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.

abstool

This is a general purpose tool for converting between a number of absolute file formats and optionally applying patches. It subsumes obj2bin, which is now depreciated.

10-Sep-2025: Note to allow multiple input files the command line syntax has changed. Patch file and output file must now be specified with the appropriate option rather than implied as the last one/two files.

```
Usage:
abstool [-l addr] [-a|-a51|-a85|-a96|-h|-i] -p patchfile -o outfile infile+
Where:
-l addr  override the load address for binary images, default is 100H (CP/M)
         subsequent binary images are concatenated
-a51     produce AOMF51 file, note no symbols or debug info
-a|-a85  produce AOMF85 file, note no symbols or debug info
-a96     produce AOMF96 file, note no symbols or debug info
-h       produce Intel Hex file, note no symbols
-i       produce Intel ISIS I bin file
-p file  specify patch file
-o file  specify output file, if omitted only a summary of infile is produced
File format can be AOMF51, AOMF85, AOMF96, Intel Hex, Intel ISIS I Bin or binary image
The last format specified is used, default is binary image

Also supports single arguments -v, -V for version info and -h for help
```

The optional patch file has contains lines which are interpreted int one of two modes, PATCH and APPEND, with PATCH being the initial mode

Numbers are all treated as hex and unless part of a string, blanks are ignored and punctuation ends a line, except for $ when used in $START or $number.

In PATCH mode each line starts with a patch address followed by any number of patch data values. The patch address is a 16 bit value and can be entered as a number, $number or $START, see below.
In APPEND mode, only patch data values are supported; the patch address is implicit

Any number of meta token assignments (see below) can be interspersed between patch values

Patch data values

Patch data values can be either of the following

```
'APPEND'     switch to APPEND mode, rest of line is interpreted as per APPEND mode
value ['x' repeatCnt] where repeatCnt is a hex number and value is one of
     number   a hex number
    'string'  C string escapes \a \b \f \n \r \t \v \' \" \\ \xnn and \nnn are supported
     -        set to uninitialised. (error in APPEND mode)
     =        leave unchanged i.e. skip the bytes. (error in APPEND mode)
     $START   patches with the two byte start address of the program
     $number  patches with a two byte hex number
```

Meta token assignments

```
metaToken ['='] value
where metaToken is one of
  TARGET  issues a warning if the specified target format is different
  SOURCE  issues a warning if the actual source file format is different
  LOAD    issues a warning if the actual load address is different
  START   set the start address if not set, else warn if source file start is different
  NAME    sets the name for AOMFxx formats otherwise ignored
  DATE    sets the date field for AOM96 otherwise ignored
  TRN     sets the TRN value for AOMFxx otherwise ignored. Error if invalid
  VER     sets the VER value for AOMF85 otherwise ignored
  MAIN    sets the MAIN module value for AOMF85 & AOMF96 (bit 0 only used)
  MASK    sets the MASK value f or AOMF51 (low 4 bits only)
and value is one of
  fileFormat, used for TARGET and SOURCE. Vaild values are
    AOMF51   - Intel absolute OMF for 8051
    AOMF85   - Intel absolute OMF for 8080/8085
    AOMF96   - Intel absolute OMF for 8096
    ISISBIN  - Intel ISIS I binary
    HEX      - Intel Hex
    IMAGE    - Binary Image
  string for NAME and DATE
  hex for other. Note LOAD and START are word values others are byte values
```

The genpatch tool can be used to create patch files in the right format

Note APPEND mode is needed to support output files that are not simple binary images. A normal patch would incorrectly include the extra data within the loaded image

aomf2bin

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

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.

dumpomf

Dumps the detail of the content of omf85, omf51, omf96 and omf86 files. Interpretation of the various formats is per the intel specifications with some extensions for omf86. Due to lack of samples, limited testing has been done on omf96. This supersedes **dumpIntel** which has now been depreciated.

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

fixobj

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. For absolute file **abstool** may be a better choice.
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 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

It compares two absolute files and generates patch information that **abstool** can use to generate files.

Supported absolute formats are AOMF51, AOMF85, AOMF96, ISIS BIN, Intel Hex and binary images.

```
usage: genpatch (-v | -V | -h)  | [-b addr]  infile targetfile [patchfile]
Where -v/-V provide version information
      -h       shows this help
      -l addr  set explicit load address for binary image files. Default 100H (CP/M)
File format can be AOMF51, AOMF85, AOMF96 Intel Hex, Intel ISIS I Bin or binary image
If patchfile is omitted then the patch data is output to stdout
```

getVersion.cmd/getVersion.pl (in Scripts directory)

Tool to generate version string for builds. It is the successor to version.cmd which is gradually being replaced.

```
usage: getVersion -v | -h | [-q] [-w|-W]

 When called without arguments version information writes to console

 -v          - displays script version information
 -h          - displays this output

 -q          - Suppress console output, ignored if not writing to file
 -w          - write file if version changed
 -W          - write file even if version unchanged

 The default generated file is _version.h as a C/C++ header file
 An optional version.in can be used to override these

 Example pre-build event:
 CALL $(SolutionDir)scripts\getVersion.cmd -W
```

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 [replaced by genpatch]

isisu.exe [replaced by abstool]

obj2bin [replaced by abstool]

omfcmp

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 [replaced by abstool]

plmpp

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

This file support extracting files form a packed source file.

Note unlike the perl variant of this utility in Intel80Tools, this version currently 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) [depreciated]

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.
```

wcat extended cat file

```
usage: wcat [options] file*
Concatenate files, or use standard input if no file is specified
Options are:
-b[n]    Binary copy. If n present force append at 2^n boundary
-o file  Write to specified file, default is standard output in option is omitted
Default copy mode is text. Converting line endings to \r\n and stopping at ^Z

Also supports single arguments -v, -V for version info and -h for help

With the -b option by default the binary files are concatenated without a gap, the optional
n value modifies this and adds zero byte padding to make sure the file is added at a 2^n
boundary e.g.
-b1     forces word alignment
-b4     forces 16 byte paragraph alignment
-b7     forces 128 byte alignment - e.g. ISIS sector
-b8     forces 256 byte alignment
```

```
Updated by Mark Ogden 10-Sep-2025
```