APPENDIX D

COMPILER PASSES

This appendix describes each of the passes of the compiler: the **cpp80** preprocessor, the **cp180** parser, the **cp280** code generator and the **cp380** assembly language optimizer. The information contained in this appendix is of interest to those users who want to modify the default operation of the cross compiler by changing the prototype file that the c compiler driver uses to control the compilation process.

The cpp80 Preprocessor

cpp80 is the preprocessor used by the C compiler to expand #defines, #includes, and other directives signalled by a **#**, before actual compilation begins.

Command Line Options

cpp80 accepts the following options, each of which is described in detail below:

```
cpp80 -[d*^ err i* +lincl +map* o* +pas p? +std s?
+xdebug x] <files>
```

-d*^ where * has the form **name=def**, define **name** with the definition string **def** before reading the input. If **=def** is omitted, the definition is taken as 1. The **name** and **def** must be in the same argument, i.e., no blanks are permitted unless the argument is quoted. Up to ten definitions may be entered in this fashion.

-err pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

-i* change the prefix used with **#include <filename>** from the default (no prefix) to the string *. Multiple prefixes (including the null prefix) to be tried in order may be specified, separated by the character |.

+lincl force a listing of all include files to be included with the source listing, and make any diagnostic output indicate the actual **#include** file name and line number. The default is not to include a listing of #include files.

+map* map characters specified in strings according to values given in the map file. The map file should contain 256 bytes, each representing the value of the character set for the target machine. The default is no mapping.

-o* write the output to the file * and write error messages to STDOUT. Default is STDOUT for output and STDERR for error messages.

+pas assume Pascal-style {comments} in the source; i.e., treat anything between { and } as a comment and everything between /* and */ as source code.

-p? change the preprocessor control character from # to **?**.

+std enforce lexical elements in the input file to meet the lexical requirements given in the ANSI C draft standard.

-s? change the secondary preprocessor control character to **?**. By default, the secondary preprocessor control character is disabled.

+xdebug produce debugging information. This option must be specified to **cpp80, cp180** and **cp280** to produce meaningful information.

-x put out lexemes for input to the C compiler parser **cp180,** not lines of text.

cpp80 processes the named <files> (or STDIN if none are given) in the order specified, and the resulting text is written to STDOUT.

For more information on the preprocessor, see Chapter 2 of your C Language Specification for Microcontroller Environments.

Preprocessor Control Character

The presence of a secondary preprocessor control character permits two levels of parameterization. For instance, the invocation

cpp80 -p@

will expand **@define** and **@ifdef** conditionals, leaving all # commands intact; invoking **cpp80** with no arguments would expand only # commands.

By default (i.e. if **-err** is not specified), **cpp80** reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

Return Status

cpp80 returns success if it produces no error diagnostics, or if the **-err** option is specified.

Examples

The standard style for writing C programs is:

```
/*  name of program
 */
#include <wslxa.h>

#define MAXN    100

COUNT things[MAXN];
...
```

The use of uppercase-only identifiers is not required by **cpp80,** but is strongly recommended to distinguish parameters from normal program identifiers and keywords.

Special Usage Considerations

Floating constants longer than 38 digits may compile incorrectly on some host systems.

The cp180 Parser

cp180 is the parsing pass of the C compiler. It accepts a sequential file of lexemes from the preprocessor **cpp80** and outputs a sequential file of flow graphs and parse trees suitable for input to the code generator **cp280.** The operation of **cp180** is largely independent of any target machine.

Command Line Options

cp180 accepts the following options, each of which is described in detail below. Because **cp180** is target architecture independent, some of the options it accepts may not be appropriate for use on your system.

cp180 -[b# +bf c +dead dl err sp m n# +nwd o* sr +std +strict strict u +xdebug] <file>

-b# enforce storage boundaries according to #, which is reduced modulo 4. A bound of 0 leaves no holes in structures or auto allocations; a bound of 1 (default) requires short, int and longer data to begin on an even bound; a bound of 2 is the same as 1, except that 4-8 byte data are forced to a multiple of four byte boundary; a bound of 3 is the same as 2, except that 8 byte data (doubles) are forced to a multiple of eight byte boundary.

+bf bitfields are of size char and int. By default, bitfields are packed in integers (16 bits). If this flag is set, bitfields are packed in chars (8 bits) or int (16 bits) according to their declaration. It is possible to mix char bitfields and int bitfields in the same file.

-c ignore case distinctions in testing external identifiers for equality, and map all names to lowercase on output. By default, case distinctions matter.

+dead flag unused variables with an error message. The default is not to complain about unused variables.

-dl generate line number information. This option must be specified in combination with the **-dl#** option to **cp280** to produce meaningful information.

-err pass error diagnostics from input file to output file, add error diagnostics generated to the output

file, and report success. Only fatal errors, such as not being able to write an output file due to lack of space, will be reported and cause the return status to change to failure.

-sp treat all floating point numbers as float and not double, even if they are declared as double. All calculations will be made on 32 bits instead of 64 bits. Space reservations will be made on a 32 bit basis, as well as argument passing.

-m treat each struct/union as a separate name space, and require **x.m** to have a structure **x** with **m** as one of its members.

-n# ignore characters after the first **#** in testing external identifiers for equality. The default is 31. The maximum is 127.

+nwd do not widen arguments. The standard behavior of the compiler is to widen integer arguments smaller than int to int size and to widen float arguments to double. If this flag is set, these promotions are not done. The code thus obtained should be more compact if char and floats are heavily used.

-o* write output to the file ***** and write error messages to STDOUT. The default is STDOUT for output and STDERR for error messages.

-sr make strings read only (i.e. put them in the text section). The default is to make strings both readable and writeable.

+std force the input to conform to the ANSI C draft standard.

+strict enforce much stronger type checking, as described below.

-strict allow more lenient (weaker) type checking, as described below. By default, the type checking done by **cp180** is between the two extremes **+strict** and **-strict**.

-u take a string "string" to be of type array of unsigned char, not array of char.

+xdebug generate debugging information for use by the **cxdb** debugger or some other debugger or in-circuit emulator. The default is to generate no debugging information.

If <file> is present, it is used as the input file instead of the default STDIN.

By default (i.e. if **-err** is not specified), **cp180** reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

If the **-strict** option is present, no diagnostic is generated when:

1) an integer is assigned a pointer value.

2) a pointer is assigned an integer constant value other than NULL (0).

3) two different pointer types are checked for assignment compatibility.

4) the types for the result on a conditional operator do not match (i.e. **x ? a : b** where **a** and **b** have different types).

If the **+strict** option is present, a diagnostic is generated:

1) if an argument to a function definition does not have an explicit type associated with it by the time the opening **{** for the function body is encountered (i.e. **f(abc){}**). Without strict type checking, the assumed type for an untyped argument is int.

2) for any implicit narrowing assignment. This affects initialization, function call arguments when a function prototype exists, and implicit assignment statements. A cast may be used to disable the checking on a case by case basis. Without strict type checking, a narrowing assignment will truncate without complaint.

3) if the return value of a function is used when the return type of the function was not previously declared. Without strict type checking, the assumed type for a function is int.

4) if an argument to a function is not the same type as the type specified in the prototype declaration.

Return Status

cp180 returns success if it produces no diagnostics, or it the **-err** option is specified and no fatal errors occur.

Examples

cp180 is usually invoked between **cpp80** and the code generator, as in:

```
cpp80 -x -o temp1 file.c
cp180 -o temp2 temp1
cp280 -o file.s temp2
```

Special Usage Considerations

cp180's processing of semicolons can be difficult to predict.

The cp280 Code Generator

cp280 is the code generating pass of the C compiler. It accepts a sequential file of flow graphs and parse trees from **cp180** and outputs a sequential file of intermediate language statements.

As much as possible, the compiler generates freestanding code, but, for those operations which cannot be done compactly, it generates inline calls to a set of machine-dependent runtime library routines.

cp280 accepts the following options, each of which is described in detail below:

cp280 -[bss ck dl# err h64180 i +list o* sp v x#] <file>

-bss inhibit generating code into the bss section.

-ck enable stack overflow checking.

-dl# produce line number information. This option must be used in combination with the **-dl** option to **cp180**. # must be either '1' or '2'. Line number information can be produced in two ways: 1) function name and line number is obtained by specifying **-dl1**; 2) file

name and line number is obtained by specifying **-dl2**. All information is coded in symbols that are in the debug symbol table.

-err pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

-h64180 Generate code for the **HD64180.** By default the compiler generates code for the **Z80.**

-i Use IEEE standard representation for floats and doubles. **THE COMPILER REPRESENTS floats and doubles AS PER THE PDP-11** REPRESENTATION AS DOCUMENTED IN THIS MANUAL. ALL LIBRARIES PROVIDED WITH THIS COMPILER ASSUME THAT CODE IS OUTPUT FOR THE PDP-11 REPRESENTATION.

+list generate line number directives interspersed with assembly language code to be used by the **lm** utility to generate a listing file. The default is to turn listings off.

-o* write the output to the file * and write error messages to STDOUT. The default is STDOUT for output and STDERR for error messages.

-rev reverse the ordering of bits in bitfields. The compiler would normally number bits in bitfields, starting with the least significant bit. If this flag is set, bits are numbered starting with the most significant bit.

-sp enable single precision for floating point computation. Automatic widening of float to double is suppressed, so float expressions are carried in single precision, and double expressions are carried in double.

-v When this option is set, each function name is send to STDERR when cp280 start processing it.

-x# map the three virtual sections, for Functions (04), Literals (02), and Variables (01), to the two physical sections, Code (bit is a one) and Data (bit is a zero). Thus, -x4 is for separate I/D space, **-x6** is for ROM/RAM code, and -x7 is for compiling tables into ROM. The default is 6.

If <file> is present, it is used as the input file instead of the default STDIN. On many systems, (other than

IDRIS/UNIX), <file> is mandatory, because STDIN is interpreted as a text file, and therefore becomes corrupted.

Return Status

cp280 returns success if it produces no diagnostics, or if the **-err** option is specified and no fatal error occurs.

Examples

cp280 usually follows **cpp80** and **cp180**, as follows:

```
cpp80 -o temp1 -x file.c
cp180 -o temp2 temp1
cp280 -o file.s temp2
```

The cp380 Assembly Language Optimizer

cp380 is the code optimizing pass of the C compiler. It reads source files of intermediate language, as generated by the **cp280** code generator, and writes assembly language statements. **cp380** is a peephole optimizer; it works by passing a moving window over the input lines, checking lines in the window for specific patterns. If the patterns are present, **cp380** replaces the lines where the patterns occur with an optimized line or set of lines. It repeatedly checks replaced patterns for further optimizations until no more are possible; then it moves the window over the file. It deals with redundant load/store operations, constants, stack handling, and other operations.

Command Line Options

cp380 accepts the following options, each of which is described in detail below:

cp380 -[e o* r# v] <file>

-e use relative jumps.

-o* write the output to the file * and write error messages to STDOUT. The default is STDOUT for output and STDERR for error messages.

Compiler Passes

-r# set the size of the window to # lines. The default is 40.

-v write a log to STDERR. This displays the number of removed instructions followed by the number of modified instructions.

If <file> is present, it is used as the input file instead of the default STDIN.

Return Status

cp380 returns success if it produces no diagnostics.

Examples

cp380 is usually invoked after **cp280** as follows:

```
cpp80 -o temp1 -x file.c
cp180 -o temp2 temp1
cp280 -o temp3 temp2
cp380 -o file.s temp3
```