APPENDIX B

COMPILER PHASE COMMAND LINES

This appendix describes the command lines and options for the individual compiler phases. Each phase of the compiler may be executed separately. The following information describes the options available to each phase.

cc1 & cc2 (C executives):

```
     cc [options] file {file} [options]

     Recognized file suffixes:
          .c         C source file
          .a         Assembly language source file
          .r         Relocatable module format file

     Recognized options:  (UPPER and lower case is equiv.)
          -a         Suppress assembly.  Leave output in ".a" file.
          -e=n       Edition number (n) is supplied to c.prep for
                     inclusion in module psect and/or to c.link for
                     inclusion as the edition number of the linked
                     module.
          -o         Inhibits assembler code optimizer pass.
          -p         Invoke compiler function profiler.
          -r         Suppress link step.  Leave output in ".r" file.
          -m=<size>  Size in pages (in kbytes if followed by a K) of
                     additional memory the linker should allocate to
                     object module.
          -l=<path>  Library file for linker to search before the
                      standard library.
          -f=<path>   Overrride other output naming.  Module name (in
                      object module) is the last name in the pathlist.
                      -f is not allowed with -a or -r.
          -c         Output comments in assembly language code.
          -s         Suppress generation of stack-checking code.
          -d<NAME>   Is equiv to #define <NAME> 1 in the
                     preprocessor.  -d<NAME>=<STRING>  is equivalent to
                     #define <NAME> <STRING>.

           -n=<name> output module name.  <name> is used to override
                     the -f default output name.

     CC1 only:
          -x         Create, but do not execute c.com command file.

     CC2 only:
          -q         Quiet mode.  Suppress echo of file names.
```

c.prep (C macro preprocessor)

```
c.prep [options] <path>
```

<path> is read as input. C.prep causes c.comp to generate a psect directive with the last element of the pathlist and _c as the psect name. If <path> is /d0/myprog.c, the psect name is myprog_c. Output is always to stdout.

Recognized options:

```
-l        Cause c.comp to copy source lines to assembly
          output as comments.
-E=<n>
-e=<n>    Use <n> as psect edition number.
-D<NAME>  Same as described above for cc1/cc2.
```

c.comp (One-pass compiler)

```
c.comp [options] [<file>] [options]
```

If <file> is not present, c.comp will read stdin. Input text need not be c.prep output, but no preprocessor directives are recognized (#include, #define, macros etc.). Output assembly code is normally to stdout. Error message output is always written to stdout.

Recognized options:

```
-s        Suppress generation of stack checking code.
-p        Generate profile code.
-o=<path> Write assembly output to <path>.
```

c.pass1 (Pass One of Two-pass Compiler)
c.pass2 (Pass Two of Two-pass Compiler)

```
c.pass1 [options] [<file>] [options]
c.pass2 [options] [<file>] [options]
```

Command line and options are the same as c.comp. If the options given to c.pass1 are not given to c.pass2 also, c.pass2 will not be able to read the c.pass1 output. Both c.pass1 and c.pass2 read stdin and write stdout normally.

c.opt (Assembly code optimizer)

```
c.opt [<inpath>] [<outpath>]
```

C.opt reads stdin and writes stdout. <inpath> must be present if <outpath> is given. Since c.opt rearranges and changes code, comments and assembler directives may be rearranged.

c.asm (Assembler)

```
c.asm <file> [options]
```

C.asm reads <file> as assemble language input. Errors are written to stderr. Options are turned on or off by the inclusion of the option character preceeded by a '-'.

Recognized options:

```
    -o=<path> Write relocatable output to path.  Must be a
              disk file.
    -l        Write listing to stdout. (default off)
    -c        List conditional assembly lines. (default on)
    -f        Formfeed for top of form. (default off)
    -g        List all code bytes generated. (default off)
    -x        Suppress macro expansion listing. (default on)
    -e        Print errors. (default on)
    -s        Print symbol table. (default off).
    -dn       Set lines per page to n. (default 66).
    -wn       Set line width to n. (default 80).
```

c.link (Linker)

```
c.link [options] <mainline> [<sub1> {<subn>} ] [options]
```

C.link turns c.asm output into executable form. All input files must contain relocatable object format (ROF) files. <mainline> specifies the base module from which to resolve external references. A mainline module is indicated by setting the type/lang value in the psect directive non-zero. No other ROF can contain a mainline psect. The mainline and all subroutine files will appear in the final linked object module whether actually referenced or not.

For the C Compiler, cstart.r is the mainline module. It is the mainline module's job to perform the initialization of data and the relocation of any data-text and data-data references within the initialized data using the information in the object module supplied by c.link.

(c.link continued)

Recognized options:

-o=<path>	Linker object output file must be a disk file. The last element in <path> is used as the module name unless overridden by -n.
-n=<name>	Use <name> as output module name.
-l=<path>	Use <path> as library file. A library file consists of one or more merged assembly ROF files. Each psect in the file is checked to see if it resolves any unresolved references. If so, the module is included in the final output module, otherwise it is skipped. No mainline psects are allowed in a library file. Library files are searched in the order given on the command line.
-E=<n>	
-e=<n>	<n> is used for the edition number in the final output module. 1 is used if -e is not present.
-M=<size>	<size> indicates the number of pages (kbytes if size is followed by a K) of additional memory, c.link will allocate to the data area of the final object module. If no additional memory is given, c.link adds up the total data stack requirements found in the psect of the modules in the input modules.
-m	Prints linkage map indicating base addresses of the psects in the final object module.
-s	Prints final addresses assigned to symbols in the final object module.
-b=<ept>	Link C functions to be callable by BASIC09. <ept> is the name of the function to be transferred to when BASIC09 executes the RUN command.
-t	Allows static data to appear in a BASIC09 callable module. It is assumed the C function called and the calling BASIC09 program have provided a sufficiently large static storage data area pointed to by the Y register.

USING AND LINKING TO USER DEFINED LIBRARIES

A library consists of a group of "C" procedures or functions that have been separately compiled into Relocatable Object Files (ROF) and subsequently merged into one library file.

If, hypothetically, you had created a set of higher level mathematic functions, that you wanted to convert into a "C" library. First you would separately compile each one using the -R option. Then you would merge them all into one large library file. If you need to scan the library file for available functions you can use the example program "RDUMP.C" to inspect any "C" library file.

For example:

```
          OS9:CC1 SIN.C COS.C TAN.C ARCOS.C -R
          OS9:CC1 ARCSIN.C ARCTAN.C EXP.C LOG.C -R
          OS9:CC1 NLOG.C SQRT.C SQR.C CUBE.C -R

          Then you would:
          OS9:MERGE SIN.R COS.R TAM.R ARCOS.R >TEMP1
          OS9:MERGE ARCSIN.R ARCTAN.R EXP.R LOG.R >TEMP2
          OS9:MERGE NLOG.R SQRT.R SQR.R CUBE.R >TEMP3
          OS9:MERGE TEMP1 TEMP2 TEMP3 >TRIG.L
```

Then to use the library simply use the -l=<pathlist> option in your command line when you compile your program.

When the linker is executed the pathlist specified will be searched to resolve any references made to the functions within the library. The linker searches all specified libraries in the order specified before searching the standard library. The linker will resolve all references on a first found basis. This means that the linker will use the first procedure or function whose name matches a reference to that name and will ignore any additional functions found that have the same name.

Procedures or functions within a library that use other functions within the same library should always appear first. For example, in the above example if the "ARCSIN" routine used the "SIN" routine, the "SIN" routine should be merged into the library file after the "ARCSIN". Another way of putting this is that all references to other procedures within a library should be forward references.