Aztec C II v 1.06 Release Document

Release 1.06 of the Aztec C development system for CP/M incorporates a number of new features in all parts of the package. These improvements are treated topically below:

The Aztec C II Compiler v 1.06b

The compiler supports the entire C language as described by Kernighan and Ritchie, with the exception of the bit field data type. Several features were added in this release:

1. #if

   This directive completes the capabilities of the preprocessor.

2. \

   The sequence, \-carriage return, can be used to continue any line. Thus, a logical line can be infinitely long. There can be no space between the backslash and the newline.

3. #include <filename>

   The compiler will accept angled brackets, < and >, as delimiters in a **#include** statement. They are handled in the same way as double quotes. (See "New Compiler Options", **-I**).

4. **unsigned**

   The compiler recognizes **unsigned char**, and **unsigned long**, as well as **unsigned int**. As before, **unsigned** defaults to unsigned int. With the Aztec compilers on 8-bit machines, **char** is unsigned; on 16-bit machines, **char** is signed by default.

5. **short int**

   Previously, the compiler did not allow this declaration. **short** and **int** are both two bytes, while **long int** is four bytes.

New Compiler Options:

The compiler has several new options. They are listed here and fully described in the manual:

1. -F In-line function entry code.
2. -I Specify areas to be searched for included files.
3. -L Specify size of local symbol table.
4. -P Sends error messages to the printer.
5. -Q Convert automatics to statics.
6. -R Produce code for RMAC by DRI.
7. -U Convert globals to externs.

Error Checking by the Compiler:

It is important to note that source code which previously compiled without error will now generate errors. This is due to some mild type checking.

1. Variables can no longer be redeclared.

   The compiler will generate an error in the obvious case:

```
int i;
double i;
```

   Nor can functions be redeclared. Hence, the following will produce an error:

```
main()
{
     double func();
     int i = func2();
}

func()
{
}
double func2()
{
}
```

   func has been declared double in main but defined as int. func2 has been implicitly declared int in main and defined double below.

2. structures cannot be passed as arguments

An attempt to do so will generate an error. However, note that an argument can be declared as a structure without producing a compile-time error:

```
func(s)
structure tag s;
{
        . . .
```

References to members of the structure are simply offsets from the beginning of s on the stack. It is not advisable to make use of this fact, since this may change.

The Linker and Libraries in v 1.06b

The major change in the linker is that it now distinguishes between initialized and uninitialized data. These are placed in distinct data areas. The linker has new options accordingly for specifying the address of each segment.

The standard run time library is now called **c.lib**; the library of math functions is called **m.lib**.

Included in the standard package is a special library, called **r.lib**. Linking with it will set up your program as an overlay of **r.com**. This reduces link time since the linker does not have to search **c.lib** for library modules. This feature is described in the manual under the section entitled "Fast Linker".

Also in the standard package is a library called **tiny.lib**. It can be used to decrease the amount of overhead in the executable program. The use and limitations of this library are described in the manual.

Fixed Bugs

1. A bug in the comparison of longs to a constant was fixed.

2. The library function, **fopen**, in append mode will now position the file according to the description in the manual.

3. The compiler will work with files which are an exact multiple of 128 bytes in length.

4. The software will report an error when the disk is full.

5. The preprocessor has been extensively rewritten. The infrequent bugs concerning white space have been fixed.

6. **open** frees the file control block of a file on an open failure.

7. **scanf** works according to the description in the library section of the manual. Several bugs were fixed.

8. The logical negation of a constant now works. For example, !1 is zero.

9. **%%** in a format string is treated as the per cent character.

10. The initializer of an automatic or register variable can be an arbitrary expression.

11. **stdout** to a disk file or the printer has a 1024 byte buffer.

12. The special caveats for overlays which would cause an "Org out of range" message from the linker no longer exist.

13. The extract option (-x) for Libutil has been fixed.

14. The **format** function supports the **g** option; this is used by **printf** and related formatting functions.

15. Under the -M option, the caveats for using M80 no longer exist. This includes: specifying an option to M80 to ensure that statics are initialized to zero; including libc.h in every source module; specifying the .8080 statement to M80.

16. The compiler allows macros to be redefined. The latest definition is current.

Addenda

Using the Tiny Library

There are several special criteria which must be satisfied when using the Tiny library. These are described in the manual. In addition to these, the user should be aware of the following:

> In order to use **gets** and **puts**, **getchar** or **putchar** must be pulled in from the Tiny library, respectively. This will prevent the linker from pulling in the **getchar** and **putchar** from **c.lib**, which would cause errors. The alternative solution is to extract **gets** and **puts** from **c.lib** and place them at the beginning of the Tiny library, or included them explicitly in the linkage.

The Fast Linker

The fast linker will automatically include the run time support provided by **c.lib**. Since floating point calculations are not supported by **c.lib**, it is necessary to link in **m.lib** when using floating point. In addition, it is necessary to include the module, **puterr.o** explicitly in the linkage. This file is provided on the distribution disk. It can also be extracted from the standard library.

SIDSYM

The documentation for the program, **SIDSYM**, was omitted from the manual section entitled, "Software Extensions: utility programs." This is included here, and can be inserted into the manual in the appropriate place.

SIDSYM is a program which converts a symbol table, created by the linker with the -T option, into a format which can be input to Digital Research's symbolic debuggers, SID and ZSID.