Aztec C II v 1.05g Release Document

Release 1.05g of Aztec C II incorporates a number of new features, some bug fixes, and performance improvements.

The major feature additions are:

1. Scientific math functions
2. Overlay support
3. I/O redirection
4. Expanded device support
5. SCANF
6. New object library utility (LIBUTIL)
7. Relative byte support (lseek) for unbuffered I/O
8. New user's manual

The linkage editor, LN, was rewritten; it's now 40 % faster.

The following bugs were fixed:

1. The were several problems with long to float conversion.

2. There were problems with evaluation of long and double conditional expressions.

3. Shifts specifying 0 bits didn't work.

4. Division of a floating point number by itself didn't equal one.

5. There were some problems using object libraries created by Microsoft's M80 and LIB programs.

6. Error in the 'strcat' function.

7. The function 'fopen' didn't position the file correctly when opening with the 'append' option.

8. Subtracting a constant from a pointer didn't work.

Special notes:

1. OVLN.COM

The chapter on overlays says in one place that root and overlay segments are created using the program LN, and in another place that they are created using OVLN. Actually, LN is used.

2. Overlay usage

There are several caveats if you are using overlays. First, the 'settop' function must be called with an argument whose value is set equal to the size of the largest overlay segment, or the length of the longest thread if overlays are nested. This call must be initiated at the beginning of the main routine before calling any other routines. The size of an overlay is displayed on the console, in hex, when it's linked. For example, if your program uses 3 overlays, and the linker says their sizes are 125ah, 236h, and 837h, then it should make the following call: settop(0x125a). The parameter to settop could be larger, if desired.

Second, If buffered I/O is used in an overlay segment at least one buffered I/O call must exist in the root. This can be accomplished by calling the 'printf' routine in the root. Note that it is not necessary that the call to 'printf' be executed, only that it be present. This will insure that the link editor will include the buffered I/O tables in the root segment.

The section on overlays in the manual incorrectly states that an overlay function can have any name. The overlay file can have any name (but it's extent must be 'ovr'). Also, the main module in an overlay must be named 'ovmain'. The example on pages XI.4 and XI.5 should describe the two overlays as follows:

Here is ovly1:

```
    ovmain(a)
    char *a;
    {
            printf("in ovly1. %s\n",a);
            return 1;
    }
```

Here is ovly2:

```
    ovmain(a)
    char *a;
    {
            printf("in ovly2. %s\n",a);
            return 2;
    }
```

Overlays can be nested, contrary to what the overlay section of the manual states. If one overlay is to call another, the command line to the linkage editor for the first overlay must specify the dash r option, "-r". This will cause LN to generate a '.rsm' file for the first overlay. This '.rsm' file must then be included in the command line to the nested overlay. Also, each overlay must include the module 'ovbgn.o'. Overlays can be nested to any level as long as the "-r" option is included on each link edit and the ".rsm" file of the calling overlay segment is included in the LN parameter list for the link edit of the called overlay segment. The "-r" option need not be specified for the last segment of any one overlay path. Extreme caution should be used in using an overlay segment in more than one path. Although this is possible to do, there is an enormous potential for error.

For example, here are three nested segments: a root segment, root, and two overlay segments, ovly1 and ovly2. root calls ovly1; ovly1 calls ovly2; ovly2 just returns.

Here is the root:

```
main()

{
 settop(0x13aa);

 ovloader("ovly1");

 return;

}
```

Here is ovly1:

```
ovly1() { ovloader("ovly2"); return;}
```

Here is ovly2:

```
ovly2() { return;}
```

The following commands will link these three segments:

```
ln -r root.o ovloader.o libc.lib
ln -r ovly1.o ovbgn.o root.rsm libc.lib
ln    ovly2.o ovbgn.o ovly1.rsm libc.lib
```

This example illustrates only one overlay path. In actual practice, there would usually be more than one path.

3. SIDSYM

The manual does not describe the program SIDSYM. This program converts a symbol table which has been created by the linkage editor, LN, into a format which can be input to Digital Research's symbolic debuggers, SID and ZSID. (The '-t' option to LN causes LN to generate a symbol table). The format for starting SIDSYM is :

```
SIDSYM infile outfile
```

where infile is the name of the file containing the symbol table generated by LN and outfile is the name of the file which is to contain the SID- and ZSID-readable symbol table. infile and outfile can specify the same file.

Release 1.05 Files:

cii.com	compiler generating 8080 code.
ln.com	Manx linkage editor
as.com	Manx relocating assembler
libutil.com	Manx object file librarian
sidsym.com	sid symbol table formatter
czii.com	'c' compiler generating z80 code.
libc.lib	8080 run-time object library. (Manx format).
math.lib	8080 scientific function object library. (Manx format).
softlibc.rel	8080 run-time object library (Microsoft format).
softmath.rel	8080 scientific function object library (Microsoft format).
libc.h	'c' program header file. Should be included in most 'c' programs.
errno.h	'c' program header file. Defines the values set in the global variable 'errno' by the floating point and math functions.
stdio.h	'c' program header file. To be used by programs requiring UNIX compatibility.
io.h	'c' program header file. Used by the run-time library i/o functions.
math.h	'c' program header file that declares the type of the math functions.
object.h	'c' program header file. Defines the Manx object file format.
fcntl.h	'c' program header file. Defines the option parameter for the 'open' function.
makelibc.sub	submit file for making libc.lib.
makemath.sub	submit file for making math.lib.
softlibc.sub	submit file for making softlibc.rel.
softmath.sub	submit file for making softmath.rel.
makezlib.sub	submit file for making z80libc.lib (a z80 version of libc.lib).
*.asm	assembler source for the run-time library functions.
*.c	'c' source for the run-time library functions.
libutil.c	'c' source for libutil.
ovloader.c	'c' source for the overlay support function, ovloader.
ovbgn.asm	assembler source for the overlay support function, ovbgn.
*.ucl	'c' source for user-supplied functions.
*.ual	assembler source for user-supplied functions.

Installation procedures:

1. To install an 8080 system, the following files must be copied from the distribution disk:

   cii.com, as.com, ln.com, libc.lib, math.lib, libc.h

2. To install a z80 system, the following files must be copied:

   czii.com, as.com, ln.com, libc.h

   Also, you must create a z80 run-time library. If you have a large-capacity disk drive, this can be done by executing the submit file makezlib.sub. This creates the file z80libc.lib. Otherwise, you will have to create the run-time library in several steps, with each of the initial steps compiling and assembling several run-time functions onto separate disks. Each of the last steps will then transfer, using the program 'libutil', the object files from one disk to the z80 library file. For example, if your system has two drives, and you determine that three disks are required to compile and assemble the run-time library routines, you could proceed as follows:

   a. create a disk with czii.com and as.com on it. Also, the disk should contain three submit files, each of which compiles and assembles the files on one of the run-time library source and object disks.

   b. copy the source files for the run-time library functions onto the three disks.

   c. place the disk containing the submit files, czii.com, as.com, and submit.com into the 'a:' drive. For each of the source and object disks, place it in the 'b:' drive and execute its submit file.

   d. create a disk containing libutil.com, submit.com, and three submit files. Each submit file will copy object files from one of the source and object disks to the z80 run-time library file. This library file should be on the disk containing libutil, and the submit files.

   e. place the disk containing the libutil and submit files in the 'a:' drive. For each of the source-and-object files, place it in the 'b:' drive, and execute its submit file.

As you can see, creating a z80 system can be a lot of work. If you are new to Aztec C, you might consider installing an 8080 system first and becoming familiar with it before installing a z80 system. An 8080 system will run on a z80 as well as an 8080.

3. To install an 8080 Microsoft system, the following files must be copied:

```
          cii.com, softlibc.rel, softmath.rel, libc.h
```

4. To install a z80 Microsoft system, the following files must be copied:

```
          czii.com libc.h
```

Also, a run-time library must be created. If you have a large capacity disk drive, this can be done by modifying and executing the submit file softlibc.sub. The modifications are: use czii instead of cii, assemble suppz80.asm instead of supp8080.asm, and replace supp8080 by suppz80.asm.

If you don't have a large capacity drive, you must create the run-time library in several steps. Use the procedure described in section 2 above as a model.