Reverse engineered Intel code

The src directory tree contains the key results of my reverse engineering of the Intel code. With few exceptions the code can be rebuilt from the provided sources to match at a byte level the original Intel binary images.

Each sub directory contains a makefile to allow the code to be rebuilt and additionally there is a makefile in the src directory that will recursively build all of the code. For all the make files there are several key targets namely

```
all         the default target
clean       removes intermediate build files
distclean   removes all but the minimum files needed to rebuild the code
rebuild     does distclean followed by all
verify      verifies the build against the reference binary image
```

Note the -j option for make can be used. Projects where this would cause a problem have a suitable makefile to protect against problems. In general this option will greatly speed up the builds.

In many of the sub directories, the source code is stored in a single packed file, ending with the extension _all.src. This makes finding and replacing globally across a project simpler as I do not have a fully fledged IDE for plm source. The format of this packed file is documented in misc.md and tools unpack.exe or unpack.pl can be used to unpack the files, although make will do this automatically.

Additionally many of the projects use an automatically generated include file using the ngenpex tool. This uses a database of definitions and will expand these into the include file based on identified references. By using this approach various changes e.g. parameter or literal renaming, are processed automatically across all the include files.

Unless otherwise highlighted the directories follow the following convention

```
application_version
e.g. isis.cli_4.1 is isis.cli for isis version 4.1
```

asm80_v4.1

This is one of the most complex builds and the comments and variable names in the corresponding C port are more up to date. The main source of the complexity is due to managing the various overlays and to auto generated different code variants for the various build models from a largely common code set.

To note, the files created

```
asm80       this is the base loaded which determines which version to use
asm80.ov0, asm80.ov1 and asm80.ov2, asm80.ov3 are the base and overlays used when
there             is limited memory
asm80.ov4   is the version with macro support
asm80.ov5   is the large memory version without macro support
asxref      is the asm cross reference utility
```

fpal_2.1

This is version 2.1 of intels fpal.lib file. This has so far only been partially documented

ftrans_1.0

This is the Intel ftrans utility used to copy files. In addition to the source files the command line format and the protocol used is documented in protocol.txt

help_1.1

The decompiled source for iPDS help v1.1

isdm3.2

Unlike the other directories the code here is for 8086, 80186 and 80286. Although I have some unreleased reverse engineering work for the libraries, the code here is mainly integrated into the build environment. Unfortunately the msdos based build tools cannot do parallel builds, nor do they support _ in a file or directory name hence the non standard name for this directory.

isis.cli_X.Y

These directories contain isis.cli reverse engineered source for ISIS versions 2.2, 3.4, 4.0, 4.1, 4.2 and 4.3.

The isis.cli for ISIS v1.1 is under the isis_1.1 directory

isis.ov0_4.X

These directories contain the decompiled source for isis.ov0 which provides directory scanning support. Part of the code appears to relate to remote directory lookup, however I do not know the protocol / system calls used and I suspect they relate to ISIS-III.

isis.t0_X.Y

These directories contain the reverse engineered source to the ISIS boot file isis.t0, for ISIS versions 2.2, 3.4, 4.0, 4.1, 4.2, 4.2w, 4.3 and 4.3w. Where needed appropriate patch files are provided to pad out the files to sector boundaries

The isis.t0 for ISIS v1.1 is located under the isis_1.1 directory.

Note isis.t0_2.2 uses the old fortran based PL/M compiler, all the others use pl/m 80 v3.1 or v4.0.

isis_X.Y

These directories contain the decompiled source for main ISIS OS file isis.bin, for ISIS versions 1.1, 2.2, 3.4, 4.0, 4.1, 4.2, 4.3 and 4.3w. ISIS 1.1 is particularly rare and the copy I have was only identified in the Cambridge Centre for Computing History archives, in September 2020, by one of their volunteers Jon Hales.

In addition the ISIS_1.1 directory contains decompiled source for

```
isis.t0, isis.cli, attrib, copy, dir, delete, edit, format, hexbin and rename
```

Note isis.bin for ISIS v1.1 appears to have been written initially in PL/M but hand modified, hence it is presented here in assembler. All other v1.1 code and isis.bin v2.2 use the fortran based PL/M compiler.

Later versions of the core isis.bin code increasingly used hand modified PL/M code, presumably to keep the size down.

ISIS 1.1 EDIT v1.2, appears to have been compiled by a different variant of the Fortran based PL/M compiler than I have access to. I manged to get the compiler to generated a binary match with some unstructured gotos and four bytes of patching. The four bytes are due to the fact that edit uses a GOTO from a nested procedure which re-initialised the stack as per more recent PL/M compilers. The versions I have do not do this stack initialisation, so I put two consecutive GOTO statements in two places, then patched the first of each pair to do the LXI SP, ...

isisUtil_X.Y and utils_2.2n

Various reverse engineered source for some of the utility applications in ISIS are in these directories. Specifically

```
v3.4  - binobj  -- note this was dropped in later ISIS versions
v4.3  - fixmap format idisk hexobj objhex submit vers
v4.3w - altmap format idisk
utils_2.2n - hexobj objhex  -- these are the latest versions of these utilities
             these come from the UTILS directory released with the ISIS.EXE
emulator
```

ixref_X.Y

Decompiled versions of the Intel PL/M cross reference tool.

Note to test this the application needs to access the ISIS.DIR file. In the tools directory there is a tool to create a version of this file sufficient to allow the tool to work.

kermit

A version of kermit for ISIS, made available with a makefile to build it.

lib_2.1, link_3.0, locate_3.0

Decompiled source of Intel's LIB 2.1, LINK 3.0 and LOCATE3.0. C ports of these utilities also exist in c-ports

plm_4.0

One of the more complex decomplications due to the overlays and the large number of shared object files.

The C port of this tool is available under c-ports and is likely to have more up to date comments and variable naming as it easier to debug the code to see what is actually happening.

plm80.lib

The assembler source for the plm80 support functions e.g. multiply, divide and word based arithmetic. I am only aware of one version of this library.

system.lib_4.0

PL/M and assembler code for the system.lib version 4.0.

Note there are minor variants of how the system.lib code was built but they are equivalent. The main variations are around whether ISISI is called directly or via a vector.

toolbox_X.0

The core for two Intel toolbox releases. Although the utility code was supplied as source, the libraries were not. The directory tree contains the reverse engineered source for these libraries.

A key issue with the libraries is that some of them were build using compilers that are not in the public domain. Indeed some may have been built using internal pre-release compilers. As a result is not possible to build all of the libraries exactly, although in principle equivalent versions could be built in assembler, using the asm80x wrapper to enable long variable names.

When originally reverse engineered some of tools I now have were not available, so some of the verification reports and ignores the differences.

Some examples of differences are

- cusp2.lib, was compiled with plm80 v1.0 which inlines some of the plm80 library functions and generates different code.
- For module MONITOR, the compiler despite claiming to be V3.1, generated sub-optimal code, so I suspect was compiled using an internal version.
- Several of the libraries are not identical despite all of the object modules in them being so. This appears to be due to a bug in the Intel librarian used, in that the dictionary locations are not all normalised e.g. block:sector 24H:00 is 23H:80H.

```
Updated by Mark Ogden 15-Sep-2020
```