thames: An ISIS-II Emulator

(Original - John Elliott, 17 November 2012, enhancements by Mark Ogden)

thames emulates enough of the ISIS-II environment to be able to run may of the ISIS II build tools as used in the CP/M 3 build process:

- asm80: ISIS-II 8080/8085 macro assembler, v4.1

- plm80: ISIS-II PL/M-80 Compiler v4.0

- link: ISIS-II object linker v3.0

- locate: ISIS-II object locater v3.0

- objhex: Converts an ISIS-II object file to hex format (v2.2)

See Enhancements for information on more recent changes.

Recent changes - 10-Oct-2023

- The file handling has been reworked and and the emulator now correctly reports an error when trying to open an already open file. This caused problems with asm48 v2.0.

- Support for device mapping has been added back in.

  Note the applications asm80 v4.1, plm80 v4.0, link v3.0, locate v3.0, ixref v1.4 and binobj, hexobj, objhex, have now all been ported to modern C and work with long file names under windows and Linux and would be the recommended tools to use rather than emulation. Additional enhancements

  1. All the command lines can now be very long lines. Although & is supported is shouldn't be needed. Additionally applications now allow empty command line, taking the command line from stdin. If stdin is redirected from a file, then implicit & at the end of each line is assumed, if not explicitly provided. An end of file terminates this.

  2. asm80 the label limit is now 31 characters and like PLM80, $ in a name is ignored.

  3. The tools now allow an _ to be included in a name. It is ignored if present in numbers

  4. Most commands now no longer require comma separators, making it much easier to work with make.

  5. lib now supports a non interactive option including a new single create operation adding a new set of files.

  6. Intermediate files are no longer needed, so parallel builds are straightforward

  7. All I/O uses stdio buffering, removing significant chunks of original code.

  8. In listings line endings use the native os format, the I/O does however handle /r on input.

  9. ixref supports file name globbing. Additional in co-ordination with plm80, the redundant label element which plm80 v4.0 no longer used, is used to allow longer file names in the cross reference.

  10. plm80 has additional functionality added similar to thames. Including options to ignore warnings and generate dependency files.

  11. asm80, plm80 now support the current date in listings.

Installing (Updated)

Installation has been modified to use cmake or the Visual Studio solution file thames.sln. For cmake the following commands are needed

```
mkdir {your chosen build directory}
cd {your choosen build directory}
cmake ..
cmake --build  .
```

In use

Thames now supports a level of auto mapping from windows/linux directories to ISIS file devices, see enhancements. You can however control the mapping explicitly, by setting up environment variables to ISIS devices to Unix/Windows paths.

The environment variable names are of the form ISIS_XX, where XX is the device name. When an application uses a device file device the value of ISIS_XX is used to map to a suitable directory.

For example, in a Bourne-style shell:

```
ISIS_F0=/home/me/isis
ISIS_F1=/home/me/isis/plm80
ISIS_F2=/home/me/isis/asm80
ISIS_F3=/home/me/isis/utils
export ISIS_F0 ISIS_F1 ISIS_F2 ISIS_F3
```

Or in a csh-style shell:

```
setenv ISIS_F0 /home/me/isis
setenv ISIS_F1 /home/me/isis/plm80
setenv ISIS_F2 /home/me/isis/asm80
setenv ISIS_F3 /home/me/isis/utils
```

Or for Windows

```
ISIS_F0=/users/me/isis
ISIS_F1=/users/me/isis/plm80
ISIS_F2=/users/me/isis/asm80
ISIS_F3=/users/me/isis/utils
```

Note the windows build supports either \ or / as directory separators and also leading disk names e.g. E:

It's also possible to set up character devices this way. For example, if the program you want to run needs to use the printer device :LP:, then you can set up a file to receive printer output:

```
ISIS_LP=/home/me/isis/lp.txt export ISIS_LP

setenv ISIS_LP /home/me/isis/lp.txt
```

Once the variables are set up, you should be able to run an ISIS
program like this:

thames *isis-command*

For example:

```
thames :F3:locate put.mod "code(0100H)" "stacksize(100)"
```

Notes:

- Arguments containing brackets have to be escaped to stop the shell
  trying to parse them.

- Filenames without a :Fn: drive specifier are assumed to be on drive
  :F0:.

- thames maps all ISIS filenames to lower case and since most applications validate for the 6.3
  alphanumeric format, you should ensure that they are also in 6.3 format i.e.
  no more than six alphanumeric characters, followed optionally by a dot and up to
  three further alphanumeric characters.

- Environment variable values are not modified and can support the native files system
  characters

This sequence of commands should build PUT.COM from CP/M 3:

```
thames :F2:asm80 putf.asm debug
thames :F1:plm80 put.plm "pagewidth(100)" debug optimize
thames :F3:link mcd80a.obj,put.obj,parse.obj,putf.obj,:F3:plm80.lib to put.mod
thames :F3:locate put.mod "code(0100H)" "stacksize(100)"
thames :F3:objhex put to put.hex
```

Implementation notes

Some areas of functionality are rarely used in practice and have had less testing.
These include the system calls CONSOLE and WHOCON, and the ability to open files other than
the console in line mode.

Debugging

Set the ISIS_TRACE environment variable to get debug logs. It should be
set to an integer between 0 and 4:

0. No debug messages

1. Echoes command line, and reports on errors parsing executable
   files.

2. Logs all ISIS-II calls.

3. As 2, but also displays contents of buffers loaded/saved for
   READ, WRITE and EXEC calls.

4. All of the above, and also traces instruction execution.

Acknowledgements

- The Z80 emulation engine was written by Ian Collier. Other than the BIOS intercept calls, the instruction set has been reduced to support the 8080/8085 only. This avoids the occasional different behaviour of Z80 instructions

- The ISX documentation on the P112 pages, and the documentation of ISIS internals at bitsavers.org, were both invaluable to me in the course of writing thames.

Enhancements

A number of enhancements have been made to the original thames emulator mainly to better support its use in an automated build process. These enhancements are documented below.

Additionally other programs known to work now include

- asm80 versions 3.0, 3.1 & 4.1

- asm48 versions 2.0, 3.0, 4.0, 4.2

- plm80 versions 3.0, 3.1 - note to use XREF on these versions, a dummy ISIS.LAB file is needed.

- lib v2.1

- Fort80 v2.1

- Pasc80 v2.2

- Basic v1.0, v1.1, vx021 f2 - note limitation on kbd hit which is sensed but needs cr to continue

- IXREF versions 1.1, 1.2, 1.3 - note using wildcards requires and ISIS.DIR file to be created.

Emulator changes

A number of ISIS applications e.g. Basic, did not work as the underlying emulation was for a z80. The emulator has been modified to support the 8080, allowing more applications to run. The original emulation can be restored by uncommenting the #define Z80 line in thames.h

The tstate counting was removed as it wasn't used and there is a partial implementation of CSTS (bios 7) using kbhit, currently getchar requires waits for a line to be input before returning. Not a major problem but be aware.

Several minor bug fixes e.g. closing of :CO: and :CI:.

Enhanced command line

The latest thames command line syntax is

```
thames [-v|-V] | [-h] [-m] [-i] [-o] [-u] [-MF file] [-ME ext] [-T] isisprog
isisargs...
```

The options are

-v -- show basic version info

-V -- show additional git related info

-h -- show usage

-m -- command line pre-processing

This option pre-processes the isisprog and isisargs as follows

- The current directory is mapped to :F0: if not defined in the environment

- Any unix/windows file names present on the command line have the directory part automatically mapped to an isis drive. Environment defined mappings are honoured, but if no appropriate mapping is found, a new one is dynamically assigned.

- After the names are mapped as above, if the line is >120 characters it is split, possibly in to multiple lines, with an & CR LF sequence separating each line.
  This is the common approach taken in many ISIS tools to support long lists of arguments. Note although this is done for all command lines, asm80 in particular does not handle this, but requires the options to be put into an included file.

- The automatic line split can be overridden as follows:

  - A single & forces a line split and is replaced by & CR LF and is useful for the LIB tool which requires the last file name in an ADD operation to be on the same line as the TO target.lib options.

  - A double & also forces a line split but is replaced by CR LF and is again useful for the LIB tool to support multiple commands.

- The pre-processed command is echoed to stdout

Examples:

With ISIS_F0=./ and ISIS_F3=src/ defined in the environment, the command

```
thames -m ../../plm80v31/link
"obj/globlb.obj,obj/startb.obj,obj/pcktok.obj,obj/asm1n.obj,obj/asm2n.obj,obj/asm
4b.obj,obj/rdsrc.obj,obj/asm3b.obj,obj/asm5n.obj,obj/asm6n.obj,obj/cntrln.obj,obj
/emitn.obj,obj/listn.obj,obj/initb.obj,../../plm80v31/system.lib,../../plm80v31/p
lm80.lib,obj/keyn.obj" to obj/asm805.rel map "print(lst/asm805.lnk)"
```

is pre-processed to

```
:F1:link
:F2:globlb.obj,:F2:startb.obj,:F2:pcktok.obj,:F2:asm1n.obj,:F2:asm2n.obj,:F2:asm4
b.obj,:F2:rdsrc.obj,&
:F2:asm3b.obj,:F2:asm5n.obj,:F2:asm6n.obj,:F2:cntrln.obj,:F2:emitn.obj,:F2:listn.
obj,:F2:initb.obj,:F1:system.lib,&
:F1:plm80.lib,:F2:keyn.obj to :F2:asm805.rel map print(:F4:asm805.lnk)
```

```
With :F1: automatically mapped to ../../plm80v31/
     :F2: automatically mapped to obj/
     :F4: automatically mapped to lst/
```

The long command line below. (note the use of && and &. The quotes are to avoid the shell interpreting them)

```
../../thames -m ../../plm80v31/lib "&&" create system.lib "&&" add
attrib.obj,ci.obj,close.obj,co.obj,consol.obj,csts.obj,delete.obj,exit.obj,iochk.
obj,iodef.obj,ioset.obj,isis.obj,lo.obj,load.obj,memck.obj,open.obj,po.obj,read.o
bj,rename.obj,rescan.obj,ri.obj,seek.obj,whocon.obj,write.obj,ui.obj,uo.obj,upps.
obj,spath.obj,error.obj,getatt.obj,getd.obj,filinf.obj,chgacs.obj,detime.obj,"&"
v1p5.obj to system.lib "&&" exit
```

is pre-processed to

```
:F1:lib
create system.lib
add
attrib.obj,ci.obj,close.obj,co.obj,consol.obj,csts.obj,delete.obj,exit.obj,iochk.
obj,iodef.obj,ioset.obj,&
isis.obj,lo.obj,load.obj,memck.obj,open.obj,po.obj,read.obj,rename.obj,rescan.obj
,ri.obj,seek.obj,whocon.obj,&
write.obj,ui.obj,uo.obj,upps.obj,spath.obj,error.obj,getatt.obj,getd.obj,filinf.o
bj,chgacs.obj,detime.obj,&
v1p5.obj to system.lib
exit
```

```
With :F1: automatically mapped to ../../plm80v31
```

-i -- load applications with bad checksums

A small number of ISIS applications do not maintain the internal checksums, cobol80 in particular modifies internal data tables during configuration but does not update the record checksum. The -i option allows these files to be loaded. In most cases it is not needed.

-o & -u -- processing of ISIS application errors

Unfortunately ISIS does not directly support returning error codes to the invoking shell. Thames now supports a work around for this by inspecting the console output for error messages and if one is detected, thames itself will return a non-zero error code. Currently ASM80, LIB, LINK, LOCATE, PLM80 and IXREF are explicitly supported, however as many error messages are common across applications, most errors in other applications should also be detected.

There are additionally three options that are used to modify the error processing behaviour:

1. -u this causes UNRESOLVED (link) and UNSATISFIED (locate) errors to be ignored. The expected
   use is when creating overlays as these errors will often occur as part of the build process.

2. -o this causes OVERLAPS (locate) to be ignored; useful when adding object files to synthesise the junk data present in memory when Intel create the isis.t0 files.
   Note the obj2bin application now handles patch files and is now the preferred way of adding "Junk" data.

-MF file & -ME ext

To support the build process these options control whether thames monitors which files are processed. This information is then used to create a dependency file as gcc would do for a c file. The -MF file option specifies the file to create and the -ME ext specifies the extent of the generated file to monitor, if not specified it defaults to .obj.

Note transient files are handled and not recorded in the dependency file.

-T -- tmp file handling

Many ISIS applications use tmp files. Unfortunately as these file names are fixed, trying to run parallel builds results in applications reading/writing to the same files. If the -T option is specified thames maps any *.tmp, asm86.nam and asm86.ent files to unique temporary files, thereby allowing parallel builds.

```
Updated by Mark Ogden 10-Oct-2023
```