pointer to screen memory, segment #3 */
unsigned short *screen = (unsigned short *)0x3000000;

/* fill screen memory with "value" */
static int fillscr(unsigned short value)
{
    int i;

    for (i = 0; i < 0x2000; i++)
        *(screen + i) = value;
}

/* small busy loop delay routine */
static void delay(void)
```

```
{
    unsigned long i = 0x20000;

    while (i--)
        ;
}

int main(void)
{
    int i;

    for (i = 0; i < 10; i++) {
        fillscr(0);
        delay();
        fillscr(0xffff);
        delay();
    }
    return 0;
}
```

Compile with

```
$ z8k-pcos-gcc -o cflicker.cmd cflicker.c
$
```

Try to add `-O2` to the compiler switches in order to enable optimizations and compare it with the version without `-O2`. The difference in speed is noticeable! Also compare it with the assembler version.

7.5 Direct screen access (C with assembler subroutine)

If you followed the last two examples, you've noticed that in order to get good performance, some parts of the program might need to be written in assembler. In this example we accelerate the C program of the previous example by providing an assembler implementation of the most time consuming function (`fillscr()`).

Here is the modified C source file (`aflicker.c`):

```
/*
 * "flicker" the screen ten times (using an external
 * assembly language subroutine to fill screen memory)
 */

/* fill screen memory with "value" */
extern void fillscr(unsigned short value);
```

```
/* small busy loop delay routine */
static void delay(void)
{
    unsigned long i = 0x20000;

    while (i--)
        ;
}

int main(void)
{
    int i;

    for (i = 0; i < 10; i++) {
        fillscr(0);
        delay();
        fillscr(0xffff);
        delay();
    }
    return 0;
}
```

The definition of the `fillscr()` function has been removed and was replaced by an external declaration of it with the same parameters and return value.

This assembler source file provides the implementation of the new `fillscr()` function (`aflicker.S`):

```
!
! fillscr() function to fill the screen memory with
! a given value
!
! extern void fillscr(unsigned short value);
!
        .z8001
        .text
        .even
        .globl  _fillscr

_fillscr:

#ifdef __STD_CALL__
        ld      r7,rr14(#4)     /* get "value" parameter */
#else
/*      ld      r7,r7           if not _STD_CALL__, first
                                parameter is passed in r7 */
```

```
#endif

/* registers r0..r7 don't need to be preserved
 * across function calls
 */
        ldl     rr4,#0x03000000 ! setup pointer to screen memory

! fill screen memory with value of rl0
        ld      r1,#0x2000-1    ! screen memory (monochrome)
                                ! is 16k in size, count in words

        ld      @rr4,r7         ! fill first word
        ldl     rr2,rr4
        inc     r5,#2           ! rr4 points to 2nd word of
                                ! screen memory
        ldir    @rr4,@rr2,r1    ! fill the complete memory
        ret
```

Compile with

```
$ z8k-pcos-gcc -O2 -o aflicker.cmd aflicker.c aflicker.S
$
```

or with `-mstd` to use "standard call" calling convention

```
$ z8k-pcos-gcc -mstd -O2 -o aflickerstd.cmd aflicker.c aflicker.S
$
```

There are some points to note here:

- When compiling, the assembler source file is passed directly to the `z8k-pcos-gcc` compiler driver, together with the C source file. `gcc` by default will invoke the assembler to translate files ending with `.s` or `.S`.
- The difference between files ending with lowercase "s" or uppercase "S" is that for uppercase "S" the C preprocessor gets invoked to process the file before it is passed on to the real assembler. Therefore it is possible to use C comments and preprocessor macros in assembly language source files (like demonstrated in the example above). Lowercase "s" files will be handed over to the assembler without preprocessing.
- The assembler code adapts to the calling convertion of the C code by means of the "`#ifdef __STD_CALL__`" clauses. See also Section 3.5 [Predefined macros], page 7.
- C symbols have an underscore prepended, so the C function `fillscr()` refers to the assembler symbol `_fillscr`.

7.6 Direct screen access (C with inline assembly)

Instead of using a separate assembler source file one can use the inline assembler of the C compiler.

Here's a "flicker" version which uses inline assembly (`ciflicker.c`):

```
/*
 * "flicker" the screen ten times (using inline
 * assembly for fillscr())
 */

/* pointer to screen memory, segment #3 */
unsigned short *screen = (unsigned short *)0x3000000;

/* fill screen memory with "value" */
static int fillscr(unsigned short value)
{
    /* Scratch variables needed to assign the registers used
       by the inline assembly part. */
    unsigned short scratch0, scratch1;
    unsigned long scratch2, scratch3;

    __asm__ volatile ("ldl   %S3,%5         \n\t"
                      "ld    %H1,#0x2000-1  \n\t"
                      "ld    @%S3,%H4       \n\t"
                      "ldl   %S2,%S3        \n\t"
                      "inc   %I3,#2         \n\t"
                      "ldir  @%S3,@%S2,%H1  \n\t"
                      : "=r" (scratch0), "=r" (scratch1),
                        "=r" (scratch2), "=r" (scratch3)
                      : "0" (value), "m" (screen)
                      : "memory" );
}

/* small busy loop delay routine */
static void delay(void)
{
    unsigned long i = 0x20000;

    while (i--)
        ;
}

int main(void)
{
    int i;

    for (i = 0; i < 10; i++) {
        fillscr(0);
```

```
        delay();
        fillscr(0xffff);
        delay();
    }
    return 0;
}
```

Please note the `scratchX` variables. They are used to allocate the registers used internally by the assembler routine. We could have written them explicitly, like '`ldl rr2,%5`' instead of '`ldl %S3,%5`' to load `screen` into `rr2`, but the compiler wouldn't know that we use `rr2` inside the assembly block. This would be a problem if the compiler holds some value in `rr2` which gets destroyed by the inline assembler code. With the usage of the `scratchX` variables the compiler takes care about the assignment of the registers and no register will change its value without the compiler's notice. When compiling with optimization enabled the `scratchX` variables also won't use any memory or stack space since they are not used afterwards.

7.7 Read a byte from a port

This C program (`pinb.c`) with inline assembly displays the contents of an I/O port:

```
/*
 * read a byte from a port and display it
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <limits.h>

int main(int argc, char **argv)
{
    unsigned long portaddr;
    char *endptr;
    unsigned char value;

    if (argc != 2) {
        fprintf(stderr, "usage: pinb <port address>\n");
        return 1;
    }

    /* get port address */
    portaddr = strtoul(*(argv + 1), &endptr, 0);
    if (portaddr > 0xffff || *endptr) {
        fprintf(stderr, "invalid port address!\n");
```

```
        return 1;
    }

    printf("reading from port 0x%04lx (%lu)", portaddr, portaddr);
    if (! (portaddr & 1))
        printf("\t\tWARNING: even address!");
    printf("\n");

    __asm__ volatile (
        "inb %Q0,@%H1 \n\t" : "=r" (value)
                            : "r" ((unsigned int)portaddr));

    printf("Port value: 0x%02x  (%u)\n", value, value);

    return 0;
}
```

7.8 Write a byte to a port

This C program (`poutb.c`) with inline assembly writes a value to an I/O port:

```
/*
 * write a byte to a port
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <limits.h>

int main(int argc, char **argv)
{
    unsigned long portaddr;
    char *endptr;
    unsigned long value;

    if (argc != 3) {
        fprintf(stderr, "usage: poutb <port address> <value>\n");
        return 1;
    }

    /* get port address */
    portaddr = strtoul(*(argv + 1), &endptr, 0);
```

```
    if (portaddr > 0xffff || *endptr) {
        fprintf(stderr, "invalid port address!\n");
        return 1;
    }

    /* get value */
    value = strtoul(*(argv + 2), &endptr, 0);
    if (value > 0xff || *endptr) {
        fprintf(stderr, "invalid value!\n");
        return 1;
    }

    printf("writing 0x%02lx (%lu) to port 0x%04lx (%lu)",
           value, value, portaddr, portaddr);
    if (! (portaddr & 1))
        printf("\t\tWARNING: even port address!");
    printf("\n");

    __asm__ volatile (
        "push @sp,r7   \n\t"
        "ld r7,%H1     \n\t"
        "outb @%H0,rl7 \n\t"
        "pop r7,@sp    \n\t": : "r" ((unsigned int)portaddr),
                                "r" ((unsigned int)value));

    return 0;
}
```

You might be curious why this '`push @sp,r7`' and '`pop r7,@sp`' sequence is used, instead of a single line '`outb @%H0,%H1`'.
The reason is that the compiler might allocate a register above `r7` for `value`. This would generate an invalid byte register access (like e.g. '`rl11`'). It's a deficiency of the inline assembler that it doesn't handle byte register restrictions correctly.

7.9 Accessing the disk directory

The runtime library doesn't implement `opendir` and related functions. Nevertheless it's possible to read a disk's directory by means of direct PCOS calls. The following program prints a list of files on a disk.

```
/*
 * dir.cmd -- display directory contents using
 *            PCOS system calls
 */

#include <stdio.h>
#include <string.h>
```

```
#include <sys/pcos.h>

char name_buf[32];  /* file name buffer */

int main(int argc, char **argv)
{
    int retval;
    int length;   /* length of found filename */
    int rlength;  /* length of filename/search mask */
    char *search_name;
    char *file_pointer;
    int drive;
    int search_mode = 1;  /* search from beginning */

    if (argc > 2) {
        printf("usage: dir <filemask>\n");
        return 1;
    }

    /* if no argument given, search the default drive;
     * with argument, parse the drive number and use
     * the remainder of the string for the file mask
     */
    if (argc == 2) {
        if (*(*(argv + 1) + 1) == ':') {
            drive = **(argv + 1) - '0';
            if (strlen(*(argv + 1)) > 2) {
                rlength = strlen(*(argv + 1)) - 2;
                search_name = *(argv + 1) + 2;
            }
            else {  /* something like "0:" */
                rlength = 0;
                search_name = NULL;
            }
        }
        else {  /* no drive specified */
            drive = -1;  /* search default drive */
            rlength = strlen(*(argv + 1));
            search_name = *(argv + 1);
        }
    }
    else {
        drive = -1;  /* search default drive */
        rlength = 0;
        search_name = NULL;
    }
```

```
    /* search for files */
    while (1) {
        length = rlength;
        file_pointer = name_buf;
        retval = _pcos_search(drive, search_mode, &length,
                              &file_pointer, search_name);
        if (retval != PCOS_ERR_OK) break;
        search_mode = 0;   /* from now on search from the
                              last file found */
        name_buf[length] = 0;  /* zero terminate name */
        printf("found %s\n", name_buf);
    }

    if (retval != PCOS_ERR_NOENT) {
        printf("_pcos_search returned error %d\n", retval);
        return 1;
    }

    return 0;
}
```

Compile with e.g.

```
$ z8k-pcos-gcc -o dir.cmd dir.c
$
```

The program accepts a command line argument which specifies the file mask to search. Some examples:

```
1> dir
1> dir *.cmd
1> dir 0:ba*.cmd
```

The parsing of the command line is very simplistic. If the second character is a ':' the program assumes a mask which includes a disk drive and acts accordingly.

PCOS provides a system call to parse a file or volume name (DisectName, #96), but currently the runtime library doesn't implement an interface to this request.
`_pcos_disectname` will be available in a future release of the toolchain.

8 The debugger

This chapter will be provided in a future revision of this document.

9 Building from source

9.1 Compiler and Assembler

The source release can be built for 2 different targets, PCOS and COFF. If you want to create programs for the M20 you only need the PCOS version. The COFF version is a more generic one and creates executables for the simulator (`z8k-coff-run`), but cannot create M20 PCOS files.

9.1.1 Building for PCOS

Building the PCOS version is a two step process:

- build the GNU toolchain consisting of binutils and gcc
- build the PCOS linker

9.1.1.1 Building GNU toolchain

The source code comes in the archive `z8kgcc-jan-19-2009.tar.bz2`. Extract the source code into a directory (from now on referred to as *<srcdir>*). Then create somewhere else a "build" directory (*<builddir>*). *<instdir>* indicates the directory where you want the compiler to be installed, and *<archive location>* refers to the directory where the downloaded `z8kgcc-jan-19-2009.tar.bz2` file is located.

```
$ mkdir <srcdir>
$ cd <srcdir>
$ bzip2 -dc <archive location>/z8kgcc-jan-19-2009.tar.bz2 | tar -xf -
$ mkdir <builddir>
$ cd <builddir>
```

Then "configure" and build the toolchain:

```
$ <srcdir>/src/configure --prefix=<instdir> --target=z8k-pcos \
--enable-target-optspace
$ make
```

`--enable-target-optspace` tells the configure machinery to compile the runtime library with `-Os` (optimize for size). This results in smaller programs which is normally good for a memory restrained system like the M20. But it's not needed for correct operation of the runtime library. So you can omit it if you wish.

After the compilation has finished, install the newly created programs:

```
$ make install
```

The first step is done now.

9.1.1.2 Building the PCOS linker

The source code comes in the archive `ldpcos-jan-19-2009.tar.bz2`. Extract the tar file and type `"make"` in the `ldpcos/ldpcos` directory. The defaults of the makefile are for a 32bit little endian machine which supports unaligned memory accesses. If your machine is different, adjust the `"make"` command line accordingly:

big endian:

```
$ make COPTS="-O2 -D_CPG_BIG_ENDIAN_"
```

64 bit system:

```
$ make COPTS="-O2 -D__64BIT__"
```

no unaligned:

```
$ make COPTS="-O2 -D_CPG_NO_UNALIGN_"
```

Combine as needed, e.g. for a 64bit big endian machine which supports unaligned accesses:

```
$ make COPTS="-O2 -D__64BIT__ -D_CPG_BIG_ENDIAN_"
```

After successful compilation install the `ldpcos` executable as default linker for the C compiler:

```
$ cp ldpcos \
<instdir>/lib/gcc-lib/z8k-pcos/2.9-ecosSWtools-990319-m20z8k-3/ld
```

The destination path, especially the "`2.9-ecosSWtools-990319-m20z8k-3`" part might be different in newer versions of the tools. If you intend to use the assembler it is recommended to put `ldpcos` additionally into the `bin` directory:

```
$ cp ldpcos <instdir>/bin
```

The last step is to adjust the default stack size of C programs. Edit the `<instdir>/lib/gcc-lib/z8k-pcos/2.9-ecosSWtools-990319-m20z8k-3/specs` file and add `"-stack 0x800"` to the link parameters.

Here's an example diff:

```
--- specs       2009-01-22 22:28:09.000000000 +0100
+++ specs.new   2009-01-22 22:28:00.000000000 +0100
@@ -17,7 +17,7 @@


 *link:
-%{!mz8002:-m z8001}
```

```
+%{!mz8002:-m z8001} -stack 0x800

 *lib:
 -lc
```

To illustrate the change, here are the contents of the `link` section of the `specs` file before the change:

```
*link:
%{!mz8002:-m z8001}
```

and these are the contents after the change:

```
*link:
%{!mz8002:-m z8001} -stack 0x800
```

This gives a default stack size of 2048 bytes. You can override the stack size at compilation time of your program with the `-Wl,-stack,xxx` command line parameter.

> **Caution:** Don't skip this change (setting the stack size to at least 0x800 bytes), since the default stack size of PCOS programs (if not explicitly set by the PCOS linker) is less than 500 bytes, which is not sufficient for C programs. The runtime library needs more stack space, and if the stack overflows it will result in strange errors which are difficult to debug.

9.1.1.3 PCOS specific parts of the runtime library

Most of the PCOS specific parts of the runtime library are in `<srcdir>/src/newlib/libc/sys/z8kpcos` and `<srcdir>/src/newlib/libc/machine/z8k`. The remaining parts are conditional defines in `newlib`'s C code.

9.1.2 Building for COFF

The source code comes in the archive `z8kgcc-jan-19-2009.tar.bz2`. Extract the source code into a directory (`<srcdir>`). Then create somewhere else a "build" directory (`<builddir>`). `<instdir>` indicates the directory where you want the compiler to be installed, and `<archive location>` refers to the directory where the downloaded `z8kgcc-jan-19-2009.tar.bz2` file is located.

```
$ mkdir <srcdir>
$ cd <srcdir>
$ bzip2 -dc <archive location>/z8kgcc-jan-19-2009.tar.bz2 | tar -xf -
$ mkdir <builddir>
$ cd <builddir>
```

Then "configure" and build the toolchain:

```
$ <srcdir>/src/configure --prefix=<instdir> --target=z8k-coff
$ make
```

Due to a problem in the compiler, the compilation will abort with an error when compiling `md5.c` of libiberty. Compile this file with "`-O`" instead of "`-O2`":

```
$ cd z8k-coff/std/libiberty
```

Redo the last failing command (compiling `md5.c`), but replace "`-O2`" with "`-O`" in the compilation parameters.

```
$ <builddir>/gcc/xgcc ... -O ...
$ cd ../../..
$ make
```

After the compilation has finished, install the newly created programs:

```
$ make install
```

9.2 Debugger

The source code comes in the archive `z8kgdb-jan-19-2009.tar.bz2`. Extract the source code into a directory (*<srcdir>*). Then create somewhere else a "build" directory (*<builddir>*). *<instdir>* indicates the directory where you want the debugger to be installed (typically the same location where the C compiler was installed), and *<archive location>* refers to the directory where the downloaded `z8kgdb-jan-19-2009.tar.bz2` file is located.

```
$ mkdir <srcdir>
$ cd <srcdir>
$ bzip2 -dc <archive location>/z8kgdb-jan-19-2009.tar.bz2 | tar -xf -
$ mkdir <builddir>
$ cd <builddir>
```

Then "configure" and build the debugger:

```
$ <srcdir>/src/configure --prefix=<instdir> --target=z8k-pcos
$ make
```

Replace `--target=z8k-pcos` with `--target=z8k-coff` to build the COFF version instead of the PCOS version. Use a different build directory for each version or clean the build directory before you build the other version. After the compilation has finished, install the newly created programs:

```
$ make install
```

9.2.1 m20stub.sav

This chapter will be provided in a future revision of this document.

Appendix A Suggested Readings

A.1 PCOS User Guide

The PCOS user guide can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/doc/english/PCOS/M20_PCOS.pdf.
This is release 2.0 from March 1983.

A.2 ASSEMBLER Language User Guide

The original Olivetti assembler language manual can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/doc/english/PCOS_asm_refman/PCOS_asm_refman.pdf.
This is version 2.0 from March 1983, code 3987670 L(0).

A.3 Olivetti M20 Hardware Manual

A copy of Olivetti's M20 hardware manual can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/doc/english/hardware_manual/Olivetti_M20_Hardware_Manual.pdf.
This is the first edition from July 1983, code 4100630 W(0).

A.4 Z8000 Technical Manual

A copy of Zilog's Z8000 technical manual can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/doc/english/Z8000_tech_man/Z8000TechMan.pdf.

A.5 Z8000 Programmer's Guide

A copy of Zilog's Z8000 programmer's guide can be found at ftp://ftp.groessler.org/pub/chris/olivetti/m20/doc/english/Z8000_prog_guide/Z8000_prog_guide.pdf.
This is an "Application Note" from Juli 1981.

Appendix B Acknowledgements

I'd like to thank Davide Bucci for his proofreading and suggestions for improvements.

Thanks to Steve Chamberlain, who wrote the original support for Z8000 in `binutils`, `gcc`, `gdb`, and `newlib`.

Appendix C Revision of this document

The document's revision is shown on the back side of the cover page. Look for the `$Id$` line.

Appendix D GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index