PhxAss

COLLABORATORS

	TITLE : PhxAss		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 24, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

PhxAss

1.1 PhxAss V4.20 Documentation (21-Aug-95)

```
                   /_____ _____ _____  ____ _____
            \___ /  / /__ ___ _\ ._ __ /  __/___ _/
           _/ __/    /   / / /// /    /__  / / / /
           \  //__/__/__/__/__/__/_/__/___ /_/_/_/
            \/            Phantasm's
================================================================================
    P H X A S S   V 4 . x x      MC680x0 / 68851 / 6888x  Macro Assembler
================================================================================


                     Preface

                Command Line Parameters

                Modifications since V2.00

                 Programmer Information

                Modifications since V3.00

                   Assembler Errors

                Modifications since V4.00

                      Linker

                Bug fixes since V2.11

                     History

                Bug fixes since V3.00

                 Acknowledgements

                Bug fixes since V4.00

                      Bugs

                  Starting PhxAss
```

The Author's Address

1.2 **Preface**

PhxAss V4.xx is a highly optimizing macro assembler for Motorola's 680x0
CPUs, 6888x FPUs and 68851 MMU (of course, the 030, 040 and 060 MMUs are
also supported).

PhxAss V4.xx requires OS2.04 (V37) as a minimum and does no longer support
older operating systems! (Kick 1.x owners: Get PhxAss V3.97, it's the last
version running under Kick 1.x). 1 MB of RAM recommended.

PhxAss V4.xx is SHAREWARE and © copyright 1994,1995 by Frank Wille
(Phoenix of Phantasm). Commercial usage of this program, without a
written permission of the author, is strictly forbidden!

Most important features:
o Fast: 15000-30000 lines per minute with standard Amigas, 50000-350000
  with A4000/040.
o Resident.
o Symbolic and Source Level Debugging.
o Automatic generation of executables (if possible).
o Creates relocatable Amiga-DOS objects or absolute code (written into
  a file (raw or Motorola S-Format), into memory or directly onto disk).
o Small Code and Small Data support (also support for __MERGED sections).
o Listing file, Cross Reference Listing, Equates file.
o Complete floating point support: You can use complex floating point
  expressions, including float functions (sine, logarithm, square root,
  power, etc.), everywhere in your source, e.g. defining float EQUates or
  SETs.
o Switches for nine different optimizations.
o Locale symbols (xxx$ and .xxx type).
o Up to 36 macro parameters.
o Support for Motorola's old and new operand style (even in 68000 mode).
o locale.library usage (english, german, swedish, italian, danish, dutch
  and french).
o Nearly all directives of the most popular assemblers like Seka, DevPac or
  AS (Aztec) are supported. Example: INCBIN, INCDIR, CODE_C, REPT, RS,
  RSRESET, EQUR, REG, OFFSET, XDEF, XREF, PUBLIC, ...
o Further development and support is guaranteed for years, because I'll
  never change my system (Amiga forever!).
o Finally: Although Shareware, there are no disabled functions in PhxAss!

You will find four different versions of PhxAss in this distribution:

1. PhxAss: The standard 680x0,FPU,MMU macro assembler.

2. SmallPhxAss: This is a 68000 only version without floating point support.
    As a result the program is much smaller.

3. FreePhxAss: This program is FREEWARE! It is intended for developers of
    PD-Compilers, who want to include PhxAss in their compiler package.
    FreePhxAss is *very* limited: No floating point expressions. No lis-

ting file, equates file, includes, macros, conditional assembly and
many other directives have been removed. But its functional extent is
completely sufficient for source texts generated by a compiler. Do
with FreePhxAss what you want to, but it would be very nice if you
could mention my name in your project (in the docs, for example).

4. GigaPhxAss: Identical to PhxAss, but source codes are not limited to
   65535 lines. Quite useful for assembling Reassembler outputs.
   I recommend the PD-Reassembler IRA by Tim Rühsen (SiliconSurfer of
   Phantasm), which was written especially for use with PhxAss.

PhxAss is SHAREWARE. So if you like it, please send me 25 DM or 20$ to be-
come a registered user. In return you will get the latest update and the
right to ask for a new update whenever you want (provided that you send me
a disk). If your domain allows large uuencoded files, I can also send them
via email.

PhxAss Support BBS:

```
            NOT ONLY FUN ...          _____
 _L_____  ____I~    T     _____ _____
  T~                   / \ T    T    | ___/Y        / Y   T~       /        /
 T|     _____/   \|   |    | T   |      ___/   |   | _____/ ____   /
 H|      \  |   |      \_   |    l_j   |   _   T    |   |        \   |  \/
 E|   _ ___/  l   |   _    |          |   T   |   l   l__       /  _l__
   |    Y  |        l_  \    |          |  |   |         l_/     /      \
   |    | |        /  |\    |    ___   |    |   |        /     /        \
  l____| l_____/___j \___j    T I___j_____j_____/\_____/_____/
  [*]=====================l____j==========[PRECiSiON-ADVERTISEMENT]===[*]
```

```
            ...lEITUNG.1..:+49.o22o4.61630...hST.dUAL14.4oo...
            ...lEITUNG.2..:+49.o22o4.61499...ZYXEl.+.19.2oo...
            ...lEITUNG.3..:+49.o22o4.23856...ZYXEl...16.8oo...
            ...lEITUNG.4..:+49.o22o4.23955...CTK.....14.4oo...

                Support-Board:  AMIGA-SECTION/Shred's-Corner
```


PHANTASM BBS:

```
                    ___a$\mathrm{\mu}$$\mathrm{\mu}$æm*×*@@@@@@@@*×**æm$\mathrm{\ ←
                      mu}$w___
                __$\mathrm{\mu}$æ#@\textdegree{}"USRv.34:+410-584-619¯"\ ←
                   textdegree{}¶@*m$\mathrm{\mu}$___
            _aæ@\textdegree{}"       ZyX19.2:+410-584-914         ¯\textdegree{} ←
               ¶Ñm___
          _æ#P¯   USR:+410-566-901-2 iSDN1:+410-566-901-1 "¶æw_
       aØ\textdegree{}¯  __          _æ#m_        _$\mathrm{\mu}$_              _æ_ ←
              \ensuremath{\lnot}Øæ_
        æ*"   ,ÆØ#Ñw_      ØØ@¯5#     æØØ@#m _$\mathrm{\mu}$æÆØØØØØMwØ@¶# __æØØm@#_
      $\mathrm{\mu}$@" 100%\ensuremath{\lnot}¶_ \ensuremath{\lnot}¶ÑÑm_   ØÞ Í#    ←
         ¸Æ#  \ensuremath{\lnot}ØwØ@\textdegree{}¯¯¯¯¯¯\ensuremath{\lnot}¶F JØÑØ@\ ←
        textdegree{} _Ø\textdegree{} \ensuremath{\lnot}#_
     _d" LEGAL__Æ#  ___\ensuremath{\lnot}\textdegree{}#ÑmJØ  ÆQ$\mathrm{\mu}$_$\ ←
        mathrm{\mu}$ØP ,  #P ,$\mathrm{\mu}$*@@@@*ÑØ  Æ@" _æÆ\textdegree{}    ¶$\ ←
```

```
            mathrm{\mu}$
  _P        æØ@#ØÞ JØ#æ_ \ensuremath{\lnot}\textdegree{}#Ø  ØØMÑØ" $\mathrm{\mu}$Ø   ←
     0  ØF   _æØF  ___æ*"oTHER nET\ensuremath{\lnot}ð
 _P         \ensuremath{\lnot}g__\ensuremath{\lnot} *P\textdegree{}"¯___æ#  ØMw_   ←
    ¢ÑØÞ l  0E    ØM¯  æ@" &fULL aMInET\ensuremath{\lnot}ð
 Æ         æØ\textdegree{}"  _  *ÆØØZØF JØØP _$\mathrm{\mu}$__\ensuremath{\lnot}´   ←
    JK \ensuremath{\lnot}#K  \ensuremath{\lnot}#  #W_    mIRROR  \ensuremath{\ ←
    lnot}K
J´ 4nODES  \ensuremath{\lnot}qæ  J#æ$\mathrm{\mu}$_ \ensuremath{\lnot}\textdegree ←
   {}ÑØÞ ÍØK $\mathrm{\mu}$@_\ensuremath{\lnot}¶qç  Øm \ensuremath{\lnot}#m_ JØ  _ ←
   \textdegree{}#m_               ¶
Ø  sYSTEM _áØ  ØØ#P\textdegree{} __Æ  ÆL_dØØØ#mJ#  Ø@#_  ¶ÑwØF JØ*_ \textdegree ←
   {}#m_ oNLINE      L
#       _ÆØÑ$^2$  __$\mathrm{\mu}$$\mathrm{\mu}$æ@""" ¯¯¯¯ ___ÆP#_ 2Q_\ ←
   ensuremath{\lnot}q_\ensuremath{\lnot}¶Ø  ÆF \ensuremath{\lnot}*_ \textdegree{}# ←
   m_ sINCE'88 1
Q        ¶E___$\mathrm{\mu}$m@P"¯æØØMÑæ_æ@@@@¶¶P _æ#MØØÑ#mÆÑ##mæ"_æØm_*_ ã#  ←
             F
¶          \ensuremath{\lnot}¶\textdegree{}¯   \ensuremath{\lnot}¶L \ensuremath{\ ←
   lnot}¶#ØQ_   _æÆØP"¯___  \textdegree{}#  0E_æØ\textdegree{} 2@ \ensuremath{\ ←
   lnot}¶"        JF
\ensuremath{\lnot}L lOcal rAtes   __JØ _ \ensuremath{\lnot}¶ÑÑm__æØP"_æ*M\ ←
   textdegree{}ØL JL JØØ\textdegree{} aÆ"              Ø
 ¶_fRom hAmburg aÆÑØØ#  Ø#m_ \ensuremath{\lnot}\textdegree{}#ØP ,øP¯  _ØÞ J&  P ←
   _Æ" a3ooo/1.5giga  J´
  #_            \ensuremath{\lnot}#_ \ensuremath{\lnot}´ J#P\textdegree{}" __aK $\ ←
    mathrm{\mu}$#    _Ø@ _ØØ  Æ"   mULTIFACECARD ,K
   ¶_        Æ@\textdegree{}      +æMØØ#  ØÞ  _æØF _ØP _  0K   IsdN-mASTER $ ←
      \mathrm{\mu}$F
    ¶m_          \ensuremath{\lnot}*$\mathrm{\mu}$r  #m__ \ensuremath{\lnot}¶Ñ#  ←
       0#__æØ@´_dP _ø#L J#         _ø´
    \ensuremath{\lnot}¶w_          _ØÞ áØØ#P^ _Øh_ ¶#Ø\textdegree{} _æ@"_ø\ ←
       textdegree{} J&  0K      _æP
    \ensuremath{\lnot}¶w_    _æØØÑ  " __$\mathrm{\mu}$$\mathrm{\mu}$$*@\ ←
        textdegree{} \ensuremath{\lnot}q_ _aæ#__ø\textdegree{}    ¶L J#  ←
            _øP
     \ensuremath{\lnot}¶æw_ ¶#¯___$\mathrm{\mu}$m@P"¯         \ensuremath{\lnot ←
        }¶P" \ensuremath{\lnot}\textdegree{}        \ensuremath{\lnot}¶"  ←
           _æ#\textdegree{}
       \ensuremath{\lnot}¶æw2¶M\textdegree{}¯lINK tO tHE oNLY lIVING wITNESS! ←
          __$\mathrm{\mu}$Æ\textdegree{}¯
         \ensuremath{\lnot}\textdegree{}¶æw_____ ←
          __$\mathrm{\mu}$æ#P"
          \ensuremath{\lnot}\textdegree{}¶Ñæm$\mathrm{\mu}$  F-E-A-R  E- ←
             H-Q  $\mathrm{\mu}$$\mathrm{\mu}$æ*@¶\textdegree{}¯  [/] ←
             CliTroMax D-SigN
             -----------------------
```

1.3 Modifications since PhxAss V2.xx

```
            Register symbols (
            EQUR
            ) must be defined before they are used. This enables
a faster addressing mode recognition.
```

There are some new optimizations possible. The optimize-flags which can be
specified after -n (since V4.00: OPT) or after the
                OPT
                 directive have
completely changed (see
                Command line paramters
).

If you have enabled the
                near-code
                 model, all jumps which are
referencing external symbols are converted to PC-relative instead of long
branch.

The '*'-symbol contains the current address. For example a 'bra *+10' would
branch to the location 12 bytes behind the 'bra'-opcode.

New directives:
                FPU
                ,
                PMMU
                ,
                CODE_C
                ,
                CODE_F
                ,
                DATA_C
                ,
                DATA_F
                ,

                BSS_C
                ,
                BSS_F
                 and
                INCDIR
                .

The
                instructions
                 and
                addressing modes
                 of 68020-68060, 68851(PMMU)
and 6888x(FPCP) are completely supported. You can use Motorola's new
addressing mode style even in the 68000 mode (e.g. MOVE (4,A5),D0).

The new addressing mode recognition has no difficulties with parentheses
'()' instead of brackets '[]' to indicate a term. An operand like

 -([x|y]*z)+6([addr+2,A4,regxy*QSIZE],[outdisp+$10<<(1+3)]),((abc-xyz)+2,A3)

would cause no problems.

PhxAss enables floating-point numbers to be used with the 6888x (FPCP)
instructions. For example: fmove.d #3.1415926536,fp7
moves the double-precision number pi to the FPCP register seven.

1.4   Modifications since PhxAss V3.00

```
                  V3.10:
o PhxAss is able to optimize forward-branches, which are coming into their
  8/16-bit range by optimzation of the subsequent code. As a result, other
  forward-branches could come into range and are also optimized, and so
  on.

V3.30:
o Symbols which are preceded by a '.' will be regarded as
                  local symbols
                    too.
o A special version of PhxAss is available, which is not limited to the
  maximum number of 65535 lines.

V3.40:
o
                  Macro
                   parameters may contain 63 characters now.
o The
                  extended addressing
                   mode recognition accepts the register
  symbols ZD0-ZD7 and ZA0-ZA7 to specify a
                  suppressed register
                  .
o Two new
                  escape codes
                   available:
  \e = escape ($1b)  and  \c = control sequence introducer ($9b).

V3.42:
o
                  Float constants
                   may be replaced by hex-constants now.
o When branch-optimization is activated, no extension-checking takes place.
  The best-possible code will be generated.

V3.47
o New
                  optimization flag
                  : 'I' forces PhxAss to ignore a 'Too large distance'
  error.

V3.50
o
                  '@'
                   is allowed to be the first character of a symbol name, providing
  the second character is non-numeric.
o The
                  RORG
                   directive is implemented.
o Two Devpac-specific directives are also supported now:
```

                    RSRESET
                     and
                    RS.x
                     for faster reading of (Devpac) include files.
o The new option
                    '-c' (V4:CASE)
                     can be used to switch off the case-
  sensitivity.

V3.51
o
                    RSSET
                     was forgotten in V3.50
o New directives:
                    IDNT
                    , COMMENT,
                    SUBTTL
                    V3.55
o From now on
                    near-data
                     symbols can be accessed not only by Absolute
  Addressing but also by Address Register Indirect mode ((An) must be the
  correct near-data base register). This will make the assembling of your
  source much faster, because PhxAss has to do less optimizations. As a
  side effect
                    XREFs
                     will be interpreted correctly and must not be re-
  placed by
                    NREFs
                    .
o If no unit name is given (by
                    TTL or IDNT
                    ), PhxAss will use the name
  of the source code without extension as the default unit name.
o The
                    OFFSET
                     directive is supported.

V3.60
o PhxAss V3.60 is pure! You can use the CLI-command RESIDENT to add it to
  the resident list.

V3.70
o '.W' and '.L' displacement-extensions for explicitely activating the 68020
  base-displacement mode.

V3.71
o
                    PROCSTART/PROCEND
                     directives for compatibility with DICE-C.

V3.75
o Immediate values are checked for their correct size. For example a
  MOVE.B #$1234,D0 will lead to an error now.

V3.80
o New option

```
                    '-w' (V4:ERRORS)
                     to determine the maximum number of errors
  which will be displayed before a request.
o The addressing mode syntax is checked much sharper (e.g. former versions
  accepted "(SP)-").


V3.81
o
                DC.W / DC.L
                 strings must no longer be aligned (e.g. DC.L "x" ->
  $00000078).


V3.90
o PhxAss was completely localized using the "locale.library".
  Available languages (August '94): english, german, polish.
o Documentation converted to AmigaGuide format.


V3.92
o New option
                '-v' (V4:VERBOSE)
                 for displaying the names of all include
  files and macros, which are accessed during assembly.
o New directive
                ELSEIF
                 for DevPac compatibility.
o Protection flags for created files will are "rw-d" now.


V3.94
o The immediate value of BTST, BSET, BCLR and BCHG is checked for valid
  range (0..7 or 0..31).
o You will need to enable (s)pecial optimization, to remove a ZRn-index.
  I think, if somebody explicitly writes 'ZRn', he doesn't want it to be
  removed by (n)ormal optimization.


V3.95
o From now on, it is possible to shift distances! Example:
   move.w   #(label2-label1)>>1,d0
  Although this is the same as "(label2-label1)/2", division and multi-
  plication is not allowed on distances, use right- or left shift instead.
  You might find it useful to use e.g. "((label2-label1)>>1)-1" to
  initialize the counter for a DB<cc>-loop - but be careful! Addition and
  subtraction after a distance-shift is not really supported, although it
  seems to work in this special case, if the distance between label1 and
  label2 is even.
  The reason is, that the shift is always executed last, which means that
  the "-1" doesn't affect the result of the shifting but the result of the
  distance directly.


V4.00:
o Conversion to OS 2. New command line parsing, using ReadArgs(), and
  new argument names.
o Automatic generation of executable load files. You no longer need to start
  the linker, if your code doesn't make use of external references.
  This feature can be disabled by using the CLI parameter
                NOEXE
                .
o Source Level Debugging support! By using the CLI parameter
```

```
                    LINEDEBUG
                  ,
  PhxAss creates a Line Debug block for each section, which contains the
  addresses of each source code line.
o Extended operand buffer from 80 to 128 characters.
o Floating point symbols and constants can be used in
                    expressions
                      of any
  complexity (like integers) now. PhxAss supports five binary operators,
  +(plus), -(minus), *(mult.), /(division), ^(power), and six unary
  operators: SIN(sine), COS(cosine), TAN(tangent), EXP(exponent), LOG(nat.
  logarithm), SQR(square root).
o New directive
                    SET.x
                      for alterable floating point symbols.
o New directive
                    INT
                      for assigning a float expression to an integer
  SET-symbol.
o
                    REPT ... ENDR
                      directives, like with DevPac.
o Floating point symbols in a listing file are displayed as floats instead
  of hexadecimals.
o Float symbols do appear in an equates file.
o Two new standard optimizations (which I must have forgotten in former
  versions):
  1. move.l #0,An  ->  suba.l An,An    2. move.l #x,An  ->  move.w #x,An
o New Small Data Mode: By writing
                    NEAR A4,-2
                      only the sections which are
  named "__MERGED", will be regarded as small data sections (similar
  to SAS/C).
o 68060 instructions implemented! (except PLPA, because it was impossible
  for me to get its code).
```

1.5 Modifications since PhxAss V4.00

```
                    V4.01:
o The INCLUDE directive does no longer ignore a label in the same line.
o Implemented abbreviations 'I' for 'INCPATH' and 'H' for 'HEADINC'.


V4.05:
o Code Sections are padded with $4E71 (NOP) instead with $0000.
o PhxAss accepts the DevPac options 'C', 'D', 'L' and 'O'.
o "DS.L 0" corresponds to "CNOP 0,4", "DS.Q 0" corresponds to "CNOP 0,8",
  etc.. Previously it works only with "DS.W 0".


V4.10:
o Operand may contain blanks. Example: 'DC.B 1, 2, 3, 4'
o Operators may have the same priorities! Examples: '*' and '/' or '<<'
  and '>>'.
o INCDIR "" is allowed for compatibility reasons.
o Operands may contain up to 511 characters (127 before V4.10).
```

o New parsing routines accelerate PhxAss by 5% - 25% !
o Swedish catalog.

V4.12:
o New directive for compatibility:
                    DX
                    . Behaves the same like DS.
o The 68060 instructions
                    PLPAR and PLPAW
                     are implemented!
o When assembly fails, PhxAss quits with a return code of 20 instead 1.

V4.14:
o Implemented the
                    __RS
                     symbol.
o PhxAss checks for conflicts between macro names and directive or
  instruction names.
o "" and '' within a string are recognized as a single ' or " character.
o If a code section ends with a minimum of eight zero-bytes, no NOP-padding
  will be performed.
o Italian catalog.

V4.15:
o Improved the compatibility with old Seka-sources. The '=' (EQU) direc-
  tives and labels, terminated by a ':', must no longer be seperated
  from the rest of the line with blanks or TABs.
o Because of numerous requests, symbol names are allowed to contain
  dots ('.'). But beware of terminating your symbols with ".w" or ".l"!
o The new directive
                    SFORM
                     enables the output of Motorola S-Records
  in absolute mode.
o Danish catalog.

V4.16:
o Increased the number of possible macro parameters to 15. Parameter 10 to
  15 can be accessed by '\a' to '\f'.
o New CLI-Parameter "RC=ERRCODE/K/N" for defining the return code in case
  of an error.

V4.18:
o MOVEQ #x,Dn with 128<x<255 will force PhxAss to display a warning,
  which can be disabled by the new NOWARN/S option.
o Increased the number of possible macro parameters again. Now there are
  36 parameters possible! (\0 - \9 and \a - \z)
o Increased the buffer sizes from 512 to 1024 bytes.
o French catalog.

V4.19
o Relocatable symbols will be suffixed by a ' in the listing file.
o SECTION allows a numerical third parameter to select the type of the
  section (should be $40000000 or $80000000, of course!).
o Freeware-PhxAss assembles 68030-060, FPU and MMU code too.

V4.20
o Dutch catalog.

o
                EQUR
                 works with register symbols.
o New directive
                FREG
                . It's like
                REG
                , but for the FPU registers.
  So FMOVEMs without any registers, which could be generated by a compiler,
  can be removed by optimization.
o Support for PHXOPTIONS option files, which are created by PhxOpts from
  Johan Johanssen. PhxOpts offers a GUI for presetting the options of
  PhxAss. But of course, the command line options have still priority.
o PHXASSINC has moved to ENV:PhxAss/ like the global PHXOPTIONS.
o New directives
                SAVE
                 and
                RESTORE
                 to save name and type of the
  current section. Useful in macros.

1.6 Bug fixes since V2.11

o Some instructions had generated a wrong error, e.g. TRAP and STOP gene-
  rated 'Assembly aborted' instead of 'Out of range'.
o 'move.l #xxxx,-(a0)' produced an illegal opcode.
o If someone writes a program without first opening a section with CODE/
  CSEG, SECTION or an initial label, all labels got wrong values.
o In some cases the equates file let PhxAss crash.
o A XDEF for a symbol which was already defined in another section would
  add this symbol to the external-hunk of the section currently active.
o Jump to Branch optimization did not check the addressing mode of the
  JMP/JSR instruction. It simply optimized all modes.
o A long branch to the next instruction was incorrectly optimized to $6x00.
o B<cc>.B was not recognized as a short branch. PhxAss accepted only
  B<cc>.S.
o The CNOP directive had disabled all optimizing in its section.
o The 'Word at odd address'-error crashed PhxAss sometimes.
o INITNEAR was useless in the absolute mode.

1.7 Bug fixes since V3.00

V3.01 (03.03.93)
o The 68020 addressing-mode ([Rn]) was assembled with a wrong size in pass
  one.

V3.02 (20.04.93)
o TRACKDISK now really works.

V3.05 (30.05.93)
o The near-data range was incorrectly limited to 32k in object files.

o The formatted text-output should also work on OS2.xx/3.xx now.
o MOVE USP,An , MOVES and MOVEP produced incorrect code.
o GLOBAL and BSS destroyed the MSW-bits of the BSS-hunk type ($000003eb).

V3.10 (04.06.93)
o PhxAss didn't accept octal numbers (@xxx).

V3.11 (06.06.93)
o CNOP definitely bug-free (I hope).
o CMPI #x,(PC) (>=68020)

V3.12 (08.06.93)
o Width 32 was impossible for bitfields.

V3.15 (12.06.93)
o Fixed bug with MOVEM-optimization.

V3.20 (03.07.93)
o References on "\@"-labels behind another macro nesting were impossible.

V3.21 (05.07.93)
o "\@" only allowed 999 macro calls (now it's unlimited).

V3.22 (06.07.93)
o Some extended adressing modes had made some problems:
  ([..],Rn.s|*x,od) and ([PC.. got a wrong size in pass one,
  ([BaseDisp]) generated an error and (bd,An/PC,Xn) (where bd is outside
  of the normal 8-bit range) crashed PhxAss.

V3.25 (17.07.93)
o Fixed bug with MOVES.
o FETOXM1 was forgotten (in my Reference Manual too).

V3.26 (18.07.93)
o TAB-Codes within strings could not be expanded.

V3.30 (25.07.93)
o Fixed bug with extended addressing modes ([..],Rn.x/*y,od), ([pc],..
  and ([pc,Rn],..

V3.31 (28.07.93)
o INITNEAR was useless in small-data mode.

V3.40 (07.08.93)
o Include paths which are suffixed by a ':' (volume names) were not re-
  cognized.
o FMOVEM.L Dn,FPcr got four bytes more in pass one than in pass two.
o Starting with page 100 the listing file became unreadable.
o The 'Out of memory' error was useless, because PhxAss crashed in most
  cases.
o CPUSHL,CINVL,CPUSHP,CINVP didn't work.
o BTST Dn,#x was missing.

V3.42 (24.08.93)
o The new formard-branch optimization destroyed the CNOP-alignments, which
  are located between the branch-instruction and the branch-destination.

V3.46 (02.09.93)
o PhxAss tried to optimize "MOVEP (d16,An)" with d16=0 into "MOVEP (An)".
  This was a bug!

V3.50 (15.09.93)
o Fixed bug with the '*'-symbol, containing the address of the current line.
o ".local" was impossible with float symbols.

V3.57 (22.09.93)
o PTESTR/PTESTW (68030) ignored the fourth operand.

V3.58 (23.09.93)
o NARG was not zero for a macro call without arguments.
o INCLUDE/INCBIN without quotes caused an error.

V3.61 (02.10.93)
o IFC '\1','' only behaved reliable if \1 was not used before.

V3.64 (24.11.93)
o The 16/32-bit displacements in the PC Indirect with Index addressing mode
  were wrong (+2 Bytes).

V3.65 (10.12.93)
o Fixed bugs in AbsLong->AbsShort and Logical Shifts optimization.

V3.70 (15.12.93)
o Fixed bug with (d16,An,ZRn) and (bd,PC).
o PhxAss tried to optimize CMPI #x,AbsLong always to PC-relative, which
  caused an 'Illegal Addressing Mode' error in 68000/010 mode.

V3.76 (07.04.94)
o Another bug in forward-branch optimizing (T-flag) made a mess with the
  object file in some specific cases.
o An illegal Bcc.B *+2 was converted into Bcc.W *+2 instead of Bcc.W *+4.

V3.77 (21.04.94)
o More than 13 sections in a file had lead to a crash or an infinite loop.

V3.78 (27.04.94)
o When PhxAss discovered an error in a line >= 32768 it didn't show neither
  the line-counter nor the incorrect line itself (now it works until 65000).

V3.79 (01.05.94)
o Absolute addressing with parentheses caused a Syntax Error.
  E.g.: "move.w label+(x+y)*z", but "move.w (x+y)*z+label" works.

V3.90 (16.09.94)
o Macro arguments which contained a comma (e.g. (d,An)) were unusable.

V3.93 (25.09.94)
o An explicit B<cc>.L was calculated 2 bytes too short in 020+ mode and 2
  bytes too far in normal mode.

V3.94 (09.10.94)
o The code generated by MOVE.B #-1,d0 (also: cmp, and, or, eor, etc.) was
  $103C $FFFF, but the bits 8-15 of the first extension word are reserved,
  for a byte-instruction! Now PhxAss politely generates: $103C $00FF.

o PhxAss changed (d,PC,ZRn) into (d,ZPC,ZRn).

V3.96 (23.10.94)
o PhxAss crashed when the macro nesting depth exceeded 8.

V3.97 (01.11.94)
o Distances, which are calculated by using a label directly behind a CNOP,
  were sometimes wrong.
o Because of some speed-improvements in V3.96, macro-arguments in opcode
  could lead to an error.

V4.00 (26.12.94)
o PhxAss crashed when a fixed number of include files were open.
o There were still some problems with removing empty sections.
o ELSEIF was documented, but not supported (forgotten).

1.8   Bug fixes since V4.00

                    V4.01 (07.01.95)
o Multiplication now has a lower priority than Division/Modulo to
  prevent situations like: 12/4*3 = 1
  I know, '*', '/' and '//' should have the same priorities, but
  currently my expression evaluation routines don't allow multiple
  operations at the same priority. Maybe I will completely rewrite
  them, in the future.

V4.05 (25.01.95)
o FreePhxAss didn't create an object file!!! :((((very ugly bug)
o The NOT operation (~) didn't pay attention to the operation size of
  the current instruction. So "move.b #~$80,d0" generated an error 97.
o An empty section, which was automatically removed during creation of
  an object file, deleted all XREFs of the subsequent sections.
o \" and \" made still some problems within strings.
o Macro parameters in the opcode field only worked, when using capital
  letters.

V4.10 (09.02.95)
o INCLUDE and INCBIN didn't work, if the file name contains blanks.

V4.11 (21.02.95)
o A numerical value within the command line, e.g. with "SMALLDATA x,y"
  crashes the system. It resulted from the massive changes in V4.10! :(

V4.14 (19.03.95)
o Beginning with error-message 89, the locale catalogs generated the
  wrong message.

V4.15 (30.04.95)
o If your source contains not a single byte, PhxAss sometimes crashed.

V4.16 (13.05.95)
o As a result of improvements in V4.15, the
                    OPT
                    -parameter didn't
  work. :(

V4.17 (18.05.95)
o MOVE16 didn't work with absolute addresses less than $8000.

V4.18 (07.06.95)
o #'"' and #"'" failed since V4.15.

V4.19 (27.06.95)
o Long range branches, which were explicitly defined as Byte branches,
  were made to Word instead Long. This caused an error in the following
  branch optimization.
o Distance shifts didn't work with equates (EQU).
o Distance shifts didn't get the correct value in the listing file.
o Forgot to close two math-libraries during clean-up.

V4.20 (12.08.95)
o Usage of scaling in 68000 mode will generate an error message now.
o ASL will be optimized to ADD by
                 (n)ormal optimization
                  (if
  possible).
o
                 \@
                  generates '_000' labels instead '000'. So you will be no longer
  forced to change you exec/types.i include file.
o It is possible again, to create an object file which contains only
  absolute
                 XDEFs
                 . It was already possible before v4.00, I think. ;)
o Now you will get an error message, when a symbol inside a macro was
  defined twice.
o When both Data and Bss sections were empty in small data mode, PhxAss
  generated an object file with 65535 size fields in the hunk header.

1.9 Starting PhxAss

                 PhxAss can be used from CLI only. You should copy it to "C:" or ↩
                  set a path
or link to its directory. If you know, you will need PhxAss very often, I
recommend to make it resident by typing "Resident C:PhxAss".

Format: PhxAss [FROM] <source file> [TO <output file>] [OPT <opt flags>]
          [EQU <equ file>] [LIST <list file>] [INCPATH {<include paths>}]
          [HEADINC {<include files>}] [PAGE=<n>] [ERRORS=<n>]
          [ERRCODE=<n>] [SMALLDATA <basReg>,[<sec>]] [SMALLCODE] [LARGE]
          [VERBOSE] [SYMDEBUG] [LINEDEBUG] [ALIGN] [case] [XREFS] [QUIET]
          [NOWARN] [SET "<symbol>[=<n>][,<symbol>...]"] [NOEXE]

Template: FROM/A,TO/K,OPT/K,EQU/K,LIST/K,I=INCPATH/K,H=HEADINC/K,PAGE/K/N,
          ERRORS/K/N,RC=ERRCODE/K/N,SD=SMALLDATA/K,SC=SMALLCODE/S,LARGE/S,
          VERBOSE/S,DS=SYMDEBUG/S,DL=LINEDEBUG/S,A=ALIGN/S,C=CASE/S,
          XREFS/S,Q=QUIET/S,NOWARN/S,SET/K,NOEXE/S

Starting PhxAss with no argument or with a single '?' will display a short

description. For a more precise description of all arguments, refer to

               Command line arguments
               .

When PhxAss is running, it can be stopped at any time by holding CTRL-C.

1.10 Command line arguments

            The standard version of PhxAss understands the following arguments ←
      :

```
FROM/A                 The only parameter, which is always required, is
[FROM] <source file>   the name of your source code file. If this name has
                       no extension, PhxAss automatically assumes ".asm"
                       for being the extension.
                       The source code must be an ASCII text file, where
                       each line is terminated by a linefeed ($0a)
                       character (the format, which all Amiga editors
                       should generate).
                       TAB-codes ($09) are allowed and completely sup-
                       ported.

TO/K                   Defines the name of the output file. If not speci-
TO <output file>       fied, PhxAss takes the source code's filename and
                       replaces its extension by ".o" . If PhxAss is able
                       to create an executable file instead of an object
                       module, the ".o"-extension will be removed.

EQU/K                  Generates an equates file. If the <file name> is
EQU <file name>        "*", the name of the source code with extension
                       ".equ" will be used. Since V4.00 equates files can
                       also contain floating point equates.

LIST/K                 Generates a listing file. If the <file name> is
LIST <file name>       "*",the name of the source code with extension
                       ".lst" will be used.

PAGE/K/N               Determines the page length for equates and listing
PAGE=<lines>           files. If <lines> equals zero, no formular feed
                       ($0c) characters will be generated.
                       The default value is 60 lines.

XREFS/S                Appends a reference list with all global symbols in
XREFS                  the listing file. If no listing file was opened,
                       this switch will cause an error.

I=INCPATH/K            Defines one or more include-paths which will be used
I <path1>[,<path2>,...] by the
                INCLUDE
                 and
                INCBIN

                                   directives. The paths, specified by  ←
                                      INCPATH/K, are
```

```
                              used directly after the path, specified by the
                              environment-variable PHXASSINC, has failed.
                              Important: If the path- or file names after INCPATH
                              or HEADINC contain blanks, you should embrace *all*
                              names with quotation marks and not only the one,
                              which contains blanks. Example:
                              INCPATH "include:,dh1:inc dir"


H=HEADINC/K                   Creates one or more INCLUDE directives at the top of
H <inc1>[,<inc2>,...]         your source code. See also: INCPATH.



DS=SYMDEBUG/S                 The names of all global labels of each section are
DS                            stored to symbol data blocks. A debugger can use
                              these names instead of addresses.


DL=LINEDEBUG/S                PhxAss creates a linedebug block, which can tell a
DL                            Source Level Debugger the right line in your source
                              code for any address. The location of your source is
                              stored in this block with a complete path, e.g.
                              "Work:Programs/Assembler/Tools/Source/Test.asm"
                              (this is, for example, not the case with SAS's
                              ASM :-).

SD=SMALLDATA/K                Forces all sections to use the small data model.
SD <basReg>,[<sec>]           <basReg> (default: 4 ) specifies the number of the
                              address register which will be used as pointer to
                              the small data section. Only the registers A2-A6 can
                              be used. <sec> is the number of the section which
                              will be your small data section (defaults to -2).
                              If <sec> is -1, all Data and Bss sections will be
                              treated as a whole small data section.
                              If <sec> is -2, only the sections which are named
                              "__MERGED", will be added to small data.


SC=SMALLCODE/S                Forces PhxAss to use the small code model. All JSR
SC                            and JMP instructions which are referencing external
                              (XREF) symbols are converted to PC-relative 16-bit
                              jumps.


LARGE/S                       Forces PhxAss to use the large code and large data
LARGE                         model in all sections.
                  NEAR
                   directives within
                              the source code will be ignored.


SET/K                         Predefines a symbol by
                  SET
                   directive.
SET "<symbol>[=<val>]"        Definition of multiple symbols must be seperated by
                              commas. <val> default to 1, when missing. Don't
                              forget the to embed the whole term, which follows
                              SET, in quotes (because of some problems with
                              ReadArgs()) !


A=ALIGN/S                     Enable auto-align for
                  DC.x
```

```
                          directives. All DC.W,
A                         DC.L, etc. directives in the code will be auto-
                          matically aligned to word-boundaries.


C=CASE/S                  Case-sensitivity off. All symbol names will be con-
C                         verted to upper case. This will slow down PhxAss
                          by 5%.


NOWARN/S                  Disables the output of warning messages.


ERRORS/K/N                Determines the maximum number of error-messages to
ERRORS=<max errors>       be displayed before a "continue?"-request. If
                          <max errors> is zero, PhxAss will never stop to
                          perform a request.


RC=ERRCODE/K/N            Defines the Shell return code in case of an error.
RC=<rc>                   <rc> defaults to 20.


VERBOSE/S                 Displays the names of all include files and macros,
VERBOSE                   which are accessed during assembly. This can be
                          helpful to locate errors with macros.


Q=QUIET/S                 Quiet mode. PhxAss makes no outputs until an error
Q                         occurs.


NOEXE/S                   PhxAss always tries to create an executable load
NOEXE                     file, instead of an object module, which requires
                          the additional use of a linker. NOEXE forces PhxAss
                          to create object modules in any case.


OPT/K                     Sets the optimize flags. The following characters,
OPT <flags>               without embedded blanks, can be specified after
                          'OPT':

                          0 (None)
                            No optimizing allowed. This flag should always
                            stand alone.

                          N (Normal)
                            Standard optimizations:
                            clr.l Dn          ->   moveq #0,Dn
                            move.l #x,Dn      ->   moveq #x,Dn
                            move.l #0,An      ->   suba.l An,An
                            move.l #xxxx,An   ->   move.w #xx,An
                            link.l(68020)     ->   link.w
                            adda/suba         ->   lea
                            ($xxxx).L         ->   ($xx).W
                            (0,An)            ->   (An)
                            asl #1,Dn         ->   add Dn,Dn
                            asl.w/b #2,Dn     ->   add Dn,Dn  add Dn,Dn

                          R (Relative)
                            ($xxxx)  ->  (xx,PC)

                          Q (Quick)
                            Conversions into addq/subq
```

```
                          B (Branches)
                            Bcc.l(020) -> Bcc.w -> Bcc.b, jmp/jsr -> bra/bsr

                          T (Total branch optimization)
                            Bcc.l(020) -> Bcc.w -> Bcc.b (forward branches)
                            Only active if 'B' is also selected.
                            WARNING! If you use this option together with a
                            listing file, then you can't rely on the line-
                            addresses in it.

                          L (Logical Shifts)
                            lsl #1,Dn  ->  add Dn,Dn
                            lsl.w/b #2,Dn  ->  add Dn,Dn + add Dn,Dn

                          P (PEA/LEA conversion)
                            move.l #x,An    -> lea x,An
                                            -> lea (x,PC),An / lea x.w,An
                            move.l #x,-(SP) -> pea x -> pea (x,PC) / pea x.w

                          S (Special optimizations)
                            pea 0                          ->  clr.l -(SP)
                            add/sub #0,An / lea (0,An),An  ->  (removed)
                            (d,An,ZRn)                     ->  (d,An) -> (An)
                            (d,PC,ZRn)                     ->  (d,PC)
                            The following are not recommendable for a MC68000
                            accessing hardware registers:
                            move #0,<ea>                   ->  clr <ea>
                            move.b #-1,<ea>                ->  st <ea>

                          M (MOVEM/FMOVEM)
                            movem Rn,<ea>  ->  move Rn,<ea>
                            movem ,<ea>    ->  (removed)
                            fmovem ,<ea>   ->  (removed)

                          I (Ignore too large distances)
                            Distances, which are currently out of range will
                            not cause an error. This is sometimes useful for
                            assembling reassembler-outputs or when you're
                            sure that all distances will come into range
                            again, by optimization of the subsequent code.
                            BE CAUTIOUS!!! If a distance has not come into
                            range, PhxAss creates faulty code!


                          There are two short cuts, which usually stand alone:

                          * Selects all standard optimizations & T
                            (OPT nrqbt).

                          ! Enables all optimizations possible
                            (OPT nrqbtlpsm).

                          If OPT is not specified the assembler uses standard
                          optimization (OPT nrqb).


The Freeware version of PhxAss does not support the following arguments:
```

EQU, LIST, XREFS, PAGE, INCPATH, HEADINC, VERBOSE, CASE

1.11 Programmer Information

Comments

Labels

M68000 Instructions

Expressions

Assembler Directives

Compiler Compatibility

PhxOpts

Environment Variables

1.12 Comments

Comments start with a ';' or with an '*'.

Example:

```
; Comment text
        moveq   #0,d0
** This is a comment too **
        nop                                     ; comment
        add.l   d0,d0                           * another comment
```

If no operand field is given, e.g. after the NOP instruction, the comment
must be preceded by a ';'. Examples:

```
        nop     * comment               -> Error!
        nop     comment                 -> Error!
```

1.13 Labels

Labels must start in the first column of a line. The colon after a label is
optional. You must insert a blank or TAB between label and opcode, if you
decide to omit the colon.

Example:
```
        Label:  moveq   #0,d0
```

Local labels have a '$' suffixed or are preceded by a '.' (since V3.30).
They are only valid between two global labels.

Example:
```
        Global1: add.w    d0,d1
                 beq.s    local1$
                 bpl.s    .local2
                 rts
        local1$: moveq    #-1,d0
        .local2: rts
        Global2:
```

The length for global and local labels is unlimited. Valid characters for
the labels are: 'a'-'z', 'A'-'Z', '0'-'9', '_'and '.' (since V4.15). The
first character may be an '@' (providing the second character is non-
numeric). Global labels cannot start with a digit.

The special '*'-symbol always contains the address of the current source
code line. This enables instructions like: bra *+4

CAUTION! Forward references with '*' will be corrected by PhxAss, but
backward references won't! I recommend to use labels, if you want to be
really safe.

1.14   Executable M68000 instructions

                       General Format

                 Standard Addressing Modes

                 Extended Addressing Modes

                 Suppressed Registers (020+)

                 M68000 Instruction Overview

1.15   General Format

A line of an assembler source text has the general format:

<label>   <opcode>   <operands>

PhxAss recognizes all operations found in Motorola's M68000PM/AD Program-
mer's Reference Manual and all of the common additions and short forms like
BHS instead of BCC, BLO instead of BCS, MOVE instead of MOVEA, ADD instead
of ADDI, etc. . In the current version all MC68000,68010,68020,68030,68040,
68060,68851,68881 and 68882 instructions are completely supported.

Labels must start at the first column. Opcodes (M68000 instructions or
assembler directives) and operands must have at least one preceding blank.

The operand field consists of one, two, three or four (68851) operands,
seperated by a comma. Imbedded blanks are allowed since V4.10.

1.16   M68000 Standard Addressing Modes

The notational conventions used in this section are:
```
   EA        - Effective address
   An        - Address register n
   Dn        - Data register n
   Xn.SIZE   - Denotes index register n (data or address) and
               index size (W for Word or L for Longword)
   PC        - Program counter
   dn        - Displacement value, n bits wide
   ( )       - Identify an indirect address in a register
```


```
 Data Register Direct
Syntax:           Dn
Generation:       EA = Dn
Extension Words:  0

 Address Register Direct
Syntax:           An
Generation:       EA = An
Extension Words:  0

 Address Register Indirect
Syntax:           (An)
Generation:       EA = (An)
Extension Words:  0

 Address Register Indirect with Postincrement
Syntax:           (An)+
Generation:       EA = (An), An = An + SIZE
Extension Words:  0

 Address Register Indirect with Predecrement
Syntax:           -(An)
Generation:       An = An - SIZE, EA = (An)
Extension Words:  0

 Address Register Indirect with Displacement (16-Bit)
Syntax:           (d16,An)  or  d16(An)
Generation:       EA = (An) + d16
Extension Words:  1

 Address Register Indirect with Index (8-Bit Displacement)
Syntax:           (d8,An,Xn.SIZE)  or  d8(An,Xn.SIZE)
Generation:       EA = (An) + (Xn) + d8
Extension Words:  1

 Program Counter Indirect with Displacement (16-Bit)
Syntax:           (d16,PC)  or  d16(PC)
Generation:       EA = (PC) + d16
```

```
Extension Words:  1

 Program Counter Indirect with Index (8-Bit Displacement)
Syntax:           (d8,PC,Xn.SIZE)  or  d8(PC,Xn.SIZE)
Generation:       EA = (PC) + (Xn) + d8
Extension Words:  1


 Absolute Short Addressing
Syntax:           (xxx).W  or  xxx.W
Generation:       EA given
Extension Words:  1


 Absolute Long Addressing
Syntax:           (xxx).L  or  xxx.L
Generation:       EA given
Extension Words:  2


 Immediate Data
Syntax:           #xxx
Generation:       Operand given
Extension Words:  1 or 2
```

1.17 68020+ Extended Addressing Modes

```
The notational conventions used in this section are:
   EA             - Effective address
   An             - Address register n
   Dn             - Data register n
   Xn.SIZE*SCALE  - Denotes index register n (data or address), the index
                    size (W or L), and a scale factor (1, 2, 4 or 8)
   PC             - Program counter
   dn             - Displacement value, n bits wide
   bd             - Base displacement
   od             - Outer displacement
   ( )            - Identify an indirect address in a register
   [ ]            - Identify an indirect address in memory


 Address Register Indirect with Index (extension of standard format)
Syntax:           (d8,An,Xn.SIZE*SCALE)
Generation:       EA = (An) + (Xn)*SCALE + d8
Extension Words:  1

 Address Register Indirect with Index and Base Displacement
Syntax:           (bd,An,Xn.SIZE*SCALE)
Generation:       EA = (An) + (Xn)*SCALE + bd
Extension Words:  1, 2 or 3

 Memory Indirect Postindexed
Syntax:           ([bd,An],Xn.SIZE*SCALE,od)
Generation:       EA = (bd + An) + Xn.SIZE*SCALE + od
Extension Words:  1, 2, 3, 4 or 5

 Memory Indirect Preindexed
Syntax:           ([bd,An,Xn.SIZE*SCALE],od)
```

```
Generation:        EA = (bd + An + Xn.SIZE*SCALE) + od
Extension Words:  1, 2, 3, 4 or 5


 Program Counter Indirect with Index (extension of standard format)
Syntax:           (d8,PC,Xn.SIZE*SCALE)
Generation:        EA = (PC) + (Xn)*SCALE + d8
Extension Words:  1


 Program Counter Indirect with Index and Base Displacement
Syntax:           (bd,PC,Xn.SIZE*SCALE)
Generation:        EA = (PC) + (Xn)*SCALE + bd
Extension Words:  1, 2 or 3


 Program Counter Memory Indirect Postindexed
Syntax:           ([bd,PC],Xn.SIZE*SCALE,od)
Generation:        EA = (bd + PC) + Xn.SIZE*SCALE + od
Extension Words:  1, 2, 3, 4 or 5


 Program Counter Memory Indirect Preindexed
Syntax:           ([bd,PC,Xn.SIZE*SCALE],od)
Generation:        EA = (bd + An + Xn.SIZE*SCALE) + od
Extension Words:  1, 2, 3, 4 or 5
```

The extended addressing modes have some ambiguities:
E.g. (0,A0) would be optimized to (A0) (one word), but maybe you want the
zero to be a 32-bit base displacement and you also want a suppressed D7
register. This instruction would have the same effect when executing, but it
consists of eight words instead of one.
Since PhxAss V3.70 you only have to write: (0.L,A0,ZD7)

1.18   68020+ Suppressed Registers

The Memory Indirect Post/Preindexed addessing modes allow the programmer to
suppress really everything. This means you may for example change the
standard format ([bd,An,Xn.SIZE*SCALE],od) into the following formats:
o   ([bd,An,Xn.SIZE*SCALE])
o   ([An,Xn.SIZE*SCALE],od)
o   ([bd],od)
o   ([An])
o   ([Xn.SIZE*SCALE])
o   ([An],od)
etc...

If you want to specify the number of a suppressed register you can use the
Zero-register symbols (ZRn, ZPC). By utilizing Zero-register symbols and the
displacement extensions .W and .L you should be able to generate any 68020
instruction bit-pattern you want (maybe helpful for reassemblers).
A .W/.L extension after the first displacement will tell PhxAss to switch
to base displacement mode and to disable optimizing for the current
instruction.

The suppressed registers are represented by the following symbols:
o suppressed data register D0-D7: ZD0-ZD7
o suppressed address register A0-A7: ZA0-ZA7

```
o suppressed PC: ZPC

It is impossible to EQUR suppressed registers!
```

1.19   M68000 Instructions supported by PhxAss

```
                    Integer Instructions for all processors

                Integer Instructions 010,020,030,040,060 only

                  Integer Instructions 020,030,040,060 only

                    Integer Instructions 040,060 only

                      Integer Instructions 68060 only

                        MOVEC Control Registers

                Floating Point Instructions 881,882,040,060

                  Floating Point Instructions 040,060 only

                        68851 PMMU Instructions

                        68030 PMMU Instructions

                       68040/060 PMMU Instructions
```

1.20   Integer Instructions (68000,68010,68020,68030,68040,68060)

```
ABCD         Dy,Dx                      Add Decimal with Extend
ABCD         -(Ay),-(Ax)
ADD.x        <ea>,Dn                    Add
ADD.x        Dn,<ea>
ADDA.x       <ea>,An                    Add Address
ADDI.x       #<data>,<ea>               Add Immediate
ADDQ.x       #<data>,<ea>               Add Quick
ADDX.x       Dy,Dx                      Add Extended
ADDX.x       -(Ay),-(Ax)
AND.x        <ea>,Dn                    And Logical
AND.x        Dn,<ea>
ANDI.x       #<data>,<ea>               And Immediate
ANDI.x       #<data>,CCR                And Immediate to Condition Codes
ANDI.x       #<data>,SR                 And Immediate to the Status Register
ASL/ASR.x    Dx,Dy                      Arithmetic Shift Left/Right
ASL/ASR.x    #<data>,Dy
ASL/ASR      <ea>
B<cc>.x      <label>                    Branch Conditionally
BCHG         Dn,<ea>                    Test a Bit and Change
BCHG         #<data>,<ea>
BCLR         Dn,<ea>                    Test a Bit and Clear
```

```
BCLR         #<data>,<ea>
BRA.x        <label>                      Branch Always
BSET         Dn,<ea>                      Test a Bit and Set
BSET         #<data>,<ea>
BSR.x        <label>                      Branch to Subroutine
BTST.x       Dn,<ea>                      Test a Bit
BTST.x       #<data>,<ea>
CHK.x        <ea>,Dn                      Check Register Against Bounds
CLR.x        <ea>                         Clear an Operand
CMP.x        <ea>,Dn                      Compare
CMPA.x       <ea>,An                      Compare Address
CMPI.x       #<data>,<ea>                 Compare Immediate
CMPM.x       (Ay)+,(Ax)+                  Compare Memory
DB<cc>       Dn,<label>                   Test Condition, Decrement, and Branch
DIVS         <ea>,Dn                      Signed Divide
DIVU         <ea>,Dn                      Unsigned Divide
EOR.x        Dn,<ea>                      Exclusive-OR Logical
EORI.x       #<data>,<ea>                 Exclusive-OR Immediate
EORI.x       #<data>,CCR                  Exclusive-OR Immediate to Cond. Codes
EORI.x       #<data>,SR                   Exclusive-OR Immediate to Status Reg.
EXG          Rn,Rm                        Exchange Registers
EXT.x        Dn                           Sign Extend
ILLEGAL                                   Take Illegal Instruction Trap
JMP          <ea>                         Jump
JSR          <ea>                         Jump to Subroutine
LEA          <ea>,An                      Load Effective Address
LINK         An,#<displacement>           Link and Allocate
LSL/LSR.x    Dx,Dy                        Logical Shift Left/Right
LSL/LSR.x    #<data>,Dy
LSL/LSR      <ea>
MOVE.x       <ea>,<ea>                    Move Data from Source to Destination
MOVEA.x      <ea>,An                      Move Address
MOVE         <ea>,CCR                     Move to Condition Codes
MOVE         <ea>,SR                      Move to the Status Register
MOVE         SR,<ea>                      Move from Status Register
MOVE         USP,An                       Move User Stack Pointer
MOVE         An,USP
MOVEM.x      <register list>,<ea>         Move Multiple Registers
MOVEM.x      <ea>,<register list>
MOVEP.x      Dx,(d,Ay)                    Move Peripheral Data (not 68060!)
MOVEP.x      (d,Ay),Dx
MOVEQ        #<data>,Dn                   Move Quick
MULS         <ea>,Dn                      Signed Multiply
MULU         <ea>,Dn                      Unsigned Multiply
NBCD         <ea>                         Negate Decimal with Extend
NEG.x        <ea>                         Negate
NEGX.x       <ea>                         Negate with Extend
NOP                                       No Operation
NOT.x        <ea>                         Logical Complement
OR.x         <ea>,Dn                      Inclusive-OR Logical
OR.x         Dn,<ea>
ORI.x        #<data>,<ea>                 Inclusive-OR Immediate
ORI.x        #<data>,CCR                  Inclusive-OR Immediate to Cond. Codes
PEA          <ea>                         Push Effective Address
RESET                                     Reset External Devices
ROL/ROR.x    Dx,Dy                        Rotate (without Extend) Left/Right
ROL/ROR.x    #<data>,Dy
```

```
ROL/ROR      <ea>
ROXL/ROXR.x  Dx,Dy                          Rotate Left/Right with Extend
ROXL/ROXR.x  #<data>,Dy
ROXL/ROXR    <ea>
RTE                                         Return from Exception
RTR                                         Return and Restore Condition Codes
RTS                                         Return from Subroutine
SBCD         Dx,Dy                          Subtract Decimal with Extend
SBCD         -(Ax),-(Ay)
S<cc>        <ea>                           Set According to Condition
STOP         #<data>                        Load Status Register and Stop
SUB.x        <ea>,Dn                        Subtract
SUB.x        Dn,<ea>
SUBA.x       <ea>,An                        Subtract Address
SUBI.x       #<data>,<ea>                   Subtract Immediate
SUBQ.x       #<data>,<ea>                   Subtract Quick
SUBX.x       Dx,Dy                          Subtract with Extend
SWAP         Dn                             Swap Register Halves
TAS          <ea>                           Test and Set an Operand
TRAP         #<vector>                      Take Trap Exception
TRAPV                                       Trap on Overflow
TST.x        <ea>                           Test an Operand
UNLK         An                             Unlink


Integer Condition Codes <cc>:
CC (HS)   carry clear (higher or same)   CS (LO)   carry set (lower)
EQ        equal                          F         never true
GE        greater or equal               GT        greater than
HI        higher                         LE        less or equal
LS        less or same                   LT        less than
MI        negative                       NE        not equal
PL        positive                       T         always true
VC        overflow clear                 VS        overflow set
```

1.21   Integer Instructions (68010,68020,68030,68040,68060)

```
BKPT         #<data>                        Breakpoint
MOVE         CCR,<ea>                       Move from the Condition Code Register
MOVEC        Rc,Rn                          Move Control Registers
MOVEC        Rn,Rc
MOVES        Rn,<ea>                        Move Address Space
MOVES        <ea>,Rn
RTD          #<displacement>                Return and Deallocate
```

1.22   Integer Instructions (68020,68030,68040,68060)

```
BFCHG        <ea>{offset:width}             Test Bit Field and Change
BFCLR        <ea>{offset:width}             Test Bit Field and Clear
BFEXTS       <ea>{offset:width},Dn          Extract Bit Field Signed
BFEXTU       <ea>{offset:width},Dn          Extract Bit Field Unsigned
BFFFO        <ea>{offset:width},Dn          Find First One in Bit Field
BFINS        Dn,<ea>{offset:width}          Insert Bit Field
```

```
BFSET       <ea>{offset:width}          Test Bit Field and Set
BFTST       <ea>{offset:width}          Test Bit Field
CALLM       #<data>,<ea>                Call Module (68020 ONLY!)
CAS.x       Dc,Du,<ea>                  Compare and Swap with Operand
CAS2.x      Dc1:Dc2,Du1:Du2,(Rn1):(Rn2) (020-040 only!)
CHK2.x      <ea>,Rn  (020-040 only!)    Check Register Against Bounds
CMP2.x      <ea>,Rn  (020-040 only!)    Compare Register Against Bounds
DIVS.L      <ea>,Dq                     Signed Divide
DIVS.L      <ea>,Dr:Dq
DIVSL.L     <ea>,Dr:Dq  (020-040 only!)
DIVU.L      <ea>,Dq                     Unsigned Divide
DIVU.L      <ea>,Dr:Dq
DIVUL.L     <ea>,Dr:Dq  (020-040 only!)
EXTB.L      Dn                          Sign Extend
LINK.L      An,#<displacement>          Link and Allocate
MULS.L      <ea>,Dl                     Signed Multiply
MULS.L      <ea>,Dh:Dl
MULU.L      <ea>,Dl                     Unsigned Multiply
MULU.L      <ea>,Dh:Dl
PACK        -(Ax),-(Ay),#<adjustment>   Pack BCD
PACK        Dx,Dy,#<adjustment>
RTM         Rn                          Return from Module (68020 ONLY!)
TRAP<cc>                                Trap on Condition
TRAP<cc>.x  #<data>
UNPK        -(Ax),-(Ay),#<adjustment>   Unpack BCD
UNPK        Dx,Dy,#<adjustment>
```

1.23   Integer Instructions (68040,68060)

```
CINVL       <caches>,(An)               Invalidate Cache Lines
CINVP       <caches>,(An)               (<caches> = DC, IC, BC or NC)
CINVA       <caches>
CPUSHL      <caches>,(An)               Push and Invalidate Cache Lines
CPUSHP      <caches>,(An)
CPUSHA      <caches>
MOVE16      (Ax)+,(Ay)+                 Move 16 Bytes Block
MOVE16      xxx.L,(An)
MOVE16      xxx.L,(An)+
MOVE16      (An),xxx.L
MOVE16      (An)+,xxx.L
```

1.24   Integer Instructions (68060)

```
LPSTOP      #<data>                     Low-Power Stop
```

Instructions that are not directly supported by the 68060, like
DIVUL, DIVSL, CAS2, CHK2, CMP2, MOVEP, will be assembled without
warning, because they are emulated by the "68060.library"
(I hope... :-).

1.25   MOVEC Control Registers (Rc)

```
                                          68010 68020 68030 68040 68060
SFC    Source Function Code                 x     x     x     x     x
DFC    Destination Function Code            x     x     x     x     x
USP    User Stack Pointer                   x     x     x     x     x
VBR    Vector Base Register                 x     x     x     x     x
CACR   Cache Control Register                     x     x     x     x
CAAR   Cache Address Register                     x     x
MSP    Master Stack Pointer                       x     x     x     x
ISP    Interrupt Stack Pointer                    x     x     x     x
TC     MMU Translation Control Register                       x     x
ITT0   Instr. Transparent Translation Reg. 0                  x     x
ITT1   Instr. Transparent Translation Reg. 1                  x     x
DTT0   Data Transparent Translation Reg. 0                    x     x
DTT1   Data Transparent Translation Reg. 1                    x     x
MMUSR  MMU Status Register                                    x     x
URP    User Root Pointer                                      x     x
SRP    Supervisor Root Pointer                                x     x
BUSCR  Bus Control Register                                         x
PCR    Processor Control Register                                   x
```

1.26   Floating Point Instructions (68881,68882,68040,68060)

Many of these instructions must be emulated for a 68040 or 68060,
but PhxAss will assemble them without any warnings.

68040 emulated instructions:
FACOS,FASIN,FATAN,FCOS,FCOSH,FETOX,FETOXM1,FGETEXP,FGETMAN,FINT,
FINTRZ,FLOG10,FLOG2,FLOGN,FLOGNP1,FMOD,FREM,FSGLDIV,FSGLMUL,FSIN,
FSINCOS,FSINH,FTAN,FTANH,FTENTOX,FTWOTOX

68060 emulated instructions:
FACOS,FASIN,FATAN,FCOS,FCOSH,FDB<cc>,FETOX,FETOXM1,FGETEXP,FGETMAN,
FLOG10,FLOG2,FLOGN,FLOGNP1,FMOD,FREM,FSGLDIV,FSGLMUL,FS<cc>,FSIN,
FSINCOS,FSINH,FTAN,FTANH,FTENTOX,FTWOTOX

```
Monadic operations:
Fxxxx       <ea>,FPn
Fxxxx       FPm,FPn
Fxxxx       FPn


FABS        Floating-Point Absolute value
FACOS       Arc Cosine
FASIN       Arc Sine
FATAN       Arc Tangent
FTANH       Hyberbolic Arc Tangent
FCOS        Cosine
FCOSH       Hyperbolic Cosine
FETOX       e to x
FETOXM1     e to x minus one
FGETEXP     Get Exponent
FGETMAN     Get Mantissa
```

```
FINT         Integer Part
FINTRZ       Integer Part, Round to Zero
FLOG10       log10
FLOG2        log2
FLOGN        loge
FLOGNP1      loge (x+1)
FNEG         Floating-Point Negate
FSIN         Sine
FSINH        Hyperbolic Sine
FSQRT        Floating-Point Square Root
FTAN         Tangent
FTANH        Hyperbolic Tangent
FTENTOX      10 to x
FTWOTOX      2 to x


Dyadic operations:
Fxxxx        <ea>,FPn
Fxxxx        FPm,FPn


FADD         Floating-Point Add
FCMP         Floating-Point Compare
FDIV         Floating-Point Divide
FMOD         Modulo Remainder
FMUL         Floating-Point Multiply
FREM         IEEE Remainder
FSCALE       Scale Exponent
FSGLDIV      Single Precision Divide
FSGLMUL      Single Precision Multiply
FSUB         Floating-Point Subtract


Special operations:
FB<cc>.x     <label>                Floating-Point Branch Conditionally
FDB<cc>      Dn,<label>             FP Test Cond., Decr., and Branch
FMOVE.x      <ea>,FPn               Move Floating-Point Data Register
FMOVE.x      FPm,<ea>
FMOVE.P      FPm,<ea>{Dn}
FMOVE.P      FPm,<ea>{#k}
FMOVE.L      <ea>,FPcr              Move F.-Point System Control Register
FMOVE.L      FPcr,<ea>              (FPcr = FPCR, FPSR or FPIAR)
FMOVECR      #ccc,FPn               Move Constant ROM
FMOVEM       <list>,<ea>            Move Multiple F.-Point Data Registers
FMOVEM       Dn,<ea>
FMOVEM       <ea>,<list>
FMOVEM       <ea>,Dn
FMOVEM.L     <list>,<ea>            Move Multiple F.-Point Control Regs.
FMOVEM.L     <ea>,<list>            (<list> = combin. of FPCR,FPSR,FPIAR)
FNOP                                No Operation
FRESTORE     <ea>                   Restore Internal Floating-Point State
FSAVE        <ea>                   Save Internal Floating-Point State
FS<cc>       <ea>                   Set According to Flt.-Point Condition
FSINCOS.x    <ea>,FPc:FPs           Simultaneous Sine and Cosine
FSINCOS      FPm,FPc:FPs
FTRAP<cc>                           Trap on Floating-Point Condition
FTRAP<cc>.x #<data>
FTST.x       <ea>                   Test Floating-Point Operand
FTST         FPm
```

```
Floating-Point Condition Codes <cc>:
F         false                            EQ        equal
OGT       ordered greater than             OGE       ordered gt. than or equal
OLT       ordered less than                OLE       ordered less than or equal
OGL       ordered greater or less than     OR        ordered
UN        unordered                        UNE       unordered or equal
UGT       unordered or greater than        UGE       unord. or gt. than or equal
ULT       unordered or less than           ULE       unord. or less than or equal
NE        not equal                        T         true
SF        signaling false                  SEQ       signaling equal
GT        greater than                     GE        greater than or equal
LT        less than                        LE        less than or equal
GL        greater than or less than        GLE       gt. or less than or equal
NGLE      not (gt. or less or equal)       NGL       not (greater or less than)
NLE       not (less than or equal)         NLT       not (less than)
NGE       not (greater than or equal)      NGT       not (greater than)
SNE       signaling not equal              ST        signaling true
```

1.27 Floating Point Instructions (68040,68060)

```
FSADD        Add Single Precision
FDADD        Add Double Precision
FSDIV        Single Precision Divide
FDDIV        Double Precision Divide
FSMOVE       Single Precision Move
FDMOVE       Double Precision Move
FSMUL        Single Precision Multiply
FDMUL        Double Precision Multiply
FSNEG        Single Precision Negate
FDNEG        Double Precision Negate
FSSQRT       Single Precision Square Root
FDSQRT       Double Precision Square Root
FSSUB        Subtract Single Precision
FDSUB        Subtract Double Precision
```

1.28 PMMU Instructions (68851)

```
PB<cc>.x    <label>                      Branch on PMMU Condition
PDB<cc>     Dn,<label>                   Test, Decr., and Branch on PMMU Cond.
PFLUSHA                                  Invalidate Entries in the ATC
PFLUSH      <fc>,#<mask>
PFLUSHS     <fc>,#<mask>
PFLUSH      <fc>,#<mask>,<ea>
PFLUSHS     <fc>,#<mask>,<ea>
PFLUSHR     <ea>                         Invalidate ATC and RPT Entries
PLOADR      <fc>,<ea>                    Load an Entry into the ATC
PLOADW      <fc>,<ea>
PMOVE       <PMMU Register>,<ea>         Move PMMU Register
PMOVE       <ea>,<PMMU Register>
PRESTORE    <ea>                         PMMU Restore Function
PSAVE       <ea>                         PMMU Save Function
PS<cc>      <ea>                         Set on PMMU Condition
```

```
PTESTR      <fc>,<ea>,#<level>               Get Information About Logical Address
PTESTR      <fc>,<ea>,#<level>,An
PTESTW      <fc>,<ea>,#<level>
PTESTW      <fc>,<ea>,#<level>,An
PTRAP<cc>                                    Trap on PMMU Condition
PTRAP<cc>.x #<data>


PMMU Condition Codes <cc>:
BS, BC  Bus Error
LS, LC  Limit Violation
SS, SC  Supervisor Only
AS, AC  Access Level Violation
WS, WC  Write Protected
IS, IC  Invalid Descriptor
GS, GC  Gate
CS, CC  Globally Sharable


PMMU Registers:
CRP, SRP, DRP, TC, BACx, BADx, AC, PSR, PCSR, CAL, VAL, SCC
```

1.29   PMMU Instructions (68030)

```
PFLUSHA                                      Flush Entry in the ATC
PFLUSH      <fc>,#<mask>
PFLUSH      <fc>,#<mask>,<ea>
PLOADR      <fc>,<ea>                         Load an Entry into the ATC
PLOADW      <fc>,<ea>
PMOVE       MRn,<ea>                          Move to/from MMU Registers
PMOVE       <ea>,MRn
PMOVEFD     <ea>,MRn
PTESTR      <fc>,<ea>,#<level>                Test a Logical Address
PTESTR      <fc>,<ea>,#<level>,An
PTESTW      <fc>,<ea>,#<level>
PTESTW      <fc>,<ea>,#<level>,An


PMMU Registers (MRn):
SRP, CRP, TC, MMUSR(PSR), TT0, TT1
```

1.30   PMMU Instructions (68040,68060)

```
PFLUSH      (An)                             Flush ATC Entries
PFLUSHN     (An)
PFLUSHA
PFLUSHAN
PTESTR      (An)                             Test a Logical Address
PTESTW      (An)


PLPAR       (An)                             Translate Logical to Physical
PLPAW       (An)                             (68060 only!)
```

1.31 Expressions

Expressions consist of symbols and constants. Symbols can be absolute,
relocatable or external. The arithmetic operations for INTEGER expressions,
supported by PhxAss, are (from highest to lowest precedence) :

```
1.  ~    not (unary)          -    negation (unary)
2.  <<   shift left           >>   shift right
3.  *    multiplication       /    division              //   modulo
4.  &    and                  |    or ('!' also allowed)  ^   exclusive or
5.  -    subtraction          +    addition
6.  ( )  parentheses  or [ ]  brackets
```

For absolute symbols and constants (which are absolute too), all arithmetic
operations are allowed.
If relocatables or externals occur in the expression, only subtraction and
addition is possible with some restrictions:

```
   reloc - abs          extern - abs          reloc - reloc
   reloc + abs          extern + abs          abs + reloc          abs + extern
   (reloc-reloc)<<abs  (reloc-reloc)>>abs  (V3.95 feature)
```

are defined, the others are illegal.

FLOAT expressions consist of floating point constants, absolute integer
constants and symbols. The following operations and functions are valid
for float expressions (V4.00 feature):

```
Binary:
   +    plus                 -    minus                *    multiplication
   /    division             ^    power
Unary:
   -    negation             sqr  square root          exp  e^x
   log  nat. logarithm       sin  sine                 cos  cosine
   tan  tangent
```

SQR, EXP, LOG, SIN, COS and TAN are functions and not case sensitive.
They are usually introducing a term, e.g. "sin(3.14159)". But if, as in
the last example, the term only consists of a single constant, it is also
allowed to write "sin:3.14159". The ':' is required to separate the
function name from a possible symbol name.

There are six types of constants:

Hexadecimal, preceded by a '$', consists of '0'-'9' and 'A'-'F' (or 'a'-'f')
Decimal, consists of '0'-'9'
Float, has the format [+/-][integer][.fraction][E[+/-]exponent]
Octal, preceded by a '@', consists of '0'-'7'
Binary, preceded by a '%', consists of '0' and '1'
String, embedded by ' or ", consists of one to four characters.

The character '\' is an escape-symbol, which can generate the following
codes:
```
   \\    the '\'-character itself
```

```
    \'      character #39 (single quote)
    \"      character #34 (quote)
    \0      character #0  (string terminator)
    \n      character #10 (line feed)
    \f      character #12 (formular feed)
    \b      character #8  (backspace)
    \t      character #9  (tabulator)
    \r      character #13 (carriage return)
    \e      character #27 (escape)
    \c      character #155(control sequence introducer)
```

"" and '' within a string will be replaced by " and ' (V4.14).

1.32 Assembler Directives

                The following paragraphs describe all directives that are ←
                    supported by
PhxAss. Important note! Directives must *not* start at the first column of
a line or they will be treated as labels! (note for Seka users :-)

Directives supported by PhxAss:

                  BLK
                Define Constant Block

                  BSS
                Bss section

                  BSS
                Allocate storage for Bss symbol

                 BSS_C
                Chip-RAM Bss section

                 BSS_F
                Fast-RAM Bss section

                 CNOP
                Align the following code

                 CODE
                Code section

                CODE_C
                Chip-RAM Code section

                CODE_F
                Fast-RAM Code section

                COMMENT
                Comment line

                 CSEG
                Code section

```
 DATA
Data section

DATA_C
Chip-RAM Data section

DATA_F
Fast-RAM Data section

  DC
Define Constant

  DCB
Define Constant Block

  DS
Define Storage

 DSEG
Data section

  DX
Define Storage

 ECHO
Print string

 ELSE
Define ELSE-part for conditional assembly

ELSEIF
Define ELSE-part for conditional assembly

  EQU
Assign expression to symbol

 EQU.x
Assign floating point expression to symbol

 EQUR
Assign register to symbol

  END
End of source text

 ENDC
End of conditional assembly

 ENDIF
End of conditional assembly

 ENDM
End of Macro definition

 ENDR
End of REPT loop
```

EVEN
Align the following code to an even address

FAIL
Abort assembly

FAR
Enter Far mode

FILE
Destination file for absolute code

FPU
Enable FPU code generation

FREG
Assign FPU register list to symbol

GLOBAL
Allocate storage for global Bss symbol

IDNT
Set unit name

IFC
Cond.Ass.: Compares two strings for equality

IFD
Cond.Ass.: Test if a symbol is defined

IFEQ
Cond.Ass.: Test if expression is zero

IFGT
Cond.Ass.: Test if expression is greater than zero

IFGE
Cond.Ass.: Test if exp. is greater or equal to zero

IFLT
Cond.Ass.: Test is exp. is less than zero

IFLE
Cond.Ass.: Test if exp. is less or equal to zero

IFNC
Cond.Ass.: Compares two strings for difference

IFND
Cond.Ass.: Test if a symbol is undefined

IFNE
Cond.Ass.: Test if expression is not zero

INCBIN
Include binary file

```
            INCDIR
            Set Include directory path

            INCLUDE
            Include another source file

            INITNEAR
            Initialize near mode base register

              INT
            Assign value of float expression to an integer SET-symbol

             LIST
            Next lines to listing file

             LOAD
            Destination address for absolute code

            MACHINE
            Set CPU type

             MACRO
            Macro definition

             MEXIT
            Exit Macro

             NEAR
            Enter Near mode

            NOLIST
            Next lines are invisible in listing file

             NREF
            Import Near-symbol

            OFFSET
            Start Offset section

              OPT
            Change optimization mode

              ORG
            Set absolute code origin

             PMMU
            Enable 68851 code generation

          PROCSTART
            Start of C-function for DICE-Compiler

            PROCEND
            End of C-function for DICE-Compiler

            PUBLIC
            Import/Export symbol
```

```
        REG
   Assign register list to symbol

    REPT
   Repeat lines between REPT and ENDR

   RESTORE
   Reactivate the last saved section

    RORG
   Set offset to start of section

     RS
   Assign value of RS-counter to symbol

   RSRESET
   Reset RS-counter

    RSSET
   Set RS-counter

    SAVE
   Saves the state of the current section

   SECTION
   Set section for following code

     SET
   Change value of SET-symbol

    SET.x
   Change value of floating point SET-symbol

    SFORM
   Creates Motorola S-Record format

   SUBTTL
    (no function)

     TTL
   Set unit name

 TRACKDISK
  Absolute code directly to disk

    XDEF
   Export symbol

    XREF
   Import symbol
```

```
The following directives are *not* supported by the Freeware version:
RSRESET,RSSET,RS,ECHO,LIST,NOLIST,INCDIR,INCLUDE,INCBIN,MACRO,ENDM,MEXIT,
RORG,OFFSET,ORG,FILE,LOAD,TRACKDISK,SFORM,COMMENT,SUBTTL,IF<cc>,ELSE,
ELSEIF,ENDC,ENDIF,REPT,ENDR,INT
```

1.33 EQU

```
    symbol    equ        <expression>
    symbol    =          <expression>
```

The expression will be assigned to the symbol.

1.34 EQU.x

```
                    symbol    equ.x     <float expression>
    symbol    =.x        <float expression>
```

An equate with extension .d,.f,.p,.s,.x will assign the value of a
floating point expression to the symbol. If you want to know more
about float expressions, refer to
          Expressions
              .
This is a special PhxAss directive.

1.35 EQUR

```
    symbol    equr       <register>
```

This directive assigns a register (D0-D7,A0-A7 or SP) or (since
v4.20) a register symbol to the specified symbol.
Since V3.00 a register symbol must be defined before it is used.

1.36 REG

```
                    symbol    reg       <register list>
```

This directive assigns the value of the register list to the symbol.
Valid register lists contain several register names (see
          EQUR
)
separated by the '/' character. The '-' character defines a range of
registers. The following are valid register lists:
    a1/a3-a5/d0/d2/d4
    d0-d7/a2-a6
    d1-3/d5-7/a0-1/a3-6   (since V3.56)

1.37 FREG

```
    symbol    freg       <register list>
```

The directive assign a FPU register list to the specified symbol.
Valid register lists contain the FPU registers FP0-FP7 seperated by

the '/' character. The '-' character defines a range of FPU regis-
ters. The following are valid FPU register lists:
   fp1/fp3-fp5/fp0
   fp0-fp7

1.38 SET

                    symbol   set   <absolute expression>

This directive assigns the value of the expression to the symbol. No
relocatables or externals are allowed within the expression. A symbol
defined by a SET directive may change its value by another SET.
There are some set-symbols which are defined by PhxAss:

PHXASS   set   1
VERSION   set   version<<16+revision

According to the connected processor and co-processor PhxAss will set
_MC68000_, _MC68010_, _MC68020_, _MC68030_, _MC68040_, _MC68060_,
_MC68881_ and _MC68882_.

NARG is zero outside a macro. Within a macro NARG is set to the number
of specified arguments.

__RS always reflects the current
          RS-counter
           value.

1.39 SET.x

    symbol   set.x   <float expression>

A SET with extension .d,.f,.p,.s,.x will assign the value of a
floating point expression to the symbol. You may change its value
by another SET, later in the source, provided that you don't change
its type (e.g. "symbol SET.S" followed by "symbol SET.D").
This is a special PhxAss directive.

1.40 INT

    symbol   int   <float expression>

The float expression will be evaluated and the result, without the
fractional part, will be assigned to an integer symbol.

1.41 RSRESET

This directive resets the internal RS-counter.

1.42 RSSET

```
            rsset     [<count>]
```

This directive sets the internal RS-counter to the <count> expression.

1.43 RS

```
                [symbol] rs.x      [<count>]
```

RS assigns the value of the internal RS-counter to the symbol, then it
increases the counter by the extension size multiplied with <count>.
If <count> is missing, it defaults to zero. For valid extensions refer
to the
        DC
         directive.
The current RS-counter value is reflected by the __RS symbol also.

1.44 IDNT

```
            idnt      <name>
            ttl       <name>
```

These directives set the name of the object file unit which the
assembler will generate. By default, it will be the name of the source
file without the extension.

1.45 SUBTTL

Source texts containing subttl will cause no error with PhxAss, but
for now it does completely nothing.
(To be honest, I've no idea what it should do! Please write me,
if somebody knows it.)

1.46 COMMENT

```
            comment  text
```

You may write any text you like behind this directive.

1.47 LIST

The following source code will be written to the listing file.

1.48 NOLIST

The following source code will not be written to the listing file.

1.49 OPT

```
                             opt       <optimize flags>
```

Changes optimization level. For a listing of all optimize flags, see

        Command line paramters
              .
This is a special PhxAss directive.

1.50 MACRO, ENDM

```
    symbol    macro
              ...text...
              endm

              macro     symbol
              ...text...
              endm
```

This directive assigns a macro to the specified symbol. The symbol
may appear on the left or right side of the directive. The text be-
tween the MACRO and ENDM directives will be inserted into the source
code when the assembler discovers this symbol. When invoking the
macro, up to 35 arguments, separated by a commas, can be specified
in the operand field. They are referenced in the macro text as '\1'
through '\9' and (since V4.18) as '\a' through '\z' for the last
26.
'\0' is reserved for the extension of the macro symbol. Example:

```
    bhs       macro
              bcc.\0    \1
              endm
```

This macro can be called by: bhs.s   label
".s" will be assigned to \0 and "label" will be assigned to \1.
A "\@" within the macro is replaced by text of the form "nnn",
where nnn is a unique three-digit number for each macro call.

Labels within a macro should consist of "\@", because defining
labels twice is illegal.

There is a limitation for using macro parameters in string con-
stants. Only \1 through \9 are allowed to avoid problems with
escape symbols.

1.51 MEXIT

Upon encountering this directive within a macro, the ↩
assembler scans
for the
ENDM
directive and leaves the macro.

1.52 END

In pass one the assembler ignores the rest of the source code and
starts pass two. In pass two the assembler closes all files and
terminates. By default the assembler terminates at end of file.

1.53 FAIL

The assembler displays the error "69 Assembly aborted !" and
terminates.

1.54 ECHO

echo   <string>

The assembler echoes the string. If <string> isn't specified, only a
newline is echoed.
This is a special PhxAss directive.

1.55 MACHINE

machine <processor-type>

This directive sets the processor-type for which the code will be
generated. Valid processor-types are:
68000, 68010, 68020, 68030, 68040, 68060
The implementation of this directive may be different in other
assemblers.

1.56 FPU

fpu [<cpID>]

This directive enables code generation for a MC68881/68882 coproces-
sor. By default the <cpID> is set to one, which should be the correct
ID for most systems using a floating point coprocessor.

Never set <cpID> to zero, because this is the constant ID for a PMMU.
If you have set the processor-type to 68040 or 68060 you should not
use this directive.
This is a special PhxAss directive.

1.57 PMMU

This directive enables code generation for a MC68851 Paged Memory
Management Unit. PMMU only makes sense if you have set the processor-
type to '68020'.
This is a special PhxAss directive.

1.58 SECTION

section <name>[,<type>[,<memflag>]]

The subsequent code will be placed in the section named <name>. There
are three section types: CODE, DATA and BSS. CODE contains the exe-
cutable M68000 instructions, DATA contains initialized data and BSS
contains uninitialized data (set to zero before the program is star-
ted). By default <type> is set to CODE. The section will be loaded to
the memory indicated by the <memflag> argument. This can be FAST or
CHIP. By default the section will be loaded to the memory with the
highest priority.

For compatibility reasons CODE_C, DATA_C and BSS_C are also recognized
as section type since V3.56.

Creating a section lets the assembler change into relocatable mode. In
this mode the following directives are illegal:

        org
        ,
        load
        ,
        file
        ,
        trackdisk
        .

1.59 CODE, CSEG

        These directives correspond to:
        section
          "CODE",code

1.60 DATA, DSEG

These directives correspond to:
section
  "DATA",data

1.61 CODE_C, CODE_F, DATA_C, DATA_F, BSS_C, BSS_F

See
CODE
,
DATA
 or
BSS
. In addition a memflag will be set, which
causes the section to be loaded to FAST (xxx_F) or to CHIP (xxx_C).

1.62 BSS

This directive corresponds to:
section
  "BSS",bss

1.63 BSS

bss   symbol,<size>

BSS with arguments does not start a section. It defines a symbol to be
in the
        BSS
        -section, reserves <size> bytes in this section and assigns
the address of the first byte to the symbol.
This directive is for Aztec-C compatibility only.

1.64 GLOBAL

global   symbol,<size>

This directive does the same as
        BSS
         symbol,<size>. In addition GLOBAL
will declare the symbol as
        XDEF
         (ext_def).
It is for Aztec-C compatibility only.

1.65 OFFSET

                              offset   [<start offset>]

This directive indicates the beginning of a special offset-section.
All the labels, which are declared in this section, will be treated
as absolute offsets instead of addresses. <start offset> defaults to
zero. This might be useful for declaring structure offsets with the

          DS.x
           directive. While writing programs for PhxAss you should
prefer the faster
          RSRESET
          ,
          RSSET
           and
          RS.x
                directives.
OFFSET was mainly implemented for compatibility reasons.

1.66 RORG

              rorg   <section offset>

This directive defines the offset of the subsequent code relative to
the start of the current section.

1.67 INCDIR

                              incdir   <path1>[,<path2>,...]

This directive does the same like the INCPATH argument (see

          Command line arguments
). Note that other assemblers don't accept
multiple paths.

1.68 INCLUDE

                              include <filename>

This directive causes PhxAss to suspend the assembling of the current
file and to assemble the file named <filename>. When done, the
assembler continues assembling the original file.
If PhxAss can't find the include file, it first searches in the
include directory defined by the environment variable PHXASSINC. Then
it searches in the include directories defined by INCPATH parameters
parameters (see
          Command line arguments

```
            ). At last, the
directories defined by
            INCDIR
             are searched.
```

1.69 INCBIN

```
                            incbin    <filename>
```

This directive causes the assembler to include a binary file into the
current section (e.g. graphics, samples or trigonometrical tables).
The assembler searches in the same include directories like
            INCLUDE
             .

1.70 XREF

```
              xref      symbol1[,symbol2,...]
```

This directive tells the assembler that the specified symbols are
externally defined and will be inserted by the linker.
Note that other assemblers may not support multiple symbols.

1.71 NREF

```
                              nref      symbol1[,symbol2,...]
```

This directive does the same like
            XREF
             , but the assembler is forced to
use these symbols as near-data relocatables.
This is a special PhxAss directive.

1.72 XDEF

```
              xdef      symbol1[,symbol2,...]
```

This directive causes the assembler to add the names and values of the
specified symbols to the external-block of the object file. The linker
can read the values of these symbols and insert them into other object
files.
Note that other assemblers may not support multiple symbols.

1.73 PUBLIC

                              public   symbol1[,symbol2,...]

When the specified symbols are defined in the current code, PUBLIC
will do the same like
         XDEF
         . When the symbols are unknown, PUBLIC
will do the same like
         XREF
         .
This directive is for Aztec-C compatibility only.

1.74 ORG

                              org   address

Defines the origin of the subsequent code and lets the assembler
change into absolute mode. Since V1.8 several ORG directives are
allowed and each one can be seen as a new section. The following
directives are illegal in absolute mode:

         ttl
         ,
         code
         ,
         cseg
         , {"section" link section},
         offset
         ,
         xref
         ,

         nref
         ,
         xdef
         ,
         public
         ,
         idnt
         .

1.75 LOAD

          load   address

After assembling is done, the absolute code will be loaded to this
address. By default the code will be loaded to the address which was
specified as origin. Be cautious with this directive, because the
destination memory will neither be checked nor allocated.
This is a special PhxAss directive (also known from SEKA).

1.76 FILE

```
          file      <filename>
```

After assembling is done, the absolute code will be written into the
file named <filename>.
This is a special PhxAss directive.

1.77 SFORM

```
          sform     <filename>
```

After assembling is done, the absolute code will be written as a
Motorola S-Record with the name <filename>.
You should always consider, that the S-Format supports only 24-Bit
addresses.
This is a special PhxAss directive.

1.78 TRACKDISK

```
          trackdisk <drive>,<startblock>[,<offset>]
```

After assembling is done, the absolute code will be written directly
to floppy disk using the 'trackdisk.device'. <drive> is valid from 0
to 3. <startblock> is valid from 0 to 1759 (or 3519, if you have a HD
drive). <offset>, which is zero by default, specifies the byte-offset
within a block and is valid from 0 to 511.
This is a special PhxAss directive.

1.79 NEAR

```
                       near      [An[,<secnum>]]
```

This directive initializes the parameters used by the near-data model.
NEAR with arguments may appear only once in your whole source code.
You shouldn't use 'NEAR An,0' before the first SECTION, CODE, DATA,
etc. directive.
After initializing the small-data model, it can be switched on and off
by NEAR and
          FAR
           without arguments. In this mode you are allowed
to access near-symbols via 'NearSymbol(An)'. Absolute references will
be automatically converted to Address Register Indirect, if possible.
The first argument, the address-register, is valid from A2 to A6 and
will be A4 by default. <secnum>, which defaults to -2, specifies the
number of the section which will be accessed by Address Register In-
direct mode.
If <secnum> is -1, all Data and Bss sections will be added to one
large small data section. Either PhxAss will do this job immediately,

when creating an executable file, or you must invoke your Linker with
the correct small data option.
If <secnum> is -2, only the Data or Bss sections which were named
"__MERGED", will be added to the small data section.

                    near   code

If the argument equals to the string "CODE" the assembler activates
the near-code model. This will force all absolute
            XREF
              jumps
into PC-relative mode.

Note that other assemblers don't accept parameters for NEAR.

1.80 FAR

This directive deactivates the near-code/data model when active.

1.81 INITNEAR

                 This directive inserts two M68000 instructions into the code ←-
                   which
will initialize the small-data model depending on the parameters set
by the
          NEAR
           directive. The assembler will generate this code
(10 bytes):
            lea   SmallDataBase,An
            lea   32766(An),An
This is a special PhxAss directive.

1.82 DC

    label   dc.?   <value>[,<value>,...]
    label   dc.b/w/l "string"[,...]

The DC (Define Constant) directive causes one or more fields of memory
to be allocated and initialized. Each field has the same size, speci-
fied by the extension of the directive. Each byte, word or longword
<value> can be an expression and may contain forward references.
The following extensions are valid:
.B   (1 byte)   Byte   .W   (2 bytes) Word
.L   (4 bytes)   Longword   .F   (4 bytes) Fast Flt. Point
.S   (4 bytes)   Single Precision .D   (8 bytes) Double Precision
.Q   (8 bytes)   Quadword(V3.42)   .X   (12 bytes) Ext. Precision
.P   (12 bytes) Packed BCD

Note that other assemblers may not support the floating-point and

quadword types.

1.83 DCB, BLK

```
                  label    dcb.x      <num>[,<fill>]
   label    blk.x     <num>[,<fill>]
```

These directives allocate a block of memory having <num> entries. The
available entry sizes are the same like with
          DC
          . The block will be
initialized with <fill>, which is zero when missing. For valid exten-
sions, refer to
          DC
          .

1.84 DS, DX

```
                  label    ds.x       <num>
   label    dx.x       <num>
```

This directive allocates a block of memory having <num> entries and
initializes each field with zero. See
          DCB
          ,
          BLK
          .

1.85 CNOP

```
            cnop      <offset>,<align>
```

This directive aligns the address of the following code to <align>.
Then the <offset> is added. Example: cnop 2,4   . This example
would align the next address two bytes behind the next longword
boundary. Note that an <align> larger than 8 makes no sense, if you're
creating relocatable code (see AllocMem(), exec.library).

1.86 EVEN

```
              This directive corresponds to
          cnop
           0,2  which will make the
address word-aligned.
```

1.87   IFcond, ELSEIF, ELSE, ENDIF, ENDC

```
These directives support conditional assembling. The general form of
the IF directive is:
            if<cond>     <expression> or symbol
            ...
            [else (or elseif)
            ...]
            endc (or endif)
PhxAss supports the following conditions:
   IFC "string1","string2" compares two strings. This is useful within
                           macros, when the strings contain macro-
                           arguments '\x' .
   IFD/IFND symbol         Tests if the symbol is defined (undefined).
   IFEQ/IFNE <exp>         Tests if <exp> is zero (not zero).
   IFGT/IFLT <exp>         Tests if <exp> is greater (less) than zero.
   IFGE/IFLE <exp>         Tests if <exp> is greater (less) than or
                           equal to zero.
```

1.88   PROCSTART,PROCEND

```
These directives are for compatibility with the DICE-C sytem. But cur-
rently they do nothing. For the future it should be possible to remove
LINK A5,#0 / UNLK A5  when A5 is not referenced between PROCSTART and
PROCEND.
```

1.89   REPT/ENDR

```
            rept       <count>
            ...
            endr
```

```
The part of source code, embedded by REPT/ENDR, will be assembled
<count> times. A negative <count> is illegal.
```

1.90   SAVE

```
                             save
```

```
Saves the name and type of the current section, so that it can be
reactivated by
         RESTORE
          later. Very useful in
macros. Example:

   print    macro
            save
            section  strings,data
   \@       dc.b     \1,0
```

```
                        restore
                        lea       \@,a0
                        bsr       printstring
                        endm
```

1.91 RESTORE

```
                              restore
```

Reactivates the section which was last saved by the
          SAVE
                    directive.

1.92 Compiler Compatibility

          A major reason for writing PhxAss was to create a program which ←
              can replace
the very slow AS-assembler of Aztec-C. There are many directives for Aztec-
compatibility, but since V3.30, where symbols preceded by a '.' are regarded
as local symbols, it is nearly impossible to assemble Aztec compiler out-
puts. The only solution is to write a program which translates all '.nnn'
symbols into '_nnn', for example.

Since introducing the new directives
              PROCSTART
               and
              PROCEND
               in V3.71,
DICE-C sources are completely supported.

1.93 PhxOpts

          PhxOpts from Johan Johanssen offers a GUI to predefine most ←
              options for
PhxAss and save them either as a global or local options file. So any of
your projects can get a local PHXOPTIONS file, which contains all options
you don't want to specify with every invocation of PhxAss. If no local
options file is present in your current directoy, PhxAss will look for the
global one in
              ENV:PhxAss
               .
PhxOpts requires the triton.library to work.

1.94 Environment Variables

PhxAss searches in ENV:PhxAss/ for environment variables. ←
Currently there

are two:

PHXASSINC   Specifies the path where to look for include files, if they
             are not located in the current directory.
             Note, that the location of PHXASSINC has changed in v4.20!
             Before it was ENV:PHXASSINC.

PHXOPTIONS   Contains several global predefined options, which will be
             passed to PhxAss when no local PHXOPTIONS in the current
             directory was found. PHXOPTIONS can be edited manually, but
             will be normally created by the
                 PhxOpts
                  GUI from Johan
             Johansson, which should be in your PhxAss archive.
             The options, which are passed to PhxAss via the command line,
             have priority and will overwrite those predefined ones.

1.95 Linker

You may use any linker which supports the standard Amiga DOS ←
object file
format. For example BLink, DLink, etc. - but I recommend that you use
PhxLnk, of course :-).

Since V4.00, you only need a linker when you have more than a single
module. PhxAss automatically generates an executable, if no external
references are present.
Two features of PhxLnk are not implemented in PhxAss:
1. Generation of HUNK_RELOC32SHORT blocks (16-bit offsets)
2. Removing zero-bytes at the end of a Code or Data section (so called
   Code-Bss or Data-Bss sections)
If you want to use one of these features (which require OS2.04 to run
your program), you should set the
               NOEXE
                switch and invoke PhxLnk.

1.96 Assembler Errors

In the current version of PhxAss the following errors could occur:

01 Out of memory

02 Unable to open utility.library

03 Can't open timer.device

04 DREL16 out of range
   Your Small Data area is too large. 64k is the limit for all data and

bss sections together.

07 HEADINC: file name expected
   Example: PhxAss HEADINC "dh0:file1,dh1:xdir/file2,"

08 IncDir path name expected
   Example: incdir "dir1","dir2",
   Caused also by INCPATH.

10 SMALLDATA: Illegal base register
   Allowed are 2-6 for A2-A6. A4 is standard.

11 MACHINE not supported
   The current version of PhxAss supports 68000, 68010, 68020,
   68030, 68040 and 68060.

12 File doesn't exist
   Unable to open your source code.

13 Missing include file name

14 Read error

15 String buffer overflow
   The length of a label, opcode or operand is limited to a length of
   128 characters.

16 Too many sections
   Maximum is 250 sections.

17 Symbol can't be made external
   XDEF can only be used on absolute or relocatable symbols.

18 Symbol was declared twice
   Only SET symbols can be declared more than once.

19 Unable to make XREF symbol
   A symbol, which is already defined in the current source code, can't
   be an XREF at the same time. Or: A symbol which is already declared as
   XREF can't be defined.

20 Illegal opcode extension
   Legal: .b .w .l .s .f .d .x .p .q

21 Illegal macro parameter
   Possible parameters are: \0 (opcode extension), \1 - \9, \a - \z
   and \@

22 Illegal characters in label
   Refer to
                Labels
                 in Programmer Information.

23 Unknown directive
   The opcode is neither a 680x0-mnemonic nor an assembler directive or
   macro.

24 Too many parameters for a macro
   35 parameters (\1 through \9 and \a through \z) are possible.

25 Can't open trackdisk.device

26 Argument buffer overflow
   Arguments are in most cases limited to 128 chracters.

27 Bad register list
   Valid register lists: d0-d3 d3-d4/a2 d2/d3/a4-a6 d7 a0/d2 d2-6/a0-4

28 Missing label
   The directive requires a label.
   Example: EQU <exp>   -> Error 28

29 Illegal seperator for a register list
   Valid seperators are '-' and '/'.

30 SET, MACRO, XDEF, XREF and PUBLIC are illegal for a local symbol

31 Not a register (try d0-d7 or a0-a7 or sp)

32 Too many ')'

33 Unknown addressing mode
   See
               Standard Addressing Modes
                and
               Extended Addressing Modes
                  for a complete description of all addressing modes.

34 Addressing mode not supported
   Example: move.b d0,a1 / move usp,d2 / clr.w (d3)+   -> Error 34

35 Can't use macro in operand
   Macros must be used as opcodes.

36 Undefined symbol

37 Missing register
   Example: mulu d0,   -> Error 37

38 Need data-register

39 Need address-register

40 Word at odd address
   Example: dc.b "Hallo"
             dc.w 0   -> Error 40
   Insert CNOP 0,2 or EVEN after string-constants.

41 Syntax error in operand

42 Relocatability error
   Example: move.l label(pc),d0 , where label is not a reloc. and/or
   label is not defined in the current section -> Error 42

```
43 Too large distance
   Example:  move.w  50000(a0),d0   -> Error 43
   Too large distance for a displacement by indirect addressing or branch.
   Short branches have a range of +126/-128 bytes. Long branches have a
   range of +32766/-32768 bytes.

44 Displacement expected
   Example:   label: move.l  label(a2),d1   -> Error 44

45 Valid address expected
   A program address was expected.

46 Missing argument

47 Need numeric symbol

48 Displacement outside of section
   Example:  bra  label   , where label is not defined in the current
   section -> Error 48

49 Only one distance allowed
   Expression can't contain several distances.
   Example:  move.l  #(label1-label2)+(label3-label4),d0   -> Error 49

50 Missing bracket/parenthesis

51 Expression stack overflow
   A maximum of 128 arguments are allowed in one expression.

52 Unable to negate an address

53 Can't use distance and reloc in the same expression
   Example:  move.l  #(label1-label2)+label3,d0   -> Error 53

54 Shift error (wrong type or negative count)
   Example: 1<<-1               -> Error 54
            label<<1            -> Error 54

55 Can't multiply an address
56 Overflow during multiplication
57 Can't divide an address
58 Division by zero
59 No logical operation allowed on addresses
60 Need two addresses to make a distance
61 Unable to sum addresses

62 Write error

63 Not a byte-, word- or long-string
   Example:  dc.d "XYZ"   -> Error 63

64 Can't subtract a XREF
   Valid operations with externals:  ext + abs , abs + ext  and  ext - abs

65 Impossible in absolute mode
   These directive can't be used in absolute mode:
   ttl, code, cseg, data, dseg, bss, section, xref, nref, xdef, public
```

66 Unknown error (fatal program failure)
   The assembler or its memory was corrupted by a faulty program running at
   the same time.

67 No externals in absolute mode
   See 65.

68 Out of range
   Example: addq.l #9,d1   -> Error 68

69 Assembly aborted
   Generated by the FAIL directive.

70 Missing ENDC/ENDIF

71 Missing macro name

72 Missing ENDM

73 Can't define macro within a macro

74 Unexpected ENDM

75 Unexpected ENDC/ENDIF

76 Impossible in relative mode
   These directive can't be used in relative mode: org, file, load, track-
   disk.

77 Parameter buffer overflow
   Macro parameters are limited to 63 characters.

78 Illegal REPT count
   The initial count for REPT should not be negative.

79 Unable to create file
   Maybe the destination disk is write-protected.

80 No reference list without a listing file
   XREFS switch was specified without the LIST switch.

81 No address allowed here
   Example: ds.l label   -> Error 81

82 Illegal characters in symbol
   See error 22.

83 Source code too large (max. 65535 lines)

84 No floating point without the appropriate math-libraries
   To use floating point symbols, you must have the following libraries
   in your LIBS: directory:
      mathtrans.library, mathieeedoubbas.library, mathieeedoubtrans.library

85 Overflow during float calculation
   This happens usually when converting the result of a float expression

```
    into a float type with lower precision, e.g. FFP or Single Precision.

86 Illegal symbol type in float expression
   Don't use relocatable symbols in float expressions.

89 Type of SET can't be changed
   Example: symbol   set.d   3.14159265
            symbol   set.x   -0.1          -> Error 89
   The value of SET is changeable, but not its type!

90 Can't mix LOAD, FILE and TRACKDISK
   Example:   load $70000
              file "mycode"   -> Error 90

91 Near mode not activated
   The near mode must be activated first, before using the INITNEAR
   directive.

92 Instruction not implemented in your machine
   The instruction exists for another processor, but not for your one. Use
   MACHINE to change processor type.

93 Illegal scale factor
   Example:   move.w  (a1,d2*3)    -> Error 93
   Valid scale factors are:  1, 2, 4 and 8

94 Missing operand
   Example:   move.l  (a0)+        -> Error 94

95 Section doesn't exist
   This error is caused by specifying an illegal section number in the NEAR
   directive.

96 Illegal RORG offset
   The relative offset must not specify an address before the actual PC.

97 Immediate operand size error
   Example:   move.b  #$1234,d0   -> Error 97

98 Missing ENDR
   Open repeat loop, when leaving the source code, an include file or macro.

99 Unexpected ENDR
   No matching REPT discovered.

100 REPT nesting depth exceeded
    The maximum nesting depth is 255.

101 Already a directive name
    You tried to define a macro, whose name is already used by a built-in
    directive or instruction.

102 SAVE nesting depth exceeded
    The maximum nesting depth is 8.

103 Unexpected RESTORE
    No matching SAVE discovered.
```

104 Missing RESTORE
    Missing a RESTORE when reaching the last line.

1.97   History / Literature

After six years of working with assemblers like SEKA, AS (Aztec-C) and A68k,
I decided in December 1991 that I need a new, powerful assembler. First,
I had the idea to buy O.M.A. or Devpac, but I don't like these modern
assemblers with an integrated editor. Other reasons for starting the de-
velopment of PhxAss were the chronic lack of money (I'm student) and the
possibility to create an assembler which will satisfy all of my demands.

I completed the first version V1.00 at the 28th of January in 1992. From now
on I used PhxAss to assemble itself (first I used A68k). It took me more
than a year and 23 versions to reach V3.00 and nearly another two years and
52 versions for V4.00 (of course PhxAss was not my only project in this
period).

Here is a list of my hardware and literature that made the development of
PhxAss possible:

Hardware: My good old A1000 (first version from '85) with 68010 CPU,
            2 MB Fast-RAM and a 33 MB Harddisk.
            (since December '93 also:) A4000, 68040, 10 MB RAM, 250 MB
            Harddisk.

Literature: Motorola MC68000/68008/68010/68HC000 8-/16-/32-Bit Micro-
            processor User's Manual (Prentice Hall)

            Motorola MC68020 32-Bit Microprocessor User's Manual (Prentice
            Hall)

            Motorola MC68040/68EC040/68LC040 Microprocessor User's Manual
            (Motorola)

            Motorola MC68881/882 Floating-Point Coprocessor User's Manual
            (Prentice Hall)

            Motorola MC68851 Paged Memory Management Unit User's Manual
            (Prentice Hall)

            Motorola M68000,MC68020,MC68030,MC68040,MC68851,MC68881/882
            Programmer's Reference Manual (Motorola)

            Amiga ROM Kernel Reference Manual: Libraries & Devices (Addison-
            Wesley)

            Amiga ROM Kernel Reference Manual: Includes & Autodocs (Addison-
            Wesley)

            Amiga Intern (Data Becker)

```
        Amiga Intern Band 2 (Data Becker)

        The Amiga Guru Book (Taunusstein)
```

1.98 Acknowledgements

Thanks to the following persons, who intensively tested PhxAss and accele-
rated its development by constructive bug-reports:

```
    Fabien Campagne (F)
    Tim Rühsen                   SiliconSurfer@Blackbox.shnet.org
    Wojciech Czyz (PL)
    Thomas Hagen Johansen (DK)   tjohansen@thj.adsp.sub.org
    Matthias Bock                Starfox@incubus.sub.org
    Christian Bauer              Cebix@ng-box.wwbnet.org
    Dave Dustin (NZ)             dave@eclipsnz.manawatu.gen.nz
    Richard Körber               Shred@tfh.dssd.sub.org
    Christian Wasner             Crisi@Blackbox.shnet.org
    Mark Knibbs (USA)            MARKK@msmail01.liffe.com
    David Neale (GB)             david@reeve.demon.co.uk
    Andy Church (USA)            achurch@binx.mbhs.edu
    Rainer Koschnick             r.koschnick@r-m-b.gun.de
    Volker Barthelmann           volker@vb.franken.de


The swedish catalogs were made by:
    Marcus Geelnard (S)          e4geeln@etek.chalmers.se

The italian catalogs were made by:
    Simone Tellini (I)           Simone.Tellini@f502.n332.z2.fidonet.org

The danish catalogs were made by:
    Morten Holm (DK)             mortenh@viking.roskildees.dk

The french catalogs were made by:
    Georges Goncalves (F)        kersten_emmrich@ramses.telesys-innov.fr

The dutch catalogs were made by:
    Leon Woestenberg (NL)        leon@stack.urc.tue.nl


Author of PhxOpts:
    Johan Johanssen (S)          Johan.Johannsen@f621.n205.z2.fidonet.org


Another acknowledgement, although gone bankrupt, is going to Commodore:

Thanks, for the only computer of the present time, which really makes
fun to work with :-)
```

1.99 Known bugs in version V4.20

```
                    o When instruction xxxx is completely removed by optimization,  ←
                         PhxAss will
  generate an illegal short branch with zero displacement:
     B<cc>.B  label
      xxxx
    label:
  This will only happen when you've set the optimize flag 'M', and xxxx is a
  MOVEM without registers, or when you have set the 'S'-flag, and xxxx is a
  'ADDA/SUBA #0,An' or 'LEA 0(An),An'.
```

o The Forward-Branch optimization (T-flag) doesn't correct the line-
  addresses in the listing file.

o Don't define labels directly before a CNOP directive!
  label1:
          CNOP   0,4
  label2:
  Unfortunately, PhxAss can only differentiate between label1 and label2
  in Pass 1. In Pass 2, it may happen that label1 is shifted too!
  Sorry, I see no solution... :((

If any bugs or questions occur, please write to :

                     My Address

1.100 My Address

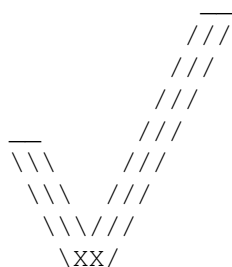SMail: Frank Wille
                          Auf dem Dreische 45
                          32049 Herford
                          GERMANY

EMail: frank@phoenix.owl.de
                          (polled daily)

                          fwille@techfak.uni-bielefeld.de
                          (currently polled one or two times per month -
                           valid until November '95 ??)


```
           __
          ///
         ///
        ///
  __    ///
 \\\   ///
  \\\ ///
   \\\///
    \XX/        A M I G A   F O R E V E R  !
```