Make support for ISIS tools

The current repository uses gnu make to build the various isis tools. To keep the the makefiles simple, a set of macros and rules have been predefined in an included file isis.mk.

This document describes the usage of isis.mk, but assumes some basic knowledge of make and makefiles. Some examples of the use of isis.mk are also provided to illustrate the how isis.mk simplifies the makefiles.

Structure of the makefile

The basic structure of the makefile is

```
# path to the top of the build tree. Not needed if ITOOLS is defined
ROOT := ../
#
# optional pre include variables
# ...
include $(ROOT)/tools/isis.mk
# use include $(ITOOLS)/tools/isis.mk   if ITOOLS is defined
#
# variable definitions and rules
# note target all:: must be defined
```

Variables used by isis.mk

Variables defined pre inclusion of isis.mk

- **ROOT** - this is set to the location of the top of the tools tree. Normally it would be a relative path
- **ITOOLS** - this is an optional environment variable which should point to the top of the tools tree. If defined then it will make it easier to support developments outside the intel tools tree, with out hard coding the tools tree location in the ROOT variable.
- **LST** - this is set to the location of the listing file directory. It can be omitted if the current directory is used.
- **OBJ** - this is set to the location of the object file directory. It can be omitted if the current directory is used.
- **PEXFILE** - this is set to the file name of the pexfile. It should be omitted if **ngenpex** is not being used
- **NOVERIFY** - set to **T** if you wish to prevent the automatic verify rule being created. Useful if you wish to create a bespoke verify rule or verify isn't applicable. Most of the time this can be omitted.

  Alternatively the variable can be set to the list of files to skip verification. In this later case the variable can be defined after the include of isis.mk

  If required the following should preferably be defined before isis.mk.
- **SRC** - this is set to the location of the source file directory. It can be omitted if the current directory is used.

- **ISIS_F0** - this is set to the directory containing the isis drive F0 files. It can be omitted if this is the current directory
- **COMPARE** - set to the program used to compare files. Not needed if the default omfcmp is being used.

  *Definitions after isis.mk must ensure path names are in unix format and for **SRC** there should be no trailing space. The macro **fixpath** can be used to force this.*

  Note, when comparing files, path names are not translated to windows format, so some windows tools e.g fc may fail

Variables modified or defined in isis.mk

- **ROOT** - the directory path is converted to unix format and any trailing / is removed. Note when the **ITOOLS** environment variable then this is used to set **ROOT** if it has not been set explicitly.
- **PATH** - environment variable updated to add the path to the unix tools
- **SHELL** - set to bash.exe
- **COMPARE** - set to $(ROOT)/tools/omfcmp if not defined
- **OBJ,LST,SRC** - set to current directory if not defined, paths converted to unix format and any trailing / removed.
- **ISIS_F0** - set to current directory if not defined
- **ISIS** - set to the thames program with the -m option set
- **PLMPP** - set to the plmpp program, a standalone PL/M pre-processor for pre V4 compilers
- **NGENPEX** - set to the ngenpex program
- **MKDEPEND** - set to the makedepend program. *Depreciated since thames can now generate the dependency information.*
- **HEXOBJ** - set to the C port of Intel's hexobj utility
- **OBJBIN** - set to the obj2bin program
- **PLM81 & PLM82** - set the the first and second pass of the old PL/M compiler - my port from the Fortran version.
- **PLM80** - the version of the PLM80 compiler to be used. Set to 4.0 if not previously specified
- **PLMFLAGS** - set to code if not defined
- **ASM80** - the version of ASM80 to be used. Set to 4.1 if not previously specified
- **ASM48** - the version of ASM48 to be used. Set to 4.2 if not previously specified
- **ASM80X** - set to the asm80x.pl program, which is a wrapper around asm80 to support long variable names. Note this previously was set to asmx.pl which is a more experimental wrapper with struct support.
- **LIB** - the version of the LIB to be used. Set to 2.1 if not previously specified
- **LINK** - the version of the LINK to be used. Set to 3.0 if not previously specified
- **LOCATE** - the version of the LOCATE to be used. Set to 3.0 if not previously specified
- **FORT80** - the version of the FORT80 compiler to be used. Set to 2.1 if not previously specified
- **plm80.lib** - simple variable to reference plm80.lib
- **system.lib** - simple variable to reference system.lib associated with specified PLM80 version
- **system.lib,3.0** - simple variable to reference the plm v3.0 system.lib file
- **system.lib,3.1** - simple variable to reference the plm v3.1 system.lib file
- **system.lib,4.0** - simple variable to reference the plm v4.0 system.lib file
- **fpal.lib,2.0** - simple variable to reference fpal.lib v2.0
- **fpal.lib,2.1** - simple variable to reference fpal.lib v2.1
- **_masterfile** - this is set to the source master file if present. Master file names end in *_all.src*
- **space** - set to the space char - used in make macros
- **comma** - set to the comma char - used in make macros

Variables defined pre or post inclusion of isis.mk

- **TARGETS** - the list of default files to build see **all** target below.

  It is also used with **distclean** and **verify** targets
- **PROTECT** - set to a list of files to keep as part of the distribution. This is only needed if *master files* are being used. The *master file, makefile, mk and any $(REF) directory* are protected automatically.
- **REF** - set to the location of the directory containing the reference file(s). Recommended to be defined after isis.mk to allow use of the ipath macro
- **ASMFLAGS** - common options for asm80 - **print** and **object** should not be included as they are used internally. Also used for asmx.pl
- **ASM48FLAGS** - common options for asm48 - **print** and **object** should not be included as they are used internally
- **FTNFLAGS** - common options for fort80 - **print**, **object** and **workfiles** should not be included as they are used internally
- **PLMFLAGS** - common options for plm80 - **print** and **object** should not be included as they are used internally. Set to code if not defined
- **LINKFLAGS** - common options for link - **map** and **print** should not be included as they are used internally
- **LOCATEFLAGS** - common options for locate - **print** should not be included as it is used internally
- **ISIS_Fn** - where n is a digit 0-9.

  Although thames now supports automatic directory - drive mapping, it is occasionally necessary to explicitly define the mapping of an ISIS drive. For example to define a specific directory for include files. These variables must use the make export feature.

  ```
  For example to map ./include to drive F3 use
  export ISIS_F3 := ./include/
  ```

Variables modified post inclusion of isis.mk

For more complex builds it may be necessary to modify variables post isis.mk. Some of the more common examples are

ASM80, ASM48, FORT80, LIB, LINK, LOCATE, PLM80 - If a file needs to be compiled with a specific version of a tool you can set these variables to specify the appropriate version. In practice it is likely that this will only be used for plm80 and fort80 as the others should produce equivalent code. Examples:

```
Using plm V3.1 for all plm builds
PLM80 = 3.1

Using plm V3.1 for a subset of files with V4.0 for the rest

list of files: PLM80 = 3.1

To support a command line specification of toolset e.g
    make V31 file.obj
define a rule
V31:
    $(eval PLM80=3.1)
```

Other than the variables noted above and the macros noted below no other names are currently reserved or modified.

Macros defined in isis.mk

A number of macros are defined in isis.mk to simplify the invocation of the isis build tools. Additionally a number of supporting macros are used that may be of use in more complex makefiles.

Build macros

For these macros file names should use the unix style pathname. Thames maps these to ISIS drive names, but see the note on ISIS_Fn above.

- **asm80** - assemble an asmfile to produce the specified object file and a listing file, (asmfile with ext .lst) in the $(LST) directory.

  ASMFLAGS are used and optional target specific options can be given except for **print** and **object** which are used internally.

  Usage: $(call asm80,objfile,asmfile[,target specific options]) **Note** unlike PL/M-80, LIB, LINK and LOCATE, ASM80 does not support the & character to extend long lines. To support a long list of command line options, these need to be saved in a file using the same format as they would appear in the assembly file. The command line then needs to include this file using using the syntax INCLUDE(filename). Alternatively this file can be included in the assembly file itself or some of the options can be specified there, to reduce the length of the command line.

- **asm48** - assemble an asmfile to produce the specified object file and a listing file, (asmfile with ext .lst) in the $(LST) directory.

  ASM48FLAGS are used and optional target specific options can be given except for **print** and **object** which are used internally.

  Usage: $(call asm48,objfile,asmfile[,target specific options])

- **asm80x** - assemble an asmfile with long name and structure support to produce the specified object file and a listing file, (asmfile with ext .lstx) in the $(LST) directory. *Currently experimental.*

  ASMFLAGS are used and optional target specific options can be given except for **print** and **object** which are used internally.

  Usage: $(call asm80x,objfile,asmfile[,target specific options])

- **fort80** - compile the specified ftnfile, to produce the specified object file and a listing file, (ftnfile with ext .lst), in the $(LST) directory.

FTNFLAGS are used and optional target specific options can be given except for **print**, **object** and **workfiles** which are used internally.

Usage: $(call fort80,objfile,ftnfile[,target specific options])

- **plm80** - compile a *file*.plm file, producing the specified object file and a listing file *file*.lst in the $(LST) directory.

  PLMFLAGS are used and optional target specific options can be given except for **print** and **object** which are used internally.

  Pre running plm80 additional processing is done as follows

  ```
  if $(PEXFILE) is defined
    ngenpex will be run to generate any .ipx file
  else
    makedepend is run to generate a dependency file in .deps
  ```

  `Usage: $(call plm80,objfile,asmfile[,target specific options])`

- **link** - link a set of files producing the specified relocatable file and a listing file in the $(LST) directory. The listing file has the same name as the relocatable file but with the extension .lin (was lnk, but windows treated as shortcut)

  LINKFLAGS are used and optional target specific options can be given except for **print** and **map** which are used internally.

  Note. Unlike other macros the listing file does not use the input filename as the basis of its name.
  `Usage: $(call link,relocfile,object files[,target specific options])`

- **link-nocheck** - this is the same as link with the exception that unresolved names are not treated as an error. It is designed to support building overlay files.

  `Usage: $(call link-nocheck,relocfile,object files[,target specific options])`

- **locate** - locates a file producing the specified file and a listing file in the $(LST) directory. The listing file has the same name as the relocatable file but with the extension .map.

  LOCATEFLAGS are used and optional target specific options can be given except for **print** which is used internally

  `Usage: $(call locate,target,relocfile[,target specific options])`

- **locate-nocheck** - this is the same as locate with the exception that unsatisfied names are not treated as an error. It is designed to support building overlay files.

  `Usage: $(call locate-nocheck,target,relocfile files[,target specific options])`

- **locate-overlaps** - this is the same as locate with the exception that overlaps names are not treated as an error. *Depreciated*

  It was designed to support using obj2bin by including junk data to synthesise what was in memory when Intel originally built isis.bin. A newer cleaner option is to use a patch file with the obj2bin command line. *This is now considered depreciated.*

  `Usage: $(call locate-overlaps,target,relocfile files[,target specific options])`

- **rm-symbols** - simple variant of locate that removes symbols from an existing located file. No options are supported.

  `Usage: $(call rm-symbols,target,source)`

- **lib** - build a specified library from a set of object files.

  Since lib does not have any print options, please use shell redirection to capture logs.

```
Usage: $(call lib,target,objects)
```

Support macros

Changing paths

The following macros convert an input list of files by replacing any existing path of each file with a new path

```
Usage:
    $(call objdir,files)        # new path is $(OBJ)
    $(call srcdir,files)        # new path is $(SRC)
    $(call lstdir,files)        # new path is $(LST)
```

Listing file names

Generate a listing file name based on the first file name passed in. This done by taking the base file name, adding the appropriate extension and path $(LST)

```
Usage:
    $(call lin,file)             # creates .lin file name
    $(call map,file)             # creates .map file name
    $(call lst,file)             # creates .lst file name
```

Utility macros

These are mainly used internally however there may be occasional need to use them elsewhere, especially fixpath, ipath and ifile

- **fixpath** - if specified file is blank convert to **.** else convert **** to **/** in file names and remove trailing **/** for all files

  ```
  Usage: $(call fixpath,files)
  Example:
      SHARED = $(call fixpath,..\..\src\)
      HERE = $(call fixpath,$(HERE))  # assuming HERE is not already defined
  sets
      SHARED = ../../src
      HERE = .
  ```

- **ipath** - returns the directory containing an isis tool. The version can be omitted if there is only one version and it is not contained in a sub directory

  ```
  Usage: $(call ipath,tool[,version])
  Example: (Assuming ROOT is ../..)
      $(call ipath,plm80,3.1)
  returns
      ../../itools/plm80/v3.1
  Example:
      $(call ipath,plm80.lib)
  returns
        ../../itools/plm80.lib
  ```

- **ifile** - returns the full path to the isis tool. As with ipath version can be omitted

```
Usage: $(call ifile,tool[,version])
Example: (Assuming ROOT is ../..)
    $(call ifile,link,2.1)
returns
    ../../itools/link/2.1/link
```

- **notlast** - returns all but the last item in a list

  Usage: $(call notlast,list) Example: $(call notlast,1 2 3 4 5) returns 1 2 3 4

- **mklist** - converts a space separated list into a comma separated list

  Usage: $(call mklist,list) Example: $(call mklist,a b c) returns a,b,c

Rules defined in isis.mk

As with normal make usage a number of predefined rules are added by isis.mk. Several of these support the specific way I work but should not get in the way of other usage.

In particular there is additional support for master files which contain all source in a single file, with each sub file marked with a control L followed by the sub file name. This combined file makes it easier to modify groups of files as I decompile / disasemble them.

Additionally I use ngenpex, which is my enhanced version of the ISIS toolbox genpex utility. If PEXFILE is specified then the .ipx files are generated automatically as part of the plm build.

Implicit build rules

Only five implicit build rules are defined, one to generate a hex file from an asm48 source file, the rest generate .obj files from .plm, .asm, .f and .asmx files. An .asmx file is an Intel asm80 compatible assembly file, but with long variable names up to 31 characters, also _ and $ can be used as null separators in variables, similar to the use of $ in PL/M. The rules are

```
$(OBJ)/%.obj: %.plm  | $(OBJ) $(LST)
    $(call plm80,$@,$<)

$(OBJ)/%.obj: %.asm  | $(OBJ) $(LST)
    $(call asm80,$@,$<)

$(OBJ)/$.obj: %.f | $(OBJ) $(LST)
    $(call fort80,$@,$<)

$(OBJ)/%.obj: %.asmx | $(OBJ) $(LST)
    $(call asm80x,$@,$<)

$(OBJ)/%.hex: %.a48 | $(OBJ) $(LST)
    $(call asm48,$@,$<)
```

The **| $(OBJ) $(LST)** is used to auto create directories

.PHONY targets

The following .PHONY targets are defined in isis.mk

- **all::** - the default rule. If a master file is detected it will make sure that the files are auto extracted. The main make file should also include a all:: rule.
- **clean::** - used to clean *.obj, *.abs, *.lst, *.lin, *.map, *.hex, *.bin files.

If **$(OBJ)**or **$(LST)** are not set to the current directory they are deleted.

A clean:: rule can be added to the main makefile if required to delete additional files. Note as toolbox contains a utility clean, the clean target noted here would normally clash. So in this case the makefiles generate a file Clean which does not clash as normal gnu make is case sensitive. However if the case insensitive version of gnu make is used the toolbox makefiles will fail.

- **distclean::** - in addition to the files deleted by clean::, this rule deletes the $(TARGETS) files and any .deps directory.

  If a master file is used all non protected files are deleted. See also **PROTECT** variable.

- **rebuild:** - runs targets distclean followed by the all target
- **verify:** - verifies the $(TARGETS) files with those of the same name in the $(REF) directory.

  If **NOVERIFY** is specified then this target is not generated by isis.mk, however one can be included in the main makefile.

Note to help with automated clean up isis.mk defines the target **.DELETE_ON_ERROR:**.

Examples

The isis tools build tree contains multiple examples of makefiles using isis.mk, however the following are fragments with commentary to help the reader better understand how to write their own makefiles. Commentary preceeded by ~~

lib_2.1 makefile

```
# path to root of build tree
ROOT := ../..                          ~~ Points to top of build tree
TARGETS := lib                         ~~ what we are building
PEXFILE:=lib.pex                       ~~ mandatory variable as lib uses ngenpex

include $(ROOT)/tools/isis.mk

# build options
LOCATEFLAGS:=SYMBOLS LINES             ~~ map file will show local symbols & debug
info
PLMFLAGS:=DEBUG                        ~~ generate the debug info
LINKFLAGS:=

objs =  lib.obj lib1.obj isis1.obj isisa.obj isis2.obj lib3.obj lib4.obj
~~ objs lists the object files needed. Generated using the implicit plm rule

all::
    $(MAKE) $(TARGETS)                 ~~ mandatory default rule, builds lib
                                       ~~ note ::
                                       ~~ using the separate $(MAKE) line make sure
that
                                       ~~ auto extracted files are handled
correctly

lib: lib.rel                           ~~ how lib is built from a reloc file
    $(call locate,$@,$^,code(3680H) name(lib) stacksize(90) purge)
    ~~ cf. $(call locate,target,relocfile[,options])
    ~~ here target = $@ = lib
    ~~   relocfile = $^ = lib.rel
```

```
    ~~     options = code(3680H) name(lib) stacksize(90) purge

    .INTERMEDIATE: lib.rel              ~~ the reloc file will be deleted after
use
    lib.rel: $(objs)                    ~~ how the reloc file is built
        $(call link,$@,$^ $(plm80.lib))
    ~~ cf. $(call link,relocfile,objects[,options])
    ~~ here relocfile = $@ = lib.rel
    ~~        objects = $^ $(plm80.lib) = lib.obj lib1.obj isis1.obj isisa.obj
    ~~                                    isis2.obj lib3.obj lib4.obj
    ~~                                    ../../itools/plm80.lib/plm80.lib
    ~~       i.e. $(objs) + plm80.lib from the itools directory
    ~~ options are not used
```

Part of tex makefile

```
ROOT=../..
REF=ref
TARGETS=tex10.com tex12.com tex21.com tex21a.com
PROTECT = build.ninja tex10.patch tex12.patch tex21.patch   ~~ avoid
clean/distclean from removing patch files
include $(ROOT)/tools/isis.mk

PLMFLAGS=code optimize
ASMFLAGS=

all::
    $(MAKE) $(TARGETS)                  ~~ for consistency, but could be all::
$(TARGETS) with no rule in this case

%.com: %.abs %.patch                    ~~ generic rule to make .com from the
relocatable & a patch file
    $(ROOT)/tools/obj2bin $^ $@

tex21a.patch:                           ~~ tex21a.com doesn't need a patch, so
generate a dummy one on the fly
    @echo "; not needed" >$@

# intermediate files                    ~~ clean up of transient files
.INTERMEDIATE: $(TARGETS:.com=.rel) $(TARGETS:.com=.abs) tex21a.patch

STACK=100                               ~~ default is 100 byte stack
tex10.abs: STACK=60                     ~~ override for tex10.com as it only has
60 bytes

%.abs: %.rel                            ~~ user defined rule to create absolute
OMF file located at 100h for CP/M
    $(call locate,$@,$^,code(100h) stacksize($(STACK)) purge)

tex10.rel: tex10.obj x0100.obj          ~~ the rules for each relocatable
    $(call link,$@,$^ $(plm80.lib))
.
.
.
```

Partial makefile from plm80_4.0

```
ROOT := ../..
SHARED := shared            ~~ plm has a shared source directory
PEXFILE :=$(SHARED)/plm.pex ~~ where the pex file is also kept
SRC := src                  ~~ define directories for src, lst and obj
LST := lst
OBJ := obj

include $(ROOT)/tools/isis.mk

# force shared folder to be on isis drive :f3:

export ISIS_F3 := $(SHARED)/ ~~ the .ipx files are in :f3: & :f2:
export ISIS_F2 := $(SRC)/    ~~ make sure the directories are mapped explicitly

PURGE := purge               ~~ if debug versions are required

TARGETS = plm80 plm80.ov0 plm80.ov1 plm80.ov2 plm80.ov3 plm80.ov4 plm80.ov5
plm80.ov6

LOCATEFLAGS:=SYMBOLS PUBLICS
PLMFLAGS:=
ASMFLAGS:=

objs = main.obj plma.obj plmb.obj memchk.obj movmem.obj fill.obj plmc.obj\
.
.
.

# the following require plm v3.1        ~~ certain objects require plm80 V3.1

$(call objdir,plm1b.obj plm2b.obj plm2g.obj): PLM80=3.1
~~ objdir used here. could have used addprefix $(OBJ)/


# add the extra place to look for source

vpath %.plm $(SHARED)    ~~ make sure implicit rules know to look in shared
folder
vpath %.asm $(SHARED)

# force make to reinspect extracted source files

~~ for complex build directory structures make sometimes misses the
~~ dependencies on the auto extracted files. Whilst creating a dependency on
~~ all source files would work in this case using $(MAKE) $(TARGETS) is simpler
all::
     $(MAKE) $(TARGETS)

.
.
.

# extra rule for clean

clean::                       ~~ extra rule over the default clean
```

```
    -rm -f $(SHARED)/*.ipx
```

Partial asm80_4.1 makefile

```
# path to root of build tree
ROOT:=../..
# path to build directories
SRC:=src
LST:=lst
OBJ:=obj

PEXFILE:=$(SRC)/asm80.pex

PROTECT := notes.txt            ~~ notes.txt will be treated as part of
distribution

include $(ROOT)/tools/isis.mk

# override default tools
PLM80 = 3.1                     ~~ asm80 built using plm v3.1
export ISIS_F3 = $(SRC)         ~~ include directory is :F3: so explicity define
it

TARGETS := asm80 asm80.ov0 asm80.ov1 asm80.ov2 asm80.ov3 asm80.ov4 asm80.ov5
asxref
.
.
.
~~ here make does need to be re-invoked as the complex auto generated files
~~ are not detected otherwise
# this forces make to reinspect the *.plx files
all::
    $(MAKE) $(TARGETS)
.
.
.
~~ simple rule to create executables & overlays in the current directory
~~ from relocatable versions in the $(OBJ) directory
## build rules to make the program and overlays
# Symbols for overlays 0,1,2 and 3 are used in the build of asm80 so the apps are
created by purging the symbols
%: $(OBJ)/% ; $(call rm-symbols,$@,$<)
.
.
.
~~ the build of asm80 has a number of files that build differently
~~ based on conditional compilation
~~ rather than list the mapping explicitly these rules generate
~~ intermediate plm files with a suffix of s, m, n, or b from the
~~ master *.plx files. These are then compiled
~~ here $(PLMPP) is used to pre-process the files
# these are special build rules to process the plx files
$(OBJ)/%m.obj: $(SRC)/%.plx
    $(PLMPP) -sMACRO -o $(SRC)/$*m.plm $<
    $(call plm80,$@,$(SRC)/$*m.plm)

$(OBJ)/%n.obj: $(SRC)/%.plx
```

```
    $(PLMPP) -o $(SRC)/$*n.plm $<
    $(call plm80,$@,$(SRC)/$*n.plm)

$(OBJ)/%s.obj: $(SRC)/%.plx
    $(PLMPP) -sSMALL -o $(SRC)/$*s.plm $<
    $(call plm80,$@,$(SRC)/$*s.plm)

$(OBJ)/%b.obj: $(SRC)/%.plx
    $(PLMPP) -sBIG -o $(SRC)/$*b.plm $<
    $(call plm80,$@,$(SRC)/$*b.plm)
```

Example makefile for testing files

```
# path to root of build tree

ROOT := ..

PLMOPT := DEBUG OPTIMIZE
ASMOPT :=
LOCATEOPT:= PUBLICS SYMBOLS
include $(ROOT)/tools/isis.mk

~~ method of compiling with the toolset specified on the command line
~~ e.g. make V31 file.obj - would set tools to v3.1 and then compile the file

# on command line if you don't want plm80 v4.0

# use make V31 target or make V30 target

.PHONY: all V31 V30
all::
     @echo usage: make [V30^|V31] target ..."

V31:
     $(eval PLM80=3.1)           ~~ set the toolset
     @echo plm80 V3.1 selected   ~~ confirm to user

V30:
     $(eval plm80=3.0)
     @echo plm80 V3.0 selected

~~ simple rule to generate an ISIS application based on a single object file
%: %.obj
     $(call link,$*.rel,$^ $(system.lib) $(plm80.lib))
     $(call locate,$@,$*.rel,purge)
     @rm $*.rel                          ~~ can't use .INTERMEDIATE so rm
manually

~~ simple rule to generate a cp/m application based on a single object file
%.com: %.obj
     $(call link,$*.rel,$^ $(plm80.lib))
     $(call locate,$*.abs,$*.rel,CODE(100h) purge)
     @rm $*.rel                          ~~ can't use .INTERMEDIATE so rm
manually
     $(ROOT)/tools/obj2bin $@ $*.abs
     @rm $*.abs                          ~~ can't use .INTERMEDIATE so rm
manually
```

Change log

11-Sep-2020

- Updated the use of ASM80X, to use a more limited wrapper for Intel asm80 files. This version only supports the long variable names.
- **ipath** and **ifile** updated to reflect restructuring of the itools directory tree.
- Modified partial tex makefile to reflect recent changes.
- Noted how toolbox handles the production of a file called clean. which clashes with the normal clean target.
- Added program invocation macros OBJBIN, HEXOBJ, PLM81 & PLM82.

3-Sep-2020

- Minor updates to include ITOOLS usage, add additional implicit rules and miscellaneous small changes
- Moved changelog to end of document

24-Oct-2019

- Modified to take account of changes to support parallel builds and combined obj2bin and patch

7-Jun-2018

- Added support for asmx (experimental asm80 wrapper) and asm48

3-Dec-2017 & 17-Dec-2017

- Cleaned up formatting
- Added TOC & internal links

17-May-2017 & 18-May-2017

- refined NOVERIFY. See below
- standardised on packed files having suffix _all.src

8-May-2017

- isis.mk no longer uses ISISTOOLS to determine the version of the tools to use. Instead the version number of a specific tool can be specified.
- A consequence of the above change is that the variable REF must be explicitly defined and V30, V31 and V40 are no longer defined. Also ASM is no longer used and the PLM80, LIB, LINK, LOCATE variables are used differently.
- Renamed macros plm and asm to plm80 and asm80 respectively in anticipation of supporting plm86 and asm86 at a later date.
- Added macro for fort80 and a default rule for file.f to file.obj
- Simple variables have been defined to reference the system and plm80 libs.
- Macros have been added to generate paths to the isis tools see ipath and ifile in the documentation below.
- Changed link listing file to have .lin extension as .lnk was treated as shortcut in windows

Mark Ogden 11-Sep-2020