thames: An ISIS-II Emulator

(John Elliott, 17 November 2012, updated by Mark Ogden, 21 October 2017)

thames emulates enough of the ISIS-II environment to be able to run the following programs (used in the CP/M 3 build process):

- asm80: ISIS-II 8080/8085 macro assembler, v1.1
- 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)
- Modified to emulate 8080 (v0.1.2) this allows basic to work, also removed tstate counting which was not used. Partial implementatio of CSTS implementation (bios 7) using kbhit, unfortunately to getchar requires cr before returing. Not a major problem but we aware.
- **Note** to revert to z80 emulation uncomment the #define Z80 line in thames.h
- Added -i option to Thames (v0.1.3) that allows cobol80 to load

Other programs known to work include

- asm80 versions 3.0, 3.1 & 4.1
- plm80 versions 3.0, 3.1 & 4.0
- lib v2.1
- Fort80 v2.1
- Pasc80 v2.2 - from thames 0.1.1g
- Basic v1.0, v1.1, vx021 from thames 0.1.2 - note limitation on kbd hit which is sensed but needs cr to continue

Installing

Installation should just be a matter of the usual sequence of commands:

```
./configure
make
make install
```

A solution file for Visual Studio under windows is also provided for windows builds.

In use

You will need to set up environment variables to map Unix/Windows directories to ISIS-II drives. 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 forces all filenames to lowercase. For maximum ISIS compatibility, you should ensure that they are also in 6.3 format - no more than six characters, followed optionally by a dot and up to three further characters. This is not enforced, but ISIS programs may not support longer filenames.

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
```

Enhancements to the original thames

The latest version of thames supports a two enhancements to simplify working with tools like make.

Command line pre-processing

If you invoke thames with a -m option the command line will be pre-processed as follows

- 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.
- The current directory is mapped to :F0: if not defined in the environment
- 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.
- 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/asm4b.obj,obj/rdsr
c.obj,obj/asm3b.obj,obj/asm5n.obj,obj/asm6n.obj,obj/cntrln.obj,obj/emitn.obj,obj/listn.obj,obj/i
nitb.obj,../../plm80v31/system.lib,../../plm80v31/plm80.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:asm4b.obj,:F2:rdsrc
.obj,&
:F2:asm3b.obj,:F2:asm5n.obj,:F2:asm6n.obj,:F2:cntrln.obj,:F2:emitn.obj,:F2:listn.obj,:F2:initb.o
bj,: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,i
oset.obj,isis.obj,lo.obj,load.obj,memck.obj,open.obj,po.obj,read.obj,rename.obj,rescan.obj,ri.ob
j,seek.obj,whocon.obj,write.obj,ui.obj,uo.obj,upps.obj,spath.obj,error.obj,getatt.obj,getd.obj,f
ilinf.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,i
oset.obj,&
isis.obj,lo.obj,load.obj,memck.obj,open.obj,po.obj,read.obj,rename.obj,rescan.obj,ri.obj,seek.ob
j,whocon.obj,&
write.obj,ui.obj,uo.obj,upps.obj,spath.obj,error.obj,getatt.obj,getd.obj,filinf.obj,chgacs.obj,d
etime.obj,&
v1p5.obj to system.lib
exit
```

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

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.
3. -i this allows applications with invalid checksums to load. This is needed for cobol80. Normally you would not need this

Implementation notes

Areas of functionality not used by the CP/M 3 build tools are untested. These include the system calls RENAME, CONSOLE, ATTRIB, ERROR 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 Z80 execution.

Acknowledgements

- The Z80 emulation engine was written by Ian Collier.
- 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.
- John Elliott for the initial implementation.


```
Updated by Mark Ogden 22-Dec-2017
```