Reverse engineering tools

The notes below document the tools I have developed for reverse engineering.

All the tools now support being invoked with -v and -V. The lower case version shows simple version information, the uppercase version provides additional git information to help identify the version. Both of these should be the only option on the command line.

Note prebuilt 32bit Windows versions of these tools along and additional tools from my GitHub c-ports repository are included as part of my GitHub Intel80Tools repository. The documentation in the Intel80Tools repository provides information on all the tools along with information on various perl and windows cmd files scripts.

aomf2bin.exe

This utility take an absolute omf85, omf86 or omf286 file and creates binary images suitable for a prom programmer. There is an ability to set the base address of the prom and, whether to pad to a prom boundary, with 0 or 0xff. Optionally separate files can be created for odd and even bytes

```
usage: aomf2bin -v | -V | option* infile [outfile | -o odd_outfile | -e
even_outfile]+
   supported options are
   -b address - sets base address of rom
   -p         - pads image to eprom boundary
   -v / -V    - show version info and exit
   -z         - sets uninitialsed data to 0 instead of 0xff
```

disIntelLib.exe

A homegrown utility to auto disassemble an Intel omf85 library into individual files. During the disassembly, whether the original code was PL/M or ASM is noted and the extension named accordingly.

```
Usage: disIntelLib infile
```

Note, the first build of this tool may fail, since it uses generated files. Subsequent build attempts should be ok.

dumpintel.exe

Dumps the detail of the content of omf85, omf51, omf96 and omf86 files. omf96 currently only shows the record types as I have no samples to verify. The others decode as per the intel specifications with some of the none Intel additions for omf86

```
usage: dumpintel -v | -V | objfile [outputfile]
```

fixobj.exe

Supports modifying omf85 files to work around lack of historic / unreleased compilers that are currently not available.

```
Usage: fixobj [-(v|V)] |  [-h] [-l] [-n] [-p file] [-t(f|p|u)]  [-v hh] infile
outfile
Where:
  -v | -V     shows version information - must be only option
  -h          create missing segdefs in MODHDR for CODE..MEMORY
  -l          remove @Pnnnn library references
  -n          mark as a non main module
  -p file     parses the file for patch information. See below
  -tf         sets translator to FORT80
  -tp         sets translator to PLM80
  -tu         sets translator to Unspecified/ASM80
  -v hh       sets version to hh hex value
 outfile      outfile can be the same as infile to do inplace edits

Using the -p option supports more advanced patching
the file can contain multiple instances of the following line types
n [(a|c) addr]        non main module with optional non compliant entry point
p addr [val]*         patch from addr onwards with the given hex values
                      addr is absolute for apps, else code relative
r oldname [newname]   renames public/external symbols from oldname to newname
                      names are converted to uppercase and $ is ignored
                      omitting newname deletes, only vaild for public
                      valid chars are ?@A-Z0-9 and length < 32
s addr                force split in record at absolute addr
the command line options with out leading - can also be used
text from # onwards is treated as a comment and blank lines are skipped
```

In addition to the documented options above, all record checksums are recalculated, with previously invalid ones being highlighted.

Option	Typical usage
-h	Although PL/M emits seg size info in the MODHDR, the linkers omit these if the size is zero. Adding in libraries see below, causes the linker to remove this seg size information. The -h option forces the standard segments CODE, DATA, STACK and MEMORY to be included even if their size is zero.
-l	This is used to allow PL/M v1.0 behaviour to be synthesised. This older version includes some of the library routines in the object files it creates, which the more recent compilers don't. Although it is possible to link the missing library routines, the public definitions of the plm80.lib routines that this creates causes conflicts when linking. The -l option strips the public definitions out of the synthesised object module.
-n	Some older applications are composed of separate applications joined together, however the Intel linker objects to linking two or more main modules. In principle converting the files to hex and joining them would work, this option makes the task simpler by removing the main program flag from the MODEND record. See patch file notes for a more advanced version.

Option	Typical usage
-t?	These options allow the trn field of the MODHDR record to be set to flag the original file as being PL/M80, FORT80 or ASM80/Unspecified. One use of this is to reset the trn to PL/M80 when the -l option is used, as linking the library routines will reset the trn to ASM80/Unspecified.
-v	This allows the version files of the MODHDR to be forced to a particular value. For example to make it look like the object file has been created by version 1.0 of the PL/M compiler

-p patchfile

The patch file option is used when more complex modifications are needed to make an object file match an original version. Multiple -p options are allowed.

Option	Typical Usage
n	This performs the same basic operation as the -n command line option, however it also allows an entry point to be defined, with a | c setting the seg id to ABS or CODE respectively and the address being the offset. According to the OMF specification the entry info is ignored for non main modules and should be set to 0, however PLM v1.0 modules does not adhere to this standard, this option allows the PLM v1.0 behaviour to be mimicked.
p	This is used to patch a file in cases where it is not possible to get known compilers to generate the same code. It only patches defined content and cannot be used to set data or uninitialised areas. Additionally fixup information is not changed, so care is needed when patching non located modules to make sure than only fixed data or offsets are modified.
r	There are two primary uses of this. One is to delete or mask public references in a more targeted manor than the -l option. The second is to rename between ASM80 short names and the compiler long names.
s	Some historic files appear to have splits in longer OMF CONTENT records, possibly due to older linkers or small memory build machines. Although this split has no impact on the loaded image, this option is used to force a split, so that exact binary images can be created. The inverse is not needed as recent versions of link/locate can be used to join records.

Note the -t, -v and patch file s option are for cosmetic changes, images will be equivalent with or without them.

Note **fixobj** is not able to resolve all differences between old files and those created by more recent tools, it does however allow creation of equivalent files. The key outstanding issue relates to problems when fixing the embedded library code that PLM v1.0 generates. In linking in the library functions, the linker does not emit the records in the same sequence, nor does it create the same the same record splits. Whilst this has no impact on subsequent use, it does mean that the files generated will not be a byte for byte match. The only resolution of this would be to write a bespoke linker.

genpatch.exe

This is used to auto generate the patch files for obj2bin. They take as input the generated omf85 object file and the target binary file and generates the specified patchfile.

```
usage: genpatch [-i] [-s] [-z] objFile binfile patchfile
where -i    indicates to interpret the bin file as an Intel .BIN file
      -s    reserved for future use to show strings as a trailing comment
      -z    by default obj2bin auto fills uninitialised data to 0 and the
patchfile
            assumes this. The option forces the 0 initialisation to be in
patchfile
```

install.cmd (in Scripts directory)

This is a windows batch file that is mainly used as part of the visual studio build process to auto copy compiled code to target directories. The master repository for this tool is my GitHub repository versionTools

```
usage: install.cmd file_with_path installRoot [configFile]
       configFile defaults to installRoot\install.cfg

install.cfg contains lines of the form type,dir[,suffix]
  where type is the closest parent directory ending in debug or release on the
path
  to the name of the file to copy. The test is case insenitive.
  dir is the directory to install to; a leading + is replaced by installRoot
  suffix is inserted into the installed filename just before the .exe extension
  In both dir & suffix a $d is replaced by the current local date string in
format yyyymmdd
  and a $t is replaced by the current local time string in format hhmmss
  All lines where type matches the input file's directory name are processed

Example with install.cfg in the current directory containing the line
x86-Release,+prebuilt
x86-Release,d:\bin,_32

install . path\x86-Release\myfile.exe
copies myfile to .\prebuilt\myfile.exe and d:\bin\myfile_32.exe

Control lines are also supported and they change what files the control lines
apply to
Each control line's impact continue until the next control line
A control line starting with a + enables processing only for the list of files
after the +
One starting  with a - only enables processing for files not in the list
a file name of * matches all files so +* renables processing for all files
-* stops all processing until the next control line (of limited use)
```

isisc.exe [depreciated]

Compares files using Intel's BIN format. This is now depreciated and the equivalent capability can be achieved in one of two ways

1. Convert the files to OMF85 format using binobj, then using omfcmp to check for differences
2. Use obj2bin with a patch file to add the junk data at the end of the file and use omfcmp or fc /b to compare

```
Usage: isisc -v | -V | file1 file2
```

isisu.exe

This utility dumps the block ranges and start address from an Intel BIN format file. This is now of limited use as I tend to convert the BIN files to OMF and load directly into my disassembler, preserving the uninitialised data information.

```
Usage: %s -v | -V | file
```

obj2bin.exe

This utility is designed to support the creation of .COM, .T0 and .BIN files and includes the ability to patch the resultant file. Patching is potentially needed for two reasons.

1. Intel's tools support uninitialised data but the .COM and .T0 are pure memory images. By default obj2bin will set these locations to 0, however the original binary production is likely to have used data in memory at the time of creation. Patching allows this random data to be matched
2. It is possible that the binary images have data after the end of the program to align with sector boundaries, The patch capability allows this to be added at the end of the file.

```
Usage: obj2bin -v | -V |  [-i | -j] infile [patchfile] outfile
Where -v/-V provide version information
      -i    produces Intel formast .BIN files
      -j    writes a jmp to entry at the start of file using
            any initial lxi sp, is skipped.
            the first byte must either uninitalised or a jmp (0c3h)

The patch file, if used  has the following format and operates in one of two
modes
PATCH the initial mode and APPEND which starts after the key word APPEND is seen
at the start of a line, the keyword is case insensitive. The two modes are
required
since for .BIN files in particular the extra data needs to occur after the start
record
Note for all lines leading space is ignored and all values are in hex.
The address used to apply the patch is determined as follows
PATCH mode:
    Each line starts with a hex address to start the patch, anything other than a
hex       value is seen the line is ignored

APPEND mode:
    Initially the address is set to the current highest used location when APPEND
is        seen there after the address increments for each new value appended
```

```
Once the patch address is known all other data is a set of space separated values
specifiers in one of the following
    value ['x' repeatCnt]
        where value can one of
        hexvalue
        'string'                 note string supports \n \r \t \' and \\ escapes
        -                        set to uninitialised
        =                        leave unchanged i.e. skip the bytes
Note in APPEND mode, - and = are teated as 0
If the value is not a valid hex value, string, - or = the rest of the line is
skipped
if the x is present and repeatCnt is invalid an error is reported

The above noted, it is safer to use a semicolon to start a comment as this is
treated
as the end of line
```

omfcmp.exe

This tool is designed to intelligently compare intel OMF85 files, however it will revert to comparing binary files.

```
Usage: omfcmp -v | -V | file1 file2
```

patchbin.exe [depreciated]

Patch a binary .COM file. The original reason for creating this is now manageable with obj2bin

```
Usage: patchbin -v | -V | patchfile filetopatch
where patch file has lines in the format
address value*
both address and value are in hex and the filetopatch is assumed to be load ax
100H
The file is extended if necessary
```

plmpp.exe

Only PL/M v4 supports a pre-processor. This utility provides a pre-processor for older versions of PL/M.

```
usage: plmpp -v | -V |  [-f] [-F] [-sVAR[=val]] [-rVAR] [-o outfile] srcfile
Where -f                - expands a level of include files, each -f does another
level
      -F                - expands all include files regardless of depth
      -sVAR[=val]       - same as PL/M's SET(VAR[=val])
      -rVAR             - same as PL/M's RESET(VAR)
      -o outfile        - specifies the output file, otherwise outputs to stdout
```

unpack.exe

This file support extracting files form a packed source file.

Note unlike the perl variant of this utility in Intel80Tools, this version always extracts and updates the timestamp.

```
Usage: unpack -v | -V | [-r] [file]
if file is not specified the default file is directory_all.src
where directory is current directory name
-r does a recursive unpack
```

version.cmd (in Scripts directory)

This is used to generate version information from a git repository for visual studio builds. The master repository for this tool is my github repository versionTools

```
usage: version [-h] | [-q] [-f] [-a appid] [CACHE_PATH OUT_FILE]

 When called without arguments version information writes to console

 -h          - displays this output

 -q          - Suppress console output
 -f          - Ignore cached version information
 -a appid    - set appid. An appid of . is replaced by parent directory name
 CACHE_PATH  - Path for non-tracked file to store git version info used
 OUT_FILE    - Path to writable file where the generated information is saved

 Example pre-build event:
 CALL $(SolutionDir)scripts\version.cmd "Generated" "Generated\version.h"

 Note if the OUT_FILE ends in .cs an C# version information file is created
otherwise
 a C/C++ header file is generated.
```

```
Updated by Mark Ogden 26-Nov-2020
```