PhxAss

COLLABORATORS			
	TITLE : PhxAss		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 9, 2024	

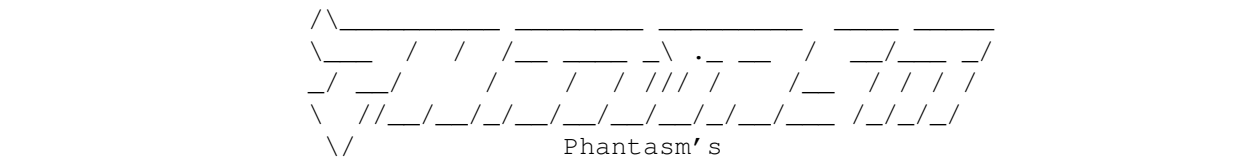
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

PhxAss

1.1 PhxAss V4.14 Anleitung (21.03.95)




```
==============================================================================
   P H X A S S   V 4 . x x      MC680x0 / 68851 / 6888x  Macro Assembler
==============================================================================
```


1.2 Einführung

PhxAss V4.xx ist ein hochoptimierender Macro Assembler für Motorolas 680x0 CPUs, 6888x FPUs und 68851 MMU (die 030er, 040er und 060er MMUs werden natürlich ebenfalls unterstützt).

PhxAss V4.xx benötigt OS2.04 (V37) als Minimalkonfiguration und wird ältere Betriebssysteme auch in Zukunft nicht mehr unterstützen! (Kick 1.x Besitzer sollten daher zu PhxAss V3.97 greifen, welches die letzte Version war, die noch unter 1.x läuft).

Die hervorstechendsten Features:
- Schnell: 15000-30000 Zeilen pro Minute auf Standard Amigas, 50000-350000 mit einem A4000/040.

o Resident.
o Symbolic und Source Level Debugging.
o Automatische Erzeugung von ausführbaren Programmen (wenn möglich).
o Unterstützung von Small Code und Small Data (auch '__MERGED' Sectionen).
o Listing File, Cross Reference Listing, Equates File.
o Vollständige Unterstützung von Fließkomma: Beliebig komplexe Fließkomma-
  ausdrücke, inklusive einiger Fließkomma Funktionen (Sinus, Logarithmus,
  Quadratwurzel, Potenzierung, etc.), können überall verwendet werden, z.B.
  zur Definition von Fließkomma EQUaten oder SETs.
o Schalter für neun verschiedene Optimierungen.
o Lokale Symbole (xxx$ und .xxx Typen).
o Unterstützung von Motorolas alter und neuer Operanden-Schreibweise (selbst
  im 68000 Modus).
o Nutzung der locale.library (Sprachen bisher: englisch, deutsch, schwe-
  disch, italienisch und polnisch(momentan nicht dabei)).
o Nahezu alle Direktiven der am meisten verbreitetsten Assembler, wie Seka,
  DevPac oder AS (Aztec), werden unterstützt. Beispiele: INCBIN, INCDIR,
  CODE_C, REPT, RS, RSRESET, EQUR, REG, OFFSET, XDEF, XREF, PUBLIC, ...
o Die Weiterentwicklung und Pflege dieses Programms ist auf Jahre hinaus ga-
  rantiert, da ich mich *niemals* vom Amiga abwenden werde (Amiga forever!).
o Zum Abschluß: Obwohl das Programm Shareware ist, wurde von mir absichtlich
  keine einzige Funktion deaktiviert.

Das PhxAss 4 Archiv sollte vier verschiedene Versionen enthalten:

1. PhxAss: Der standard 680x0,FPU,MMU Macro Assembler.

2. SmallPhxAss: Dies ist eine Spezialversion für den 68000 ohne Fließkomma
   Unterstützung. Viel kürzer als die standard Version.

3. FreePhxAss: Dieses Programm ist FREEWARE! Es es hauptsächlich für Ent-
   wickler von PD-Compilern gedacht, die PhxAss zum Übersetzen ihrer
   Compiler-Ausgaben verwenden möchten. Der Funktionsumfang ist *sehr*
   begrenzt: Keine Untersützung für 68030, 040, 060, FPU, MMU. Keine
   Macros oder bedingte Assemblierung. Viele Direktiven wurden ausgebaut.
   Trotzdem ist der Funktionsumfang für einen 68020 Compiler vollkommen
   ausreichend. Mit FreePhxAss können Sie machen was Sie wollen, es wäre
   allerdings nett, wenn Sie dann meinen Namen in Ihrem Projekt erwähnen.

4. GigaPhxAss: Identisch zur standard Version, ist aber nicht mehr auf maxi-
   mal 65535 Zeilen im Quellcode beschränkt und eignet sich daher hervor-
   ragend zum Übersetzen von Reassembler-Ausgaben.
   Ich empfehle den PD-Reassembler IRA von Tim Rühsen (SiliconSurfer/Phan-
   tasm), der speziell für die Benutzung mit PhxAss geschrieben wurde.

PhxAss ist SHAREWARE. Wenn Sie Gefallen an ihm gefunden haben, schicken Sie
mir doch bitte 25 DM um ein registrierter Benutzer zu werden. Als Gegenleis-
tung gibt es sofort das neueste Update, sowie das Recht jederzeit ein neues
Update von mir zu verlagen (vorausgesetzt Sie schicken mir eine Diskette).

Die beste Lösung ist natürlich, die jeweils neueste Version aus dem extra
dafür vorgesehenen

*** PHXASS SUPPORT BRETT ***

zu beziehen.

Box: SUiCiDE BBS in Bielefeld (Deutschland, Ostwestfalen-Lippe)
Port 1: +49-521-897178 (V.34)
Port 2: +49-521-896868 (V.FC)

Login: SAUGER
Passwort: <keins>

Momentan gibt es zwei Bretter: 'P-BIN' für die neuesten Updates und
'P-TEXT' für Fragen, Probleme, Bug-Reports, Verbesserungsvorschläge, etc.

1.3 Verbesserungen seit PhxAss V2.xx

Register Symbole (EQUR) müssen, bevor sie benutzt werden, erst einmal
definiert werden. Das ermöglicht eine schnellere Erkennung der
Adressierungsart.

Einige neue Optimierungen möglich. Die Optimierungs-Schalter, welche direkt
hinter -n (seit V4.00: OPT) oder nach der OPT Direktive angegeben werden,
haben sich komplett geändert (siehe CLI Parameter).

Wenn das near code Modell gewählt wurde, werden alle Sprünge auf externe
Symbole nach PC-Relativ konvertiert anstatt nach Long-Branch.

Das '*'-Symbol beinhalten die aktuelle Adresse. Zum Beispiel würde 'bra *+10'
auf eine 12 Bytes entfernte Position hinter dem 'bra'-Opcode verzweigen.

Neue Direktiven: FPU, PMMU, CODE_C\
, CODE_F, DATA_C, DATA_F,
BSS_C, BSS_F and INCDIR.

Die Instruktionen und Adressierungsarten\
vom 68020-68060, 68851(PMMU)
und 6888x(FPCP) werden vollständig unterstützt. Motorolas neue Schreibweise
für Adressierungsarten kann selbst im 68000 Modus eingesetzt werden (z.B.
MOVE (4,A5),D0).

Die neue Adressierungsarterkennung hat bei Verwendung von runden Klammern
'()' statt eckigen '[]', um einen Ausdruck zu klammern, keine Schwierig-
keiten mehr. Ein Operand, wie z.B.

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

sollte keinerlei Probleme verursachen.

PhxAss emöglicht die Benutzung von Fließkommazahlen in 6888x (FPCP) Instruk-
tionen. Zum Beispiel: fmove.d #3.1415926536,fp7
speichert die doppelt genaue Fließkommazahl Pi im FPCP Register sieben.

1.4 Verbesserungen seit PhxAss V3.00

V3.10:
o PhxAss kann Vorwärts-Branches optimieren, die erst durch Optimierung des nachfolgenden Codes optimierbar werden. Dadurch können weitere Vorwärts-Branches in Reichweite kommen, u.s.w.

V3.30:
o Symbole, die mit einem '.' begonnen werden, werden ebenfalls als Lokale Symbole betrachtet.
o Spezialversion von PhxAss (GigaPhxAss) ist verfügbar, die nicht mehr auf eine maximale Anzahl von 65535 Quelltextzeilen beschränkt ist.

V3.40:
o Macro Parameter dürfen nun 63 Zeichen beinhalten.
o Die 020+ Adressierungsarterkennung verarbeitet jetzt auch die Zero-Register ZD0-ZD7 und ZA0-ZA7 um ein unterdrücktes Register genauer zu bezeichnen.
o Zwei neue Escapce Codes verfügbar:
  \e = Escape ($1b) und \c = Control Sequence Introducer ($9b).

V3.42:
o Fließkomma Konstanten dürfen auch durch Hex-Konstanten ausgedrückt werden.
o Wenn Branch-Optimierung eingeschaltet ist, wird die Opcode-Extension nicht mehr geprüft. Es wird sowieso der bestmögliche Code generiert.

V3.47
o Neuer Optimierungs Schalter: 'I' zwingt PhxAss dazu einen 'Too large distance' Fehler zu ignorieren.

V3.50
o '@' darf als erstes Zeichen eines Symbols verwendet werden, unter der Voraussetzung, daß das zweite Zeichen keine Ziffer ist.
o Die RORG Direktive wurde implementiert.
o Untersützung zweier DevPac-spezifischen Direktiven:
  RSRESET und RS.x zum schnelleren Verarbeiten von (Devpac) Includes.
o Die neue Option '-c' (V4:CASE) kann dazu benutzt werden, die Beachtung der Groß/Kleinschreibung von Symbolen auszuschalten.

V3.51
o RSSET hatte ich in V3.50 vergessen.
o Neue Direktiven: IDNT, COMMENT, SUBTTL

V3.55
o Von nun an können near-data Symbole außer durch absolute Adressierung, auch im Address Register Indirect Modus (mit (An) als Basisregister) angesprochen werden. Als positiver Nebeneffekt läuft die Übersetzung solch eines Quelltextes sehr viel schneller ab, da PhxAss viel weniger optimieren muß. Zusätzlich werden dann XREFs korrekt interpretiert
 und müssen nicht durch NREFs ersetzt werden.
o Wenn ein Unit Name nicht explizit durch TTL oder IDNT angegeben wird, verwendet PhxAss den Namen des Quelltextes ohne seine Namenserweiterung.
o Die OFFSET Direktive wird unterstützt.

V3.60
o PhxAss V3.60 ist Pure! Er kann durch den CLI-Befehl RESIDENT zur Resident-Liste des Betriebssystems hinzugefügt werden.

V3.70
o '.W' and '.L' als Erweiterung an einem Displacement aktivieren automatisch
  den 68020 Base-Displacement Modus.

V3.71
o PROCSTART/PROCEND Direktiven für Kompatibilität mit DICE-C.

V3.75
o Bei Immediate-Werte wird genau geprüft, ob sie sich im richtigen Bereich
  befinden. Zum Beispiel würde ein "MOVE.B #$1234,D0" von nun an zu einer
  Fehlermeldung führen.

V3.80
o Neue Option '-w' (V4:ERRORS) um die maximale Anzahl an Fehlermeldungen
  zu bestimmen, die angezeigt werden bevor eine Abfrage erscheint.
o Die Syntax der Adressierungsarten wird viel genauer geprüft (z.B. akzep-
  tierten die Vorläuferversionen "(SP)-").

V3.81
o DC.W und DC.L Strings müssen nicht mehr genau auf Wort, bzw. Langwort-
  grenzen ausgerichtet sein (z.B. DC.L "x" -> $00000078).

V3.90
o PhxAss wurde durch Einsatz der "locale.library" lokalisiert. Verfügbare
  Sprachen (im August '94) sind: englisch, deutsch und polnisch.
o Die Anleitung wurde ins AmigaGuide Format konvertiert.

V3.92
o Neue Option '-v' (V4:VERBOSE) um die Namen aller Include-Dateien und
  Macros anzuzeigen, die während der Assemblierung aufgerufen werden.
o Neue Direktive ELSEIF für Kompatibilität mit DevPac.
o Die Protection-Flags der erzeugten Object-Datei werden auf "rw-d" gesetzt.

V3.94
o Bei BTST, BSET, BCLR and BCHG wird genau geprüft, ob sich die Immediate-
  Werte im Bereich von 0-7 oder 0-31 befinden.
o Jetzt muß erst (S)pecial Optimization eingeschaltet werden um einen ZRn-
  Index wegzuoptimieren, da ich der Meinung bin, daß ein ausdrückliches
  'ZRn' im Quelltext nicht schon bereits durch (N)ormal Optimization wieder
  verschwinden sollte.

V3.95
o Distanzen dürfen ge'shiftet' werden! Beispiel:
   move.w   #(label2-label1)>>1,d0
  Obwohl dies eigentlich dasselbe wie "(label2-label1)/2" ist, ist Division
  und Multiplikation für Distanzen nicht erlaubt. Statt dessen sollten
  Rechts- oder Linkverschiebungen verwendet werden.
  Oft ist es recht nützlich "((label2-label1)>>1)-1" zu schreiben, um z.B.
  den Zähler einer DB<cc>-Schleife zu initialisieren - aber Vorsicht!
  Addition und Subtraktion hinter einem Distanz-Shift wird nicht wirklich
  unterstützt, obwohl es in diesem Spezialfall, wenn die Distanz zwischen
  label1 und label2 durch zwei teilbar ist, zu funktionieren scheint.
  Der Grund ist, daß die Verschiebung grundsätzlich als letztes ausgeführt
  wird, was bedeutet daß die "-1" sich direkt auf die Distanz *vor* der
  Verschiebung auswirkt.

V4.00:

o Umstellung auf OS 2. Neues Commandline-Parsing mittels ReadArgs() und neue
  Namen für die möglichen Argumente.
o Automatische Erzeugung von ausführbaren Load Files. Wenn im Code keinerlei
  externe Referenzen vorkommen, wird kein Linker mehr benötigt.
  Dieses Feature kann durch NOEXE auch ausgeschaltet werden.
o Source Level Debugging wird unterstützt! Durch LINEDEBUG wird
  PhxAss dazu veranlaßt einen Linedebug Block zu für jede Section zu erzeu-
  gen, die zu jeder Zeile des Quelltexted die zugehörige Adresse enthält.
o Operandenpuffer von 80 auf 128 Zeichen erweitert.
o Fließkomma Symbole und Konstanten können ab jetzt in belibig komplizierten
  Ausdrücken verwendet werden (wie bei Integer). PhxAss untersützt fünf
  binäre Operatoren, +(plus), -(Minus), (*Mult.). /(Division), ^(Poten-
  zierung), und sechs unäre Operatoren: SIN(Sinus), COS(Cosinus),
  TAN(Tangens), EXP(Exponentialfkt.), LOG(nat. Logarithmus), SQR(Quadrat-
  wurzel).
o Neue Direktive SET.x für variable Fließkommasymbole.
o Neue Direktive INT um das Ergebnis eines Fließkommaausdrucks in ein
  Integer SET-Symbol zu überführen.
o REPT ... ENDR Direktiven, wie in DevPac.
o Fließkommasymbole in einem Listing File werden jetzt auch wirklich in
  Fließkommaschreibweise ausgegeben statt als Hexadezimalzahlen.
o Fließkommasymbole erscheinen auch im Equates File.
o Zwei neue Standardoptimierungen (die ich bisher wohl vergessen haben muß):
  1. move.l #0,An -> suba.l An,An   2. move.l #x,An -> move.w #x,An
o Neuer Small-Data Modus: Durch NEAR A4,-2 werden nur noch die Sectionen
  als Small-Data Sectionen angesehen, die den Namen "__MERGED" tragen
  (wie bei SAS/C).
o 68060 Instruktionen implementiert (bis auf PLPA, da es für mich bis heute
  leider nicht möglich war, den Code in Erfahrung zu bringen).

1.5 Verbesserungen seit PhxAss V4.00

V4.01:
o Die INCLUDE Direktive kann jetzt zusammen mit einem Label in einer
  Zeile stehen.
o 'INCPATH' darf durch 'I' und 'HEADINC' durch 'H' abgekürzt werden.

V4.05:
o Wenn Code-Sectionen nicht auf einer 32-Bit Grenze enden, wird das
  fehlende Word mit $4E71 (NOP) statt mit $0000 gefüllt.
o Die DevPac Optionen 'C', 'L', 'D' und 'O' werden verstanden.
o DS.L 0 kommt einem CNOP 0,4 gleich, DS.Q 0 einem CNOP 0,8, etc..
  Bisher funktionierte dies nur mit DS.W 0.

V4.10:
o Operand darf Leerzeichen enthalten. Beispiel: 'DC.B 1, 2, 3, 4'
o Endlich sind Operatoren mit gleicher Priorität möglich! Beispiele: '*'
  und '/' sowie '<<' und '>>'.
o INCDIR "" ist aus Kompatibilitätsgründen erlaubt.
o Ein Operand darf jetzt bis zu 511 Zeichen beinhalten (bisher 127).
o Neue Parsing-Routinen beschleunigen PhxAss um 5% - 25% !
o Schwedischer Catalog.

V4.12:
o Neue Direktive für Kompatibilität: DX. Verhält sich genauso wie DS.

o Die 68060 Instruktionen PLPAR und PLPAW sind implementiert!
o Wenn der Übersetzungsvorgang fehlschlug, gibt PhxAss einen Return-
  Code von 20 statt 1 zurück.

V4.14:
o Das __RS Symbol wurde implementiert.
o Namenskonflikte zwischen Macros und Direktiven sowie Instruktionen werden
  nun streng geprüft.
o "" und '' werden als einzelnes " bzw. ' in einem String erkannt.
o Wenn eine Code-Section auf mindestens acht Null-Bytes endet, wird von dem
  in V4.05 eingeführten NOP-padding abgesehen.
o Italienischer Catalog.

1.6 Beseitigte Bugs seit V2.11

o Einige Instruktionen erzeugten eine falsche Fehlermeldung, z.B. generier-
  ten TRAP und STOP 'Assembly aborted' anstatt 'Out of range'.
o 'move.l #xxxx,-(a0)' erzeugte einen illegalen Opcode.
o Wenn jemand ein Programm schreibt, ohne vorher eine Section mit CODE/CSEG,
  SECTION oder einem Label begonnen zu haben, erhielten alle Labels falsche
  Werte.
o Teilweise Abstürze beim Erzeugen des Equates Files.
o Ein XDEF mit einem Symbol, das schon in einer anderen Section definiert
  wurde zwang dieses in den Extern-Hunk der augenblicklich aktiven Section.
o Bei Jump und Branch Optimierungen wurde die Adressierungsart der JMP/JSR-
  Instruktion völlig außer Acht gelassen. Es wurden einfach alle Adressie-
  rungsarten optimiert.
o Long Branch auf eine direkt nachfolgende Adresse wurde fälschlicherweise
  zu $6x00 optimiert, was natürlich wieder ein Long Branch ist.
o B<cc>.B wurde nicht als Short Branch erkannt, sondern nur B<cc>.S.
o Die CNOP Direktive verbot sämtliche Optimierungen in ihrer Section.
o Die 'Word at odd address'-Fehlermeldung ließ PhxAss manchmal abstürzen.
o INITNEAR funktionierte im Absoluten Modus überhaupt nicht.

1.7 Beseitigte Bugs seit V3.00

V3.01 (03.03.93)
o Die 68020 Adressierungart ([Rn]) wurde in Pass 1 mit einem anderen
  Speicherbedarf übersetzt als in Pass 2. Folge: Alle nachfolgenden Labels
  verschoben.

V3.02 (20.04.93)
o TRACKDISK scheint jetzt endlich fehlerlos zu sein.

V3.05 (30.05.93)
o Die Adressierungsraum für Near-Data Symbole wurde im fertigen Object
  Module fälschlicherweise auf 32k begrenzt.
o Formatierte Textausgaben sollten jetzt auch unter OS2.xx/OS3.xx klappen.
o MOVE USP,An , MOVES und MOVEP erzeugten falschen Code.
o GLOBAL und BSS zerstörten die MSW-Bits im BSS Hunk-Typenfeld ($000003eb).

V3.10 (04.06.93)

```
o PhxAss wollte keine Oktalzahlen (@xxx) akzeptieren.

V3.11 (06.06.93)
o CNOP ist (hoffentlich) endgültig fehlerlos.
o CMPI #x,(PC) (>=68020)

V3.12 (08.06.93)
o Bitfeldbreite 32 war nicht möglich.

V3.15 (12.06.93)
o Fehler während MOVEM-Optimierung beseitigt.

V3.20 (03.07.93)
o Das "\@"-Symbol besaß nach einer weiteren Macro-Verschachtelung innerhalb
  eines Macros einen falschen Wert.

V3.21 (05.07.93)
o Bisher ließ "\@" nur 999 Macro-Aufrufe zu. Jetzt gibt es keine
  Limitierung mehr.

V3.22 (06.07.93)
o Probleme mit einigen erweiterten Adressierungsarten:
  ([..],Rn.s|*x,od) und ([PC..  wurden in Pass 1 mit einer anderen Größe
  übersetzt,  ([BaseDisp])  führte zu einer Fehlermeldung und  (bd,An/PC,Xn)
  (wobei bd mind. 16 Bit benötigt) führte zu einem Absturz.

V3.25 (17.07.93)
o Bug mit MOVES beseitigt.
o FETOXM1 habe ich völlig übersehen (mein Reference Manual allerdings auch).

V3.26 (18.07.93)
o TAB-Codes innerhalb von Strings wurden nicht richtig expandiert.

V3.30 (25.07.93)
o Weitere Fehler mit erweiterten Adressierungsarten ([..],Rn.x/*y,od),
  ([pc],.. und ([pc,Rn],..  beseitigt.

V3.31 (28.07.93)
o INITNEAR war im Small-Data Modus völlig nutzlos.

V3.40 (07.08.93)
o Include-Pfade die auf ':' endeten (z.B. Volume-Namen) wurden nicht
  erkannt.
o FMOVEM.L Dn,FPcr  war in Pass 1 vier Bytes länger als in Pass 2.
o Ab Seite 100 wurde das Listing File völlig unleserlich.
o Die 'Out of memory' Fehlermeldung war eigentlich überflüssig, da PhxAss
  bisher in solch einem Fall sowieso abstürzte.
o CPUSHL,CINVL,CPUSHP,CINVP funktionierten nicht.
o BTST Dn,#x hatte ich vergessen.

V3.42 (24.08.93)
o Die neue Vorwärts-Branch Optimierung zerstörte alle CNOP-Alignments, die
  sich zwischen der Branch-Instruktion und dem Sprungziel aufhielten.

V3.46 (02.09.93)
o PhxAss versuchte "MOVEP (d16,An)" mit d16=0 zu "MOVEP (An)" zu
  optimieren. Das war natürlich ein Bug!
```

V3.50 (15.09.93)
o Fehler mit dem '*'-Symbol, das die Adresse der aktuellen Zeile enthält, beseitigt.
o ".local" war für Fließkommasymbole nicht möglich.

V3.57 (22.09.93)
o PTESTR/PTESTW (68030) ignorierten ihren vierten Operanden.

V3.58 (23.09.93)
o NARG war bei einem Macro-Aufruf ohne Parameter nicht Null, wie es eigentlich hätte seien sollen.
o INCLUDE/INCBIN ohne Anführungszeichen führten zu einer Fehlermeldung.

V3.61 (02.10.93)
o Auf IFC '\1','' konnte man sich bisher nur verlassen, wenn \1 vorher noch nicht benutzt wurde.

V3.64 (24.11.93)
o Die 16/32-Bit Displacements in der Adressierungsart 'PC Indirect with Index' waren um zwei Bytes verschoben.

V3.65 (10.12.93)
o Fehler bei der AbsLong->AbsShort und Logical Shifts Optimierung.

V3.70 (15.12.93)
o Fehler (d16,An,ZRn) und (bd,PC) beseitigt.
o PhxAss versuchte 'CMPI #x,AbsLong' immer zu PC-Relativ zu optimieren, was im 68000/010 Modus natürlich illegal ist.

V3.76 (07.04.94)
o Ein weiterer Fehler in der Vorwärts-Branch Optimierung (T-Schalter) brachte die Object Datei in einigen Spezialfällen vollkommen durcheinander.
o Bcc.B *+2 wird jetzt endlich in Bcc.W *+4 umgewandelt (statt *+2).

V3.77 (21.04.94)
o Mehr als 13 Sectionen in einem Quelltext führten entweder zu einem Absturz oder einer Endlosschleife.

V3.78 (27.04.94)
o Wenn PhxAss einen Fehler in einer Zeile >= 32768 entdeckte, wurde weder die fehlerhafte Zeile noch deren Zeilennummer ausgegeben (jetzt klappt es bis 65000).

V3.79 (01.05.94)
o Absolute Adressierung mit runden Klammern führte zu Fehlern.
Z.B.: "move.w label+(x+y)*z", aber "move.w (x+y)*z+label" funktioniert.

V3.90 (16.09.94)
o Macro Argumente mit einem Komma (z.B. (d,An)) konnten nicht angewandt werden.

V3.93 (25.09.94)
o Ein ausdrücklicher B<cc>.L - Sprung wurde im 020+ Modus als zwei Bytes zu kurz und sonst als zwei Bytes zu weit bestimmt.

V3.94 (09.10.94)
o Der Code, der durch MOVE.B #-1,d0 (auch: cmp, and, or, eor, etc.) erzeugt wurde, war $103C $FFFF, obwohl die Bits 8-15 des ersten Extension-Words laut Motorola für eine Byte-Instruktion reserviert sind, also Null sein müssen. Jetzt erzeugt PhxAss $103C $00FF, wie es schon immer sein sollte.
o PhxAss änderte ein (d,PC,ZRn) in ein (d,ZPC,ZRn) um.

V3.96 (23.10.94)
o Wenn die Macro-Verschachtelungstiefe acht überstieg, gab es Abstürze.

V3.97 (01.11.94)
o Distanzen, die unter anderem aus einem Label bestanden der direkt nach einem CNOP folgte, waren manchmal falsch.
o Durch umgangreiche Geschwindigkeitsverbesserungen in 3.96 oder 3.95 konnten in diesen Versionen keine Macro-Argumente im Opcode benutzt werden.

V4.00 (26.12.94)
o Wenn eine bestimmte Anzahl von Include Dateien benötigt wurde stürzte PhxAss ab.
o Probleme beim automatischen Entfernen von leeren Sectionen.
o ELSEIF wird zwar in der Anleitung dokumentiert, war aber noch gar nicht eingebaut (wohl vergessen).

1.8 Beseitigt Bugs seit V4.00

V4.01 (07.01.95)
o Multiplikation hat jetzt eine niedrigere Priorität als Division/Modulo, um Situationen wie 12/4*3 = 1 zu verhindern.
Ich weiß natürlich, daß '*', '/' und '//' eigentlich dieselbe Priorität haben müßten, doch momentan erlauben meine Ausdruck-Auswertungsroutinen nur eine bestimmte Priorität pro Operator. Wahrscheinlich werde ich diese Routinen demnächst einmal völlig neu schreiben müssen.

V4.05 (25.01.95)
o FreePhxAss erzeugte gar keine Object-Files!!! :((((böser Fehler)
o Der NOT-Operator (~) wirkt sich nur noch in der Datenbreite der aktuellen Instruktion aus. Somit erhält man bei "move.b #~$80,d0" keinen Error 97 mehr.
o Leere Sectionen, die beim Erzeugen der Object-Datei entfernt werden, hatten sämtliche XREFs der nachfolgenden Sectionen gelöscht.
o \" und \' innerhalb von Strings funktionierten noch nicht richtig.
o Bisher konnte man nur dann Macro-Parameter im Opcode-Feld verwenden, wenn man Großbuchstaben benutzt.

V4.10 (09.02.95)
o INCLUDE und INCBIN funktionierten nicht, wenn der Dateiname Leerzeichen enthielt.

V4.11 (21.02.95)
o CLI-Parameter mit numerischen Werten, z.B. "SMALLDATA x,y" führten jedesmal zu einem Absturz. Dieser Fehler lag in den massiven Änderungen, die ich in V4.10 getätigt hatte, begründet.

V4.14 (19.03.95)

o Ab Fehler 89 waren die durch die Catalogs erzeugten Texte falsch.

1.9 PhxAss starten

PhxAss kann nur vom CLI aus genutzt werden. Am besten kopieren Sie ihn nach "C:" oder definieren wenigstens einen Pfad oder Link. Wenn PhxAss häufiger benutzt wird, kann es durchaus von Vorteil sein das Programm mittels "Resident C:PhxAss" resident im Speicher zu halten.
Aufruf:

```
Format:    PhxAss [FROM] <Quelltext> [TO <Zieldatei>] [OPT <opt Schalter>]
           [EQU <Equ File>] [LIST <List File>] [INCPATH {<Include Pfade>}]
           [HEADINC {<Incl. Dateien>}] [PAGE=<n>] [ERRORS=<n>]
           [SMALLDATA <basReg>,[<sec>]] [SMALLCODE] [LARGE] [VERBOSE]
           [SYMDEBUG] [LINEDEBUG] [ALIGN] [CASE] [XREFS] [QUIET]
           [SET "<symbol>[=<n>][,<symbol>...]"] [NOEXE]

Schablone: FROM/A,TO/K,OPT/K,EQU/K,LIST/K,I=INCPATH/K,H=HEADINC/K,PAGE/K/N,
           ERRORS/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,SET/K,NOEXE/S
```

Wenn PhxAss ohne Parameter oder nur mit einem '?' aufgerufen wird, wird eine Kurzbeschreibung der wichtigsten Argumente ausgegeben. Für eine genauere Beschreibung sollten Sie in CLI Parameter nachschlagen.

PhxAss kann durch halten der Tasten CTRL-C jederzeit unterbrochen werden.

1.10 CLI Parameter

Die Standard Version von PhxAss versteht die folgenden Argumente:

FROM/A [FROM] <Quelltext>	Der einzige Parameter der zwingend verlangt wird, ist der Name des zu Übersetzenden Quelltextes. Wenn der angegebene Name keine Erweiterung besitzt, hängt PhxAss automatisch ein ".asm" daran. Der Quelltext muß ein ASCII Text, bei dem jede Zeile durch ein Linefeed ($0a) abgeschlossen wird, sein (das Format, daß normalerweise alle Amiga Editoren erzeugen). TAB-Codes ($09) sind erlaubt und werden auch unterstützt.
TO/K TO <Zieldatei>	Bestimmt den Namen der Zieldatei. Wenn keiner angegeben wird nimmt PhxAss den Quelltextnamen und ersetzt seine Namenserweiterung durch ".o". Wenn es PhxAss sogar möglich ist, eine ausführbare Datei statt eines Object-Moduls zu erzeugen, wird das ".o" wieder gestrichen.
EQU/K EQU <Dateiname>	Erzeugt ein Equates File. Wenn der <Dateiname> "*" ist, so wird der Name des Quelltextes, allerdings mit ".equ" Namenserweiterung, verwendet. Seit V4.00

	erscheinen im Equates File auch Fließkommasymbole.
LIST/K LIST <Dateiname>	Erzeugt ein Listing File. Wenn der <Dateiname> "*" ist, so wird der Name des Quelltextes, allerdings mit ".lst" Namenserweiterung, verwendet.
PAGE/K/N PAGE=<Zeilen>	Bestimmt die Seitenlänge für Equates- und Listing-Files. Wenn <Zeilen> gleich Null ist, werden keine FF ($0c) Zeichen zum Seitenvorschub erzeugt. Der voreingestellte Wert ist 60 Zeilen.
XREFS/S XREFS	Hängt eine Referenztabelle aller im Quelltext benutzten globalen Symbole an das Listing File an. Wenn kein Listing File benutzt wurde, wird dieser Schalter zu einer Fehlermeldung führen.
I=INCPATH/K I <pfad1>[,<pfad2>,...]	Definiert ein oder mehrere Include-Pfade, welche dann von den Direktiven INCLUDE und INCBIN benutzt werden. Diese Pfade werden dann benutzt, wenn der Pfad der durch die Umgebungsvariable PHXASSINC gegeben ist, zu keinem Erfolg führt. Wichtig: Wenn die Pfad- oder Dateinamen hinter INCPATH oder HEADINC Leerzeichen enthalten, sollte man unbedingt *alle* Namen in Anführungszeichen einbetten, nicht nur den, der die Leerzeichen enthält. Beispiel: INCPATH "include:,dh1:inc dir"
H=HEADINC/K H <inc1>[,<inc2>,...]	Erzeugt eine oder mehrere INCLUDE Direktiven am Anfang des Quelltextes. Siehe auch: INCPATH.
DS=SYMDEBUG/S DS	Alle globalen Symbolnamen werden als Symbol Data Blocks der Zieldatei hinzugefügt. Ein Debugger kann diese Namen dann anstatt Adressen verwenden.
DL=LINEDEBUG/S DL	PhxAss erzeugt einen Linedebug Block, der einem Source Level Debugger für jede Adresse im Code die zugehörige Quelltextzeile nennt. Der Ort des Quelltextes wird in diesem Block mit komplettem Pfad gespeichert, z.B. "Work:Programs/Assembler/Tools/Source/Test.asm" (das ist zum Beispiel nicht der Fall bei SAS's ASM :-).
SD=SMALLDATA/K SD <basReg>,[<sec>]	Zwingt PhxAss dazu in allen Sectionen das Small Data Modell zu verwenden. <basReg> (normal: 4) bestimmt das Adressregister, welches ständig als Zeiger auf die Small Data Section verwendet werden soll. Erlaubt sind die Register A2-A6. <sec> ist die Nummer der Section die im Small Data Modus adressiert werden soll (voreingestellt ist -2). Bei <sec> = -1 werden alle Data und Bss Sectionen als eine große Small Data Section betrachtet. Bei <sec> = -2 werden nur die Sectionen mit Namen "__MERGED" zur Small Data Section hinzugenommen.
SC=SMALLCODE/S	Zwingt PhxAss das Small Code Modell zu vewenden, bei

SC
dem alle JMP und JSR Instruktionen, die auf externe (XREF) Symbole zugreifen, in PC-Relative 16-Bit Sprünge umgewandelt werden.

LARGE/S
LARGE
Zwingt die Benutzung des Larg Code und Large Data Modells für alle Sectionen. NEAR Direktiven im Quelltext werden dadurch ignoriert.

SET/K
SET "<Symbol>[=<Wert>]"
Ein Symbol wird mittels der SET Direktive vordefiniert. Wenn Sie mehrere Symbole definieren wollen, müssen Sie sie durch ein Komma voneinander trennen. Wenn <Wert> nicht angegeben ist, wird der voreingestellte Wert '1' verwendet.
Die SET-Definitionen müssen immer von zwei Anführungszeichen eingeschlossen sein (da ReadArgs() sonst Probleme bekommt)!

A=ALIGN/S
A
Schaltet die automatische Ausrichtung für DC.x Direktiven ein. Alle DC.W,DC.L,etc. werden dadurch auf Wortgrenzen ausgerichtet.

C=CASE/S
C
Schaltet die Unterscheidung von Groß/Kleinschreibung aus. Die Symbolnamen werden alle in Großbuchstaben umgewandelt. PhxAss wird dadurch etwa 5% langsamer.

ERRORS/K/N
ERRORS=<max errors>
Bestimmt die Anzahl der Fehlermeldungen, die bis zur nächsten "Fortfahren?"-Abfrage ausgegeben werden. Wenn man <max errors> auf Null setzt, macht PhxAss überhaupt gar keine Abfrage mahr.

VERBOSE/S
VERBOSE
Alle Include-Dateien und Macros, die während des Übersetzungsvorgangs aufgerufen werden, werden nach Verschachtelungstiefe eingerückt ausgegeben. Das kann z.B. recht hilfreich sein, wenn man Fehler mit Macros lokalisieren will.

Q=QUIET/S
Q
Durch diesen Schalter macht PhxAss keine Ausgaben, außer es kommt zu einem Fehler.

NOEXE/S
NOEXE
PhxAss versucht normalerweise immer ein ausführbares Load File, statt eines Object Moduls, welches danach noch einen Linker benötigen würde, zu erzeugen. Durch NOEXE wird PhxAss dazu gezwungen in jedem Fall ein Object Modul zu generieren.

OPT/K
OPT <flags>
Setzt die Optimierungs-Schalter. Folgende Schalter können, ohne daß sie durch Leerzeichen getrennt werden, hinter OPT angegeben werden:

0 (None)
Optimierung ist verboten. Dieser Schalter sollte nur einzeln auftreten.

N (Normal)
Standard Optimierungen:

```
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)

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

Q (Quick)
   Konvertierungen nach addq/subq

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

T (Totale Branch Optimierung)
   Bcc.l(020) -> Bcc.w -> Bcc.b (Vorwärts Branches)
   Ist nur aktiv wenn 'B' auch gewählt wurde.
   WARNUNG! Wenn diese Optimierung zusammen mit
   einem Listing File benutzt wird, ist auf die
   Zeilenadressen darin leider kein großer Verlaß
   mehr, da sich diese nachträglich alle verschie-
   ben können.

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

P (PEA/LEA Konvertierung)
   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 (Spezielle Optimierungen)
   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)
   Die folgenden sind für einen 68000, der direkt
   auf Hardwareregs. zugreift, nicht zu empfehlen:
   move #0,<ea>                    ->  clr <ea>
   move.b #-1,<ea>                 ->  st <ea>

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

I (Ignoriere zu große Distanzen)
   Distanzen, die sich nicht in Reichweite befinden
   erzeugen keinen Fehler. Das kann z.B. beim Über-
   setzen eines Reassembler-Outputs nützlich sein,
   oder wenn mach sicher ist, daß die Distanz durch
   Optimierung des nachfolgenden Codes wieder in den
   korrekten Bereich zurückkehrt.
   Aber VORSICHT!!! Wenn die Distanz *nicht* opti-
   miert werden kann, erzeugt PhxAss natürlich völ-
```

```
                    lig falschen Code.


                    Es gibt auch noch zwei Abkürzungen, die normaler-
                    weise nur einzeln auftreten sollten:

                    * Wählt alle Standard Optimierungen & T (OPT nrqbt).

                    ! Aktiviert alle Optimierungen (OPT nrqbtlpsm).

                    Wenn das Schlüsselwort OPT nicht angegeben wurde,
                    benutzt der Assembler die standard Optimierungen
                    OPT nrqb.

Die Freeware Version von PhxAss kennt die folgenden Argumente nicht:

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

1.11 Programmierer Info

```
                    Kommentare
                Sprungmarken (Labels)
                 M68000 Instruktionen
                    Ausdrücke
                 Assembler Direktiven
              Compiler Kompatibilität
```

1.12 Kommentare

```
Kommentare werden durch ein ';' oder ein '*' eingeleitet.

Beispiele:

; Kommentartext
        moveq   #0,d0
** Dies ist ebenfalls ein Kommentar **
        nop                             ; Kommentar
        add.l   d0,d0                   * und noch ein Kommentar


Wenn kein Operandenfeld existiert, z.B. nach einer NOP Instruktion, *muß*
der Kommentar durch ein ';' eingeleitet werden. Beispiele:

        nop     * comment               -> Fehler!
        nop     comment                 -> Fehler!
```

1.13 Sprungmarken (Labels)

Labels müssen in der ersten Spalte einer Zeile beginnen. Der Doppelpunkt hinter dem Label ist optional.

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

Lokale Labels werden entweder durch ein '$' abgeschlossen oder beginnen mit einem '.' (seit V3.30). Sie sind nur im Bereich zwischen zwei globalen Labels gültig.

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

Die Länge der globalen und lokalen Labels ist unbegrenzt. Es folgen die gültigen Zeichen für Labels: 'a'-'z', 'A'-'Z', '0'-'9' und '_' . Das erste Zeichen darf auch ein '.' oder '@' (unter der Vorraussetzung, daß das zweite Zeichen keine Ziffer ist) sein. Globale Labels dürfen nicht mit einer Ziffer beginnen.

Das spezielle Symbol '*' enthält immer die Adresse der aktuellen Quelltext Zeile. Dadurch werden Anweisungen wie z.B. "bra *+4" möglich.

VORSICHT! Vorwärtsreferenzen auf '*' werden von PhxAss bei einer Optimierung korrigiert, Rückwärtsreferenzen hingegen nicht! Um wirklich sicher zu sein, sollte man sowieso immer Labels verwenden.

1.14 Ausführbare M68000 Instruktionen

1.15 Instruktionsformat

Eine Assembler Quelltextzeile hat üblicherweise das Format:

<Label> <Operation> <Operanden>

Phxass erkennt alle Operationen aus Motorolas M68000PM/AD Programmer's Reference Manual sowie alle gebräuchlichen Abkürzungen wie BHS statt BCC, BLO statt BCS, MOVE statt MOVEA, ADD statt ADDI, usw. . In der vorliegenden Version werden alle MC68000,68010,68020,68030,68040,68060,68851,68881 und 68882 Instruktionen vollständig unterstützt.

Labels müssen unbedingt mit der ersten Spalte einer Zeile beginnen. Operationen (M68000 Instruktionen sowie Assembler Direktiven) müssen durch wenigstens ein Leerzeichen eingeleitet werden.

Das Operandenfeld kann aus bis zu vier (beim 68851) Operanden bestehen, die durch Kommas getrennt werden. Seit V4.10 darf das Operandenfeld auch Leerzeichen enthalten.

1.16 M68000 Standard Adressierungsarten

```
Bedeutung, der in diesem Abschnit verwendeten Abkürzungen:
   EA       - Effektive Adresse
   An       - Adressregister n
   Dn       - Datenregister n
   Xn.SIZE  - Bezeichnet Indexregister n (Data- oder Adressreg.) und sein
              Format (W für Wort oder L für Langwort)
   PC       - Programmzähler (Program Counter)
   dn       - Displacement (Verschiebung), mit n Bits Breite
   ( )      - Identifiziert eine indirekte Adresse in einem Register


 Data Register Direct
 (Datenregister Direkt)
Syntax:           Dn
Generation:       EA = Dn
Extension Words:  0

 Address Register Direct
 (Adressregister Direkt)
Syntax:           An
Generation:       EA = An
Extension Words:  0

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

 Address Register Indirect with Postincrement
 (Adressregister Indirekt mit nachfolgender Inkrementierung)
Syntax:           (An)+
Generation:       EA = (An), An = An + SIZE
Extension Words:  0

 Address Register Indirect with Predecrement
 (Adressregisrer Indirekt mit vorausgehender Dekrementierung)
Syntax:           -(An)
Generation:       An = An - SIZE, EA = (An)
Extension Words:  0

 Address Register Indirect with Displacement (16-Bit)
 (Adressregister Indirekt mit 16-Bit Verschiebung)
Syntax:           (d16,An)  or  d16(An)
```

```
Generation:       EA = (An) + d16
Extension Words:  1

 Address Register Indirect with Index (8-Bit Displacement)
 (Adressregister Indirekt mit Index und 8-Bit Verschiebung)
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)
 (Programmzähler Indirekt mit 16-Bit Verschiebung)
Syntax:           (d16,PC)  or  d16(PC)
Generation:       EA = (PC) + d16
Extension Words:  1

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

 Absolute Short Addressing
 (Absolute 16-Bit Adressierung)
Syntax:           (xxx).W  or  xxx.W
Generation:       EA bereits gegeben
Extension Words:  1

 Absolute Long Addressing
 (Absolute 32-Bit Adressierung)
Syntax:           (xxx).L  or  xxx.L
Generation:       EA bereits gegeben
Extension Words:  2

 Immediate Data
 (Unmittelbare Adressierung)
Syntax:           #xxx
Generation:       Operand gegeben
Extension Words:  1 or 2
```

1.17 68020+ Erweiterte Adressierungsarten

```
Bedeutung, der in diesem Abschnit verwendeten Abkürzungen:
   EA             - Effektive Adresse
   An             - Adressregister n
   Dn             - Datenregister n
   Xn.SIZE*SCALE  - Bezeichnet Indexregister n (Data- oder Adressreg.), sein
                    Format (W für Wort oder L für Langwort) und den Faktor
                    mit dem der Index multipliziert wird (1, 2, 4 oder 8).
   PC             - Programmzähler (Program counter)
   dn             - Displacement (Verschiebung), mit n Bits Breite
   bd             - Basis Displacement (base displacement)
   od             - Äußeres Displacement (outer displacement)
   ( )            - Identifiziert eine indirekte Adresse in einem Register
   [ ]            - Identifiziert eine indirekte Adresse im Speicher
```

```
 Address Register Indirect with Index
 (Adressregister Indirekt mit Index, Erweiterung des Standardformats)
Syntax:           (d8,An,Xn.SIZE*SCALE)
Generation:       EA = (An) + (Xn)*SCALE + d8
Extension Words:  1

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

 Memory Indirect Postindexed
 (Speicher Indirekt mit nachfolgender Indizierung)
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
 (Speicher Indirekt mit vorausgehender Indizierung)
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
 (Programmzähler Indirekt mit Index, Erweiterung des Standardformats)
Syntax:           (d8,PC,Xn.SIZE*SCALE)
Generation:       EA = (PC) + (Xn)*SCALE + d8
Extension Words:  1

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

 Program Counter Memory Indirect Postindexed
 (Programmzähler Speicher Indirekt mit nachfolgender Indizierung)
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
 (Programmzähler Speicher Indirekt mit vorausgehender Indizierung)
Syntax:           ([bd,PC,Xn.SIZE*SCALE],od)
Generation:       EA = (bd + An + Xn.SIZE*SCALE) + od
Extension Words:  1, 2, 3, 4 or 5
```

Die erweiterten Adressierungsarten haben ein paar Mehrdeutigtkeiten:
Z.B. würde (0,A0) gewöhnlich zu (A0) (ein Wort) optimiert, aber vielleicht möchten Sie gerne, daß die Null ein 32-Bit Displacement ist, und außerdem noch ein unterdrücktes D7 Register? Diese Instruktion würde dann genau dieselben Auswirkungen haben, wenn sie ausgeführt wird, allerdings benötigt sie acht Worte im Speicher, statt nur einem.
Seit PhxAss V3.70 muß man, um dies zu erreichen, nur "(0.L,A0,ZD7)"

schreiben.

1.18 Unterdrückte 68020+ Register

Die 'Memory Indirect Post/Preindexed' Adressierungsarten ermöglichen es dem Programmierer praktisch alles zu unterdrücken. Das bedeutet, daß man sich zum Beispiel aus dem in Reference Manuals angegebenem Standardformat ([bd,An,Xn.SIZE*SCALE],od) die folgenden Adressierungsarten bauen kann:

- ([bd,An,Xn.SIZE*SCALE])
- ([An,Xn.SIZE*SCALE],od)
- ([bd],od)
- ([An])
- ([Xn.SIZE*SCALE])
- ([An],od)

usw...

Um das unterdrückte Register genau zu spezifizieren, können die Zero-Register Symbole ZRn, ZPC verwendet werden. Durch den Einsatz dieser Zero-Register sowie der Displacement-Extensions '.W' und '.L' kann wirklich jede 68020 Instruktion erzeugt werden (hilfreich für Reassembler). Durch eine .W/.L Extension am ersten Displacement wechselt PhxAss automatisch in den 020+ Base-Displacement Modus und schaltet die Optimierung für die aktuelle Instruktion ab.

Die unterdrückten Register werden durch die folgenden Symbole repräsentiert:

- Unterdrücktes Datenregister D0-D7: ZD0-ZD7
- Unterdrücktes Adressregister A0-A7: ZA0-ZA7
- Unterdrückter PC: ZPC

Unterdrückte Register können nicht, durch EQUR, einem Symbol zugewiesen werden.

1.19 M68000 Instruktionsüberblick

Integer Instruktionen für alle Prozessoren
Integer Instruktionen für 010,020,030,040,060
Integer Instruktionen für 020,030,040,060
Integer Instruktionen für 040,060
Integer Instruktionen für 68060
MOVEC Kontrollregister
Fließkommainstruktionen 881,882,040,060
040/060er Fließkommainstruktionen
68851 PMMU Instruktionen
68030 PMMU Instruktionen
68040/060 PMMU Instruktionen

1.20 Integer Instruktionen (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 (nicht 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 Instruktionen (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 Instruktionen (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 (nur 68020!)
CAS.x       Dc,Du,<ea>                  Compare and Swap with Operand
CAS2.x      Dc1:Dc2,Du1:Du2,(Rn1):(Rn2) (nur 020-040!)
CHK2.x      <ea>,Rn  (nur 020-040!)     Check Register Against Bounds
CMP2.x      <ea>,Rn  (nur 020-040!)     Compare Register Against Bounds
DIVS.L      <ea>,Dq                     Signed Divide
DIVS.L      <ea>,Dr:Dq
DIVSL.L     <ea>,Dr:Dq  (nur 020-040!)
DIVU.L      <ea>,Dq                     Unsigned Divide
DIVU.L      <ea>,Dr:Dq
DIVUL.L     <ea>,Dr:Dq  (nur 020-040!)
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 (nur 68020!)
TRAP<cc>                                Trap on Condition
TRAP<cc>.x  #<data>
UNPK        -(Ax),-(Ay),#<adjustment>   Unpack BCD
UNPK        Dx,Dy,#<adjustment>
```

1.23 Integer Instruktionen (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 Instruktionen (68060)

```
LPSTOP      #x                        Low-Power Stop
```

Die Integer Instruktionen, die vom 68060 nicht direkt unterstützt werden, wie DIVUL, DIVSL, CAS2, CHK2, CMP2, MOVEP, werden trotzdem übersetzt, da sie meines Wissens durch die "68060.library" emuliert werden (so hoffe ich zumindest... :-).

1.25 MOVEC Kontrollregister (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 Fließkomma Instruktionen (68881,68882,68040,68060)

Viele von den nachfolgenden Instruktionen müssen für einen 68040 oder 68060 Software-emuliert werden. Trotzdem wird PhxAss den entsprechenden Code ohne jegliche Warnung erzeugen.

Emulierte Instruktionen beim 68040:
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

Emulierte Instruktionen beim 68060:

```
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
FTANTH      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 FP System Control Register
FMOVE.L     FPcr,<ea>                   (FPcr = FPCR, FPSR or FPIAR)
FMOVECR     #ccc,FPn                    Move Constant ROM
FMOVEM      <list>,<ea>                 Move Multiple FP Data Registers
FMOVEM      Dn,<ea>
FMOVEM      <ea>,<list>
FMOVEM      <ea>,Dn
FMOVEM.L    <list>,<ea>                 Move Multiple FP Control Regs.
FMOVEM.L    <ea>,<list>                 (<list> = comb. of FPCR,FPSR,FPIAR)
FNOP                                    No Operation
FRESTORE    <ea>                        Restore Internal FP State
FSAVE       <ea>                        Save Internal Floating-Point State
FS<cc>      <ea>                        Set According to FP 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 Fließkomma Instruktionen (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 Instruktionen (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 Log. 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 Instruktionen (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 Instruktionen (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 Ausdrücke

Ausdrücke bestehen aus Symbolen und Konstanten. Symbole können absolut, relocatibel oder extern sein. PhxAss unterstützt die folgenden arithmetischen Operationen für INTEGER Ausdrücke (von der höchsten zur niedrigsten Priorität) :

```
1.  ~    Nicht (unär)      -    Negation (unär)
2.  <<   Links Shift       >>   Rechts Shift
3.  *    Multiplikation    /    Division                //   Modulo
4.  &    Und               |    Oder ('!' auch erlaubt)  ^    Exklusiv Oder
5.  -    Subtraktion       +    Addition
6.  ( )  runde Klammern    [ ]  eckige Klammern
```

Für absolute Symbole und Konstanten (die ebenfalls absolut sind), sind alle arithmetischen Operationen erlaubt.
Wenn relocatible oder externe Symbol im Ausdruck erscheinen, ist nur Subtraktion und Addition mit einigen Einschränkungen möglich:

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

sind erlaubt, alle anderen Ausdrücke sind illegal.

Fließkomma Ausdrücke bestehen aus Fließkomma Konstanten und Symbolen, sowie absoluten Integer Konstanten und Symbolen. Die folgenden Operationen und Funktionen sind für Fließkommaausdrücke erlaubt (seit V4.00):

```
Binär:
   +    Plus              -    Minus             *    Multiplikation
   /    Division          ^    Potenzierung
Unär:
   -    Negation          sqr  Quadratwurzel     exp  e^x
```

```
   log  Nat. Logarithmus   sin  Sinus              cos  Cosinus
   tan  Tangens
```

SQR, EXP, LOG, SIN, COS und TAN sind Funktionen und können aus Groß- oder aus Kleinbuchstaben bestehen. Normalerweise werden sie direkt vor einen geklammerten Term geschrieben, z.B. "sin(3.14159)". Wenn aber, wie im letzten Beispiel, der Term nur aus einer einzigen Konstante besteht, ist es auch erlaubt "sin:3.14159" zu schreiben. Der ':' ist unbedingt erfoderlich um den Funktionsnamen von einem möglichen Symbolnamen trennen zu können.

Es gibt sechs verschiedene Arten von Konstanten:

Hexadezimale, eingeleitet durch ein '$', bestehen aus '0'-'9' und 'A'-'F' (oder 'a'-'f')
Dezimale, bestehen aus '0'-'9'
Fließkomma, haben das Format [+/-][integer][.nachkomma][E[+/-]exponent]
Oktale, eingeleitet durch ein '@', bestehen aus '0'-'7'
Binäre, eingeleitet durch ein '%', bestehen aus '0' und '1'
String-, eingebettet von ' oder ", bestehen aus bis zu vier Zeichen.

Das '\' Zeichen ist ein Escape-Symbol, das die folgenden Codes erzeugen kann:

```
  \\     der '\'-character selbst
   \'    character #39 (Apostroph)
   \"    character #34 (Anführungszeichen)
   \0    character #0  (Stringbegrenzer)
   \n    character #10 (Zeilenvorschub)
   \f    character #12 (Formularvorschub)
   \b    character #8  (Rückstelltaste)
   \t    character #9  (Tabulator)
   \r    character #13 (Returntaste)
   \e    character #27 (Escape Code)
   \c    character #155(leitet Kontrollsequenz (CSI) ein)
```

"" und '' innerhalb von Strings werden durch " bzw. ' ersetzt (V4.14).

1.32 Assembler Direktiven

In den folgenden Abschnitten werden alle Direktiven die PhxAss unterstützt ausführlich beschrieben.
Wichtiger Hinweis! Direktiven dürfen *nicht* in der ersten Spalte einer Zeile beginnen, oder sie werden als Labels behandelt! (das war ein Hinweis für Seka User :-)

Folgende Direktiven werden von PhxAss unterstützt:

```
      BLK      Konstanten Block definieren
      BSS      Bss Section
      BSS      Speicher für ein Bss Symbol reservieren
     BSS_C     Chip-RAM Bss Section
     BSS_F     Fast-RAM Bss Section
     CNOP      Nachfolgenden Code ausrichten
     CODE      Code Section
    CODE_C     Chip-RAM Code Section
    CODE_F     Fast-RAM Code Section
```

```
 COMMENT    Kommentarzeile
  CSEG      Code Section
  DATA      Data Section
 DATA_C     Chip-RAM Data Section
 DATA_F     Fast-RAM Data Section
   DC       Konstante definieren
   DCB      Konstanten Block definieren
   DS       Speicherplatz reservieren
  DSEG      Data Section
   DX       Speicherplatz reservieren
  ECHO      Text ausgeben
  ELSE      ELSE-Teil für bedingte Assemblierung definieren
 ELSEIF     ELSE-Teil für bedingte Assemblierung definieren
   EQU      Ausdruck einem Symbol zuweisen
  EQU.x     Fließkommaausdruck einem Symbol zuweisen
  EQUR      Register einem Symbol zuweisen
   END      Ende des Quelltextes
  ENDC      Ende der bedingten Assemblierung
  ENDIF     Ende der bedingten Assemblierung
  ENDM      Ende einer Macro Definition
  ENDR      Ende der REPT Schleife
  EVEN      Nachfolgenden Code auf gerade Adresse ausrichten
  FAIL      Übersetzungsvorgang abbrechen
   FAR      In den FAR-Code/Data Modus wechseln
  FILE      Name der Zieldatei für absoluten Code
   FPU      FPU Codeerzeugung erlauben
 GLOBAL     Speicherplatz für globales Bss Symbol reservieren
  IDNT      Unit Namen definieren
   IFC      Bed.Ass.: Zwei Strings auf Gleichheit testen
   IFD      Bed.Ass.: Testen ob ein Symbol definiert ist
  IFEQ      Bed.Ass.: Testen ob ein Ausdruck Null ist
  IFGT      Bed.Ass.: Testen ob ein Ausdruck größer Null ist
  IFGE      Bed.Ass.: Testen ob ein Ausdruck größer oder gleich Null ist
  IFLT      Bed.Ass.: Testen ob ein Ausdruck kleiner Null ist
  IFLE      Bed.Ass.: Testen ob ein Ausdruch kleiner oder gleich Null ist
  IFNC      Bed.Ass.: Testet zwei Strings auf Verschiedenheit
  IFND      Bed.Ass.: Testen ob ein Symbol undefiniert ist
  IFNE      Bed.Ass.: Testen ob ein Ausdruck ungleich Null ist
 INCBIN     Binärdatei einbinden
 INCDIR     Pfad für Include-Dateien setzen
 INCLUDE    Quelltext einbinden
 INITNEAR   Near-Modus Basisregister initialisieren
   INT      Wert eines Fließk.ausdrucks an ein Int. SET-Symbol übergeben
  LIST      Die folgenden Zeilen erscheinen im Listing File
  LOAD      Zieladresse für absolute Codeerzeugung
 MACHINE    CPU-Typ bestimmen
  MACRO     Macro Definition
  MEXIT     Macro vorzeitig verlassen
  NEAR      Near Modus
 NOLIST     Folgezeilen sind im Listing File unsichtbar
  NREF      Near Modus Symbol importieren (wie XREF)
 OFFSET     Beginnt eine OFFSET Section
   OPT      Optimierungsmodus ändern
   ORG      Startadresse für absolute Codeerzeugung
  PMMU      68851 Code Generierung
PROCSTART   Beginn einer C-Funktion beim DICE-Compiler
 PROCEND    Ende einer C-Funktion beim DICE-Compiler
```

```
    PUBLIC     Importiere oder exportiere Symbol
      REG      Dem Symbol eine Registerliste zuweisen
     REPT      Zeilen zwischen REPT...ENDR werden beliebig oft wiederholt
     RORG      Abstand zum Anfang der Section für den nachfolgenden Code
      RS       Wert des RS-Zählers einem Symbol zuweisen
    RSRESET    RS-Zähler auf Null zurücksetzen
     RSSET     RS-Zähler auf bestimmten Wert setzen
    SECTION    Der folgende Code kommt in die angegebene Section
      SET      Wert eines SET-Symbols ändern
     SET.x     Wert eines Fließkomma SET-Symbols ändern
    SUBTTL     (ohne Funktion)
      TTL      Namen der Unit bestimmen
   TRACKDISK   Schreibt absoluten Code direkt auf eine Diskette
     XDEF      Symbol exportieren
     XREF      Symbol importieren

Diese Direktiven werden von der Freeware Version *nicht* unterstütuzt:
RSRESET,RSSET,RS,ECHO,LIST,NOLIST,INCDIR,INCLUDE,INCBIN,MACRO,ENDM,MEXIT,
RORG,OFFSET,ORG,FILE,LOAD,TRACKDISK,COMMENT,SUBTTL,IF<cc>,ELSE,ELSEIF,
ENDC,ENDIF,FPU,PMMU,REPT,ENDR,INT
```

1.33 EQU

```
   symbol   equ      <ausdruck>
   symbol   =        <ausdruck>
```

Der Ausdruck wird dem Symbol zugewiesen.

1.34 EQU.x

```
   symbol   equ.x    <fließkommaausdruck>
   symbol   =.x      <fließkommaausdruck>
```

Ein EQU mit .d,.f,.p,.s,.x-Erweiterung übergibt sen Wert eines Fließkommaausdrucks an ein Symbol. Wenn Sie mehr über Fließkommaausdrücke wissen wollen, schlagen sie im Abschnitt Ausdrücke nach.
Diese Direktive ist PhxAss-speziell.

1.35 EQUR

```
   symbol   equr     <register>
```

Durch diese Direktive wird einem Symbol ein CPU Register (D0-D7, A0-A7 oder SP) zugewiesen.
Seit V3.00 muß das Registersymbol bereits vor der ersten Benutzung definiert sein.

1.36 REG

```
   symbol   reg      <registerliste>
```

Hiermit wird ein Symbol mit einer ganzen Registerliste belegt. Gültige Registerlisten dürfen mehrere Registernamen (siehe EQUR), getrennt durch '/'-Zeichen, enthalten. Das '-'-Zeichen definiert einen ganzen Bereich von Registern. Es folgen Beispiele für gültige Registerlisten:

```
   a1/a3-a5/d0/d2/d4
   d0-d7/a2-a6
   d1-3/d5-7/a0-1/a3-6  (seit V3.56)
```

1.37 SET

```
   symbol   set      <absoluter ausdruck>
```

Der Wert eines absoluten Ausdrucks wird einem SET-Symbol zugewiesen. In diesem Ausdruck sind keine relocatiblen oder externen Symbol zulässig. Ein Symbol, das durch SET definiert wurde, kann jederzeit durch eine weitere SET Direktive geändert werden.
Einige SET-Symbole sind schon von PhxAss vordefiniert:

```
_PHXASS_    set      1
_VERSION_   set      version<<16+revision
```

Je nachdem welcher Prozessor und Coprozessor im System vorhanden ist, definiert PhxAss auch die folgenden Symbole:
_MC68000_, _MC68010_, _MC68020_ and _MC68881_. The symbols _MC68030_, _MC68040_, _MC68060_, _MC68881_ und _MC68882_.

NARG enthält die Anzahl der spezifizierten Macro-Parameter und ist außerhalb eines Macros immer Null.

__RS enthält immer den aktuellen Wert des RS-Zählers.

1.38 SET.x

```
   symbol   set.x    <fließkommaausdruck>
```

Ein SET mit .d,.f,.p,.s,.x-Erweiterung weist einem Symbol den Wert eines Fließkommaausdrucks zu. Dieser Wert darf durch weitere SETs verändert werden, unter der Voraussetzung das der Typ des Symbols nicht verändert wird (z.B. bei "symbol SET.S" gefolgt von "symbol SET.D").
Diese Direktive ist PhxAss-speziell.

1.39 INT

```
   symbol   int      <fließkommaausdruck>
```

Der Fließkommaausdruck wird berechnet und sein Resultat, ohne den Nachkommateil, einem Integer SET-Symbol zugewiesen.

1.40 RSRESET

Diese Direktive setzt den internen RS-Zähler auf Null zurück.

1.41 RSSET

```
            rsset    [<zähler>]
```

Diese Direktive setzt den internen RS-Zähler auf den Wert des <zähler> Ausdrucks.

1.42 RS

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

Durch RS wird der aktuelle Wert des internen RS-Zählers an ein Symbol übergeben (wenn eines angegeben wurde). Danach wird der RS-Zähler um die in der Opcode-Erweiterung angegebene Datenbreite multipliziert mit <anzahl> inkrementiert. Wenn <anzahl> fehlt, wird sie gleich Null gesetzt. Gültige Opcode-Erweiterungen können Sie z.B. im Abschnitt der DC Direktive erfahren.
Der augenblickliche Zustand des RS-Zählers kann zusätzlich über das __RS Symbol abgefragt werden.

1.43 IDNT

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

Diese Direktiven bestimmen den Namen der Object Modul Unit, die der Assembler erzeugen wird. Normalerweise wird dazu der Name des Quelltextes, ohne seine Namenserweiterung, verwendet.

1.44 SUBTTL

Ein Quelltext, der SUBTTL enthält, wird auf PhxAss keinen Fehler erzeugen, aber bisher tut die Direktive auch noch nichts.
(Um ehrlich zu sein, ich wüßte auch gar nicht was sie tun sollte? Wenn jemand weiß, worum es sich hier handelt, kann er mir ja mal schreiben...)

1.45 COMMENT

```
            comment  text
```

Hinter diese Direktive kann beliebiger Kommentartext geschrieben werden.

1.46 LIST

Der nachfolgende Quelltext wird in einem Listing File zu sehen sein.

1.47 NOLIST

Der nachfolgende Quelltext wird in einem Listing File nicht zu sehen sein.

1.48 OPT

```
            opt      <optimierungsschalter>
```

Ändert den Optimierungsmodus. Eine Auflisting aller Optimierungsschalter ist in CLI Parameter enthalten.
Diese Direktive ist PhxAss-speziell.

1.49 MACRO, ENDM

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

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

Durch diese Direktive wird ein Macro definiert. Der Name des Macros darf auf der linken oder der rechten Seite erscheinen. Wenn PhxAss im Quelltext auf einen Macronamen stößt, wirdern die Zeilen zwischen MACRO und ENDM in den Quelltext eingefügt. Bei einem Macroaufruf können bis zu neun, durch Kommas getrennte, Parameter im Operandenfeld angegeben werden. Sie werden im Macro als '\1' bis '\9' bezeichent. '\0' ist für die Opcode-Erweiterung reserviert. Beispiel:

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

Das Macro wird aufgerufen durch: bhs.s label
Dabei wird ".s" in \0 und "label" in \1 gespeichert.
'\@' im Macro wird durch eine einzigartige, dreistellige, Ziffernfolge ersetzt, die bei jedem neuen Macro-Aufruf anders ist.

Labels innerhalb eines Macros sollten unbedingt '\@' enthalten, um eine Mehrfach-Definition von Labels zu verhindern.

1.50 MEXIT

Wenn der Assembler innerhalb eines Macros auf diese Direktive stößt, sucht er sofort nach ENDM und verläßt das Macro.

1.51 END

Im ersten Durchgang wird der Rest des Quelltextes hinter END ignoriert. Im zweiten Durchgang schließt der Assembler alle offenen Dateien und beendet den Übersetzungsvorgang.
Normalerweise geschieht dies automatisch am Ende eines Quelltextes.

1.52 FAIL

PhxAss gibt die Fehlermeldung "69 Assembly abortet!" aus, und beendet den Überetzungsvorgang.

1.53 ECHO

```
echo    <string>
```

Der angegebene String wird auf StdOut ausgegeben (normalerwiese CLI-Fenster). Wenn kein String angegeben wurde, wird nur ein Zeilenvorschub ausgegeben.
Diese Direktive ist PhxAss-speziell.

1.54 MACHINE

```
machine  <processor>
```

Diese Direktive bestimmt den Prozessortyp für den der nachfolgende Code generiert werden soll. Gültige Prozessoren sind:
68000, 68010, 68020, 68030, 68040, 68060
Die Implementierung dieser Direktive kann auf anderen Assemblern abweichen.

1.55 FPU

```
fpu [<cpID>]
```

Code Generierung für einen MC68881/68882 Coprozessor wird erlaubt.
Normalerweise sollte <cpID> auf '1' gesetzt werden, da dies auf den
meisten Systemen die korrekte ID für einen Fließkommacoprozessor ist.
(Vielleicht haben Sie aber auch mehrere FPUs, die unter verschiedenen
IDs angesprochen werden müssen :-)
<cpID> sollte auf jeden Fall niemals auf '0' gesetzt werden, da diese
ID fest für eine PMMU reserviert ist.
Wenn als Prozessortyp bereits schon 68040 oder 68060 gewählt wurde,
sollte diese Direktive nicht verwendet werden (wegen interner FPU).
Diese Direktive ist PhxAss-speziell.

1.56 PMMU

Code Generierung für eine MC68851 Paged Memory Management Unit wird
hierdurch ermöglicht. PMMU macht nur Sinn, wenn als Prozessortyp
bereits '68020' gewählt wurde.
Diese Direktive ist PhxAss-speziell.

1.57 SECTION

```
section  <name>[,<typ>[,<memflag>]]
```

Der nachfolgende Code wird in die Section mit dem Namen <name>
plaziert. Es gibt drei verschiedene Sectionstypen: CODE,DATA und BSS.
CODE enthält die ausführbaren M68000 Instruktionen, DATA enthält
initialisierte und BSS uninitialisierte Daten (werden beim Programm-
start auf Null gesetzt). Wird <typ> nicht angegeben, so entspricht
dies einer CODE Section.
Die Section wird normalerweise in den Speicher mit der höchsten
Priorität geladen. Durch Angabe des <memflag> als FAST oder CHIP,
kann man aber auch bestimmen ob die Section unbedingt ins Fat- oder
Chip-Ram geladen werden muß.

Aus Gründen der Kompatibilität werden CODE_C, DATA_C und BSS_C eben-
falls als Sectionstypen anerkannt (seit V3.56).

Durch das Erzeugen einer Section wechselt der Assembler in den Relo-
catiblen Modus, in dem dann die folgenden Direktiven nicht mehr be-
nutzt werden können:
org, load, file, trackdisk.

1.58 CODE, CSEG

Diese Direktiven entsprechen: section "CODE",code

1.59 DATA, DSEG

Diese Direktiven entsprechen: section "DATA",data

1.60 CODE_C, CODE_F, DATA_C, DATA_F, BSS_C, BSS_F

Siehe CODE, DATA oder BSS. Außerdem wird <memflag> definiert, wodurch die Section ins Fast- (xxx_F) oder ins Chip-Ram (xxx_C) gezwungen wird.

1.61 BSS

Diese Direktive entspricht: section "BSS",bss

1.62 BSS

```
bss      symbol,<anzahl>
```

Wenn BSS Parameter im Operandenfeld besitzt, entspricht dies einer völlig anderen Direktive: Das angegebene Symbol wird in der BSS-Section plaziert, und es werden hierfür <anzahl> Bytes Speicher reserviert.
Diese Direktive ist nur zur Kompatibilität mit Aztec-C vorhanden.

1.63 GLOBAL

```
global   symbol,<anzahl>
```

Diese Direktive tut praktisch dasselbe wie BSS symbol,<anzahl>, allerdings wird das Symbol gleichzeitig, mittels XDEF exportiert.
Diese Direktive ist nur zur Kompatibilität mit Aztec-C vorhanden.

1.64 OFFSET

```
offset   [<start offset>]
```

Kennzeichnet den Anfang einer speziellen Offset-Section. Alle Labels, die in dieser Section definiert wurden, werden wie absolute Offsets behandelt, statt wie Adressen. <start offset> ist normalerweise Null. Solch eine Section kann nützlich sein, um Struktur-Offsets mit Hilfe der DS.x Direktive zu erzeugen.
Solange Sie Ihre Programmer aber mit PhxAss schreiben, sollten sie die viel schnelleren RSRESET, RSSET und RS.x Direktiven verwenden.
OFFSET wurde nur aus Gründen der Kompatibilität implementiert.

1.65 RORG

```
rorg     <section offset>
```

Hiermit wird die genaue Position des nachfolgenden Code, relativ zum Start der aktuellen Section, bestimmt.

1.66 INCDIR

```
incdir   <pfad1>[,<pfad2>,...]
```

Diese Direktive macht dasselbe wie das INCPATH Argument (siehe auch CLI Parameter).
Andere Assembler unterstützen meistens keine Mehrfachpfaddefinitionen.

1.67 INCLUDE

```
include  <datei>
```

Durch diese Direktive wird PhxAss dazu veranlaßt die Übersetzung des momentanen Quelltextes zu unterbrechen und den Quelltext aus <datei> zu übersetzen. Wenn dies geschafft ist, wird mit dem Original-Quelltext weiter gemacht.
Wenn PhxAss die Include-Datei nicht finden kann, wird zuerst im Verzeichnis, das in der Environment-Variablen PHXASSINC gespeichert wurde, nachgesehen. Dann werden die Verzeichnisse, die durch mögliche INCPATH-Parameter (siehe CLI Parameter) bezeichnet wurden, durchsucht. Zum Schluß wird in den durch INCDIR spezifizierten Verzeichnissen nachgesehen. Dann erst gibt es eine Fehlermeldung.

1.68 INCBIN

```
incbin   <datei>
```

Das angegebene Binärdatei wird in die aktuelle Section eingefügt (das können z.B. Grafiken, Samples oder trigonometrische Tabellen sein). PhxAss durchsucht dabei dieselben Include-Verzeichnisse wie die INCLUDE Direktive.

1.69 XREF

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

Die hier angegebenen Symbole sind extern definiert und werden erst durch den Linker eingesetzt.
Andere Assembler unterstützen meistens nur ein Symbol.

1.70 NREF

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

NREF tut praktisch dasselbe wie XREF, aber der Assembler wird dazu gezwungen, diese Symbole im Near-Modus zu adressieren. Diese Direktive ist PhxAss-speziell.

1.71 XDEF

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

Die angegebenen Symbole werden exportiert, in dem ihre Namen und Adressen im External-Block der Object-Datei gespeichert werden. Dadurch kann der Linker die Adressen dieser Symbole lesen und sie in andere Object-Dateien einsetzen.
Andere Assembler unterstützen meistens nur ein Symbol.

1.72 PUBLIC

```
            public   symbol1[,symbol2,...]
```

Wenn das angegebene Symbol im vorliegenden Quelltext definiert ist, macht PUBLIC dasseble wie XDEF. Ansonsten, wenn es unbekannt ist, ist PUBLIC mit XREF identisch.
Diese Direktive ist nur zur Kompatibilität mit Aztec-C vorhanden.

1.73 ORG

```
            org      adresse
```

Definiert die absolute Startadresse des nachfolgenden Codes und läßt den Assembler in den absoluten Modus wechseln. Ab V1.8 sind auch mehrere ORG Direktiven im Quelltext erlaubt, wobei jede als der Beginn einer neuen Section gesehen werden kann. Folgende Direktiven sind im absoluten Modus nicht verfügbar:
ttl, code, cseg, data, dseg, bss, section, offset,
xref, nref, xdef, public, idnt.

1.74 LOAD

```
            load     adresse
```

Nachdem der Übersetzungsvorgang beendet ist, wird der absolute Code ab der angegebenen Adresse im Speicher abgelegt. Normalerweise wird er an die Adresse gespeichert, die bereits als Startadresse festgelegt worden ist. Seien Sie vorsichtigt mit dieser Direktive, da der

Speicher an der Zieladresse weder geprüft noch allociert wird.
Diese Direktive ist PhxAss-speziell (auch von SEKA bekannt).

1.75 FILE

```
file     <datei>
```

Nachdem der Übersetzungsvorgang beendet ist, wird der absolute Code in der angegebenen Datei gespeichert.
Diese Direktive ist PhxAss-speziell.

1.76 TRACKDISK

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

Nachdem der Übersetzungsvorgang beendet ist, wird der absolute Code direkt, unter Benutzung des 'trackdisk.device' auf Diskette geschrieben. Dabei ist <laufwerk> von 0 bis 3 und <startblock> von 0 bis 1759 gültig (bei HD-Disks doppelt soviel). <offset> ist normalerweise Null, und bestimmt den Abstand zum Blockanfang (0-511).
Diese Direktive ist PhxAss-speziell.

1.77 NEAR

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

Diese Direktive initialisiert die vom Near/Small-Data Modell benötigten Parameter. NEAR inklusive Argumenten darf im ganzen Quelltext nur ein einziges Mal auftauchen. Nach dieser Initialisierung kann der Near-Data Modus durch NEAR und FAR ohne Argument an und ausgeschaltet werden. 'NEAR An,0' sollte allerdings nicht vor der ersten SECTION, CODE, DATA, etc. Direktive angewandt werden.
In diesem Modus ist es möglich Symbole über 'NearSymbol(An)' anzusprechen. Absolute Referenzen auf Near-Symbole werden automatisch nach 'Address Register Indirect' konvertiert (wenn möglich).
Das erste Argument, das Adressregister, ist von A2 bis A6 gültig und ist A4, wenn es nicht näher bestimmt wurde. <secnum> ist mit -2 voreingestellt und bestimmt die Nummer der Near-Section.
Durch <secnum>=-1 werden alle Data und Bss Sectionen zu einer großen Small-Data Section vereinigt. Entwender geschieht das direkt durch PhxAss, falls es möglich war sofort eine ausführbare Datei zu erzeugen, oder ein Linker muß mit der passenden Small-Data Option aufgerufen werden.
Durch <secnum>=-2 werden nur die Data oder Bss Sectionen die den Namen "__MERGED" tragen, zur Small-Data Section hinzugefügt.

```
near     code
```

Wenn der String "CODE" als Argument verwendet wird, aktiviert PhxAss

den Small-Code Modus, in dem alle extern definierten, absoluten, JMPs und JSRs nach PC-Relativ (16-Bit) konvertiert werden.

Andere Assembler akzeptieren keine Argumente hinter NEAR.

1.78 FAR

Deaktiviert den Small-Code/Data Modus, falls er aktiv war.

1.79 INITNEAR

Hierdurch werden zwei Instruktionen in den Code eingefügt, welche den Small-Data Modus unter Verwendung der bei NEAR getätigten Definitionen initialisieren. Folgender Code wird dabei erzeugt (10 Bytes):

```
            lea     SmallDataBase,An
            lea     32766(An),An
```

Diese Direktive ist PhxAss-speziell.

1.80 DC

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

Die DC (Define Constant) Direktive reserviert und initialisiert ein oder mehrere Speicherfelder. Jedes Feld hat dieselbe Größe, die durch die Opcode-Erweiterung bestimmt wird. Jeder Byte-,Word- oder Longword-<wert> kann ein Ausdruck sein und Vorwärtsreferenzen beinhalten.
Folgende Opcode-Erweiterungen sind möglich:

.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			

Die meisten anderen Assembler dürften mit Fließkomma oder Quadwords Probleme bekommen.

1.81 DCB, BLK

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

Diese Direktiven reservieren einen Speicherblock mit <num> Einträgen. Der verfügbaren Größen der Einträge sind dieselben wie in DC.
Der Block wird mit <füllwert> initialisiert, der Null ist, wenn kein besonderer Wert angegeben wurde. Alle erlaubten Opcode-Erweiterungen sind bei DC nachzulesen.

1.82 DS, DX

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

Ein Speicherblock mit <anz> Einträgen wird reserviert und mit Null initialisiert. Siehe auch DCB link dcb} oder @{.

1.83 CNOP

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

Die Adresse des nachfolgenden Codes wird auch eine durch <align> teilbare Adresse ausgerichtet. Danach wird der <offset> hinzuaddiert. Beispiel: cnop 2,4 . Dieses Beispiel würde die nächste Adresse auf zwei Bytes nach der nächsten Longword-Grenze ausrichten. Werte für <align>, die größer als acht sind, machen keinen Sinn für relocatiblen Code (siehe AllocMem(), exec.library).

1.84 EVEN

Diese Direktive entspricht einem cnop 0,2 was dafür sorgt, daß die nächste Adresse gerade ist.

1.85 IFcond, ELSEIF, ELSE, ENDIF, ENDC

Diese Direktiven ermöglichen bedingte Assemblierung. Die übliche Form der IF Direktive ist:

```
            IF<bedingung>   <ausdruck> oder Symbol
            ...
            [ELSE (oder ELSEIF)
            ...]
            ENDC (oder ENDIF)
```

PhxAss unterstützt die folgenden Bedingungen:

IFC "string1","string2"	Vergleicht zwei Strings. Die kann innerhalb von Macros nützlich sein, wenn die Strings z.B. '\x'-Macroargumente enthalten.
IFD/IFND symbol	Testet ob das Symbol definiert/undef. ist.
IFEQ/IFNE <exp>	Testet ob <exp> Null/nicht Null ist.
IFGT/IFLT <exp>	Testet ob <exp> größer/kleiner Null ist.
IFGE/IFLE <exp>	Testet ob <exp> größer/kleiner Null oder gleich Null ist.

1.86 PROCSTART,PROCEND

Diese Direktiven sind zur Kompatibilität mit dem DICE-C System vorhanden, doch bewirken sie zu diesem Zeitpunkt noch überhaupt nichts. In der Zukunft sollte es hiermit möglich sein LINK A5,#0 / UNLK A5 aus einer C-Funktion zu entfernen, wenn A5 zwischen PROCSTART und PROCEND nicht referenziert wurde.

1.87 REPT/ENDR

```
rept     <zähler>
...
endr
```

Der Teil des Quelltextes, der von REPT/ENDR eingeschlossen ist, wird so oft wiederholt wie es <zähler> angibt. Ein negativer <zähler> ist natürlich verboten.

1.88 Compiler Kompatibilität

Ein Hauptgrund, warum ich damals begann PhxAss zu schreiben, war einen Assembler zu besitzen der den unglaublich langsamen AS-Assembler von Aztec-C ersetzt. Daher existieren sehr viele Direktiven um Aztec-Kompatibilität zu erreichen, aber seit V3.30, wo Symbole die mit einem '.' beginnen als lokale Symbole betrachtet werden, ist es praktisch unmöglich Aztec-Compiler Code zu übersetzen. Die einzige Lösung wäre z.B. ein Programm zu schreiben, das alle '.nnn' Symbole in '_nnn' umwandelt.

Seit der Einführung der neuen Direktiven PROCSTART und PROCEND in V3.71 werden DICE-C Quelltexte vollkommen unterstützt.

1.89 Linker

Jeder Linker, der das standard Amiga DOS Object Modul Format unterstützt, darf verwendet werden. Also z.B. BLink oder DLink.
Empfohlen wird natürlich die Verwendung von PhxLnk :-).

Ab V4.00 wird der Linker jedoch nur noch für das zusammenbinden von mehreren Modulen benötigt. PhxAss erzeugt automatisch ein ausführbares Programm, solange keine externen Referenzen vorhanden sind.
Zwei Features von PhxLnk sind allerdings nicht in PhxAss implementiert:
1. Erzeugung von HUNK_RELOC32SHORT Blöcken (16-Bit Offsets)
2. Entfernen von Null-Bytes am Ende einer Code- oder Data-Section (die sogenannten Code-Bss bzw. Data-Bss Sectionen)

Wenn Sie eines dieser Features nutzen möchten (wobei Ihr Programm dann allerdings nur noch ab OS2.04 läuft), sollten Sie den NOEXE Schalter setzen und PhxLnk aufrufen.

1.90 Fehlermeldungen

In der vorliegenden Version von PhxAss können die folgenden Fehlermeldungen auftreten:

01 Nicht genügend Speicher vorhanden.

02 Kann die utility.library nicht öffnen

03 Kann das timer.device nicht öffnen

04 DREL16 Symbol ist außer Reichweite
Ihr Small Data Segment ist zu groß. Alle Data und Bss Sections dürfen zusammen nicht 64k überschreiten.

07 HEADINC: Dateiname erwartet
Beispiel: PhxAss HEADINC "dh0:datei1,dh1:xdir/datei2,"

08 IncDir Pfadname erwartet
Beispiel: incdir "dir1","dir2",
Kann auch durch INCPATH ausgelöst werden.

10 SMALLDATA: Illegales Basisregister
Erlaubt sind 2-6 für A2-A6. A4 ist Standard.

11 Diese MACHINE wird nicht unterstützt
Ungültiger Prozessortyp angegeben. Die aktuelle Version von PhxAss unterstützt die Prozessoren 68000, 68010, 68020, 68030, 68040 und 68060.

12 Datei existiert nicht
Es war nicht möglich die Quelltextdatei zu öffnen.

13 Kein Include-Dateiname angegeben

14 Lesefehler

15 Überlauf des Stringpuffers
Die Länge eines Labels, Opcodes oder Operandenfeldes darf 128 Zeichen nicht überschreiten.

16 Zu viele Sectionen
Maximal möglich sind 250.

17 Symbol kann nicht extern gemacht werden
XDEF kann nur auf absolute oder ralocatible Symbole angewandt werden.

18 Symbol wurde doppelt deklariert

19 Symbol kann kein XREF werden
Ein Symbol, das bereits im aktuellen Quelltext definiert worden ist, kann nicht gleichzeitig extern definiert worden sein. Oder: Ein Symbol, das bereits mit XREF als extern definiert wurde, darf nicht im aktuellen Quelltext definniert werden.

20 Illegale Namenserweiterung im Opcode Feld
Erlaubt: .b .w .l .s .f .d .x .p .q

```
21 Ungültiger Macro Parameter
   Mögliche Parameter sind: \0 (Opcode-Erweiterung), \1 - \9 und \@

22 Ungültige Zeichen in der Sprungmarke
   Siehe Marken (Labels) im Programmierer Info Abschnitt.

23 Unbekannte Direktive
   Der Opcode ist weder ein 680x0-Mnemonic noch eine Assembler Direktive
   oder ein Macro.

24 Zu viele Makroparameter
   Neun Parameter ( \1 bis \9 ) sind maximal möglich.

25 Kann das trackdisk.device nicht öffnen

26 Argumentenpuffer ist übergelaufen
   Argumente sind in den meisten Fällen auf 128 Zeichen beschränkt.

27 Fehlerhaft Registeliste
   Gültige Registerlisten:
   d0-d3  d3-d4/a2  d2/d3/a4-a6  d7  a0/d2  d2-6/a0-4

28 Vermisse die Sprungmarke
   Die Direktive benötigt eine Marke (Label).
   Beispiel: EQU <exp>   -> Error 28

29 Illegales Trennzeichen für eine Registerliste
   Gültige Trennzeichen sind '-' und '/'.

30 Für ein lokales Symbol sind SET, MACRO, XDEF, XREF und PUBLIC nicht
   erlaubt

31 Kein Register (versuchen Sie d0-d7, a0-a7 oder sp)

32 Zu viele ')'

33 Unbekannter Adressierungsmodus
   Siehe Standard Adressierungsarten und Erweiterte Adressierungsarten
   für eine genaue Beschreibung aller möglichen Adressierungsarten.

34 Adressierungsmodus wird nicht unterstützt
   Beispiel:   move.b  d0,a1  /  move  usp,d2  /  clr.w  (d3)+  -> Error 34

35 Makro darf nicht im Operanden stehen

36 Undefiniertes Symbol

37 Vermisse Register
   Beispiel:   mulu   d0,   -> Error 37

38 Benötige ein Datenregister

39 Benötige ein Adressregister

40 Word an ungerader Adresse
   Beispiel:  dc.b "Hallo"
```

```
            dc.w 0        -> Error 40
   Fügen Sie ein CNOP 0,2 oder EVEN nach String-Konstanten ein.

41 Syntaxfehler im Operanden

42 Relozierungsfehler
   Beispiel:  move.l  label(pc),d0  , wobei Label keine Adresse aus der
    aktuellen Section ist.  -> Error 42

43 Zu große Distanz
   Beispiel:   move.w  50000(a0),d0   -> Error 43
   Zu große Distanz bei indirekter Adressierung oder einem Branch-Befehl.
   Byte-Branches haben eine Reichweite von +126/-128 Bytes. Word-Branches
   haben eine Reichweite von +32766/-32768 Bytes.

44 Distanzwert erwartet
   Beispiel:   label: move.l  label(a2),d1   -> Error 44

45 Gültige Adresse erwartet

46 Vermisse Argument

47 Benötige numerisches Symbol

48 Distanz ist außerhalb der Section
   Beispiel:   bra  label   , wobei label in der aktuellen Section undefi-
   niert ist:  -> Error 48

49 Nur eine Distanz erlaubt
   Ausdruck darf nicht mehrere Distanzen beinhalten.
   Beispiel:   move.l  #(label1-label2)+(label3-label4),d0   -> Error 49

50 Vermisse Klammer

51 Ausdrucksstapel ist übergelaufen
   Maximal 128 Argumente sind pro Ausdruck möglich.

52 Adressen können nicht negiert werden

53 Distanz und Reloc-Adresse dürfen nicht im selben Ausdruck stehen
   Beispiel:   move.l  #(label1-label2)+label3,d0   -> Error 53

54 Fehler beim Shiften (nur abs. Ausdrücke oder Distanzen erlaubt)
   Beispiel: 1<<-1              -> Error 54
             label<<1           -> Error 54

55 Adressen können nicht multipliziert werden
56 Überlauf während Multiplikation
57 Adressen können nicht dividiert werden
58 Division durch Null
59 Logische Operation mit Adressen sind illegal
60 Für eine Distanz werden zwei Adressen benötigt
61 Adressen können nicht summiert werden

62 Schreibfehler

63 Kein Byte-, Word- oder Longword String
```

```
   Beispiel:   dc.d "XYZ"   -> Error 63

64 XREF kann nicht subtrahiert werden
   Erlaubte Operationen mit XREFs: ext + abs , abs + ext  und  ext - abs

65 Im absoluten Modus unmöglich
   Folgende Direktiven können im absoluten Modus nicht verwendet werden:
   ttl, code, cseg, data, dseg, bss, section, xref, nref, xdef, public

66 Unbekannter Fehler (Fataler Programmfehler)
   Teile des Assemblers oder seines Speicherbereichs wurden durch ein
   fehlerhaftes Programm, das zur selben Zeit lief, zerstört.

67 Externe Symbole sind im absoluten Modus nicht möglich
   Siehe Fehler 65.

68 Außer Reichweite
   Beispiel:   addq.l  #9,d1   -> Error 68

69 Assemblierungs abgebrochen
   Wird durch die FAIL Direktive hervorgerufen.

70 Vermisse ENDC/ENDIF

71 Vermisse den Makronamen

72 Vermisse ENDM

73 In einem Makro ist keine weitere Makrodefinition möglich

74 Überflüssiges ENDM

75 Überflüssiges ENDC/ENDIF

76 Im relativen Modus nicht möglich
   Folgende Direktiven können im relativen (relocatiblen) Modus nicht
   benutzt werden: org, file, load, trackdisk.

77 Parameterpuffer ist übergelaufen
   Makroparameter dürfen maximal 63 Zeichen enthalten.

78 Ungültiger REPT Zähler
   Der Startzähler für REPT sollte nicht negativ sein.

79 Kann Datei nicht erzeugen
   Vielleicht ist die Zieldiskette schreibgeschützt.

80 Für eine Referenztabelle muß ein Listing File existieren
   Der XREFS Schalter wurde ohne den LIST Schalter gesetzt.

81 Hier ist keine Adresse erlaubt
   Beipspiel:   ds.l label   -> Error 81

82 Symbol enthält ungültige Zeichen
   Siehe auch Fehlermeldung 22.

83 Quelltext ist zu lang (maximal 65535 Zeilen)
```

```
84 Fließkomma ohne die erforderlichen Mathe-Libraries ist nicht möglich
   Um Fließkommasymbole zu benutzen, müssen sich die folgenden Libraries in
   Ihren LIBS: Verzeichnis befinden:
      mathtrans.library, mathieeedoubbas.library, mathieeedoubtrans.library

85 Überlauf während Fließkommaberechnung
   Meißtens geschieht dies, wenn das Ergebnis eines Fließkommaausdrucks
   in einen Fließkommatyp niedrigerer Präzision konvertiert werden muß,
   z.B. nach FFP oder Single Precision.

86 Fließkommaausdruck enthält ungültigen Symboltyp
   Benutzen Sie keine relozierbaren Symbole in Fließkommaausdrücken.

89 Symboltyp darf mit SET nicht geändert werden
   Beispiel: symbol   set.d    3.14159265
             symbol   set.x    -0.1        -> Error 89
   Der Wert eines SET-Symbols ist variabel, aber nicht sein Typ!

90 LOAD, FILE und TRACKDISK können nicht gemischt verwendet werden
   Beispiel:  load $70000
              file "mycode"   -> Error 90

91 Near-Modus ist nicht aktiv
   Der Near-Modus muß, bevor man die INITNEAR Direktive verwendet kann, erst
   durch NEAR aktiviert werden.

92 Instruktion mit eingestellter MACHINE nicht möglich
   Die Instruktion existiert für einen anderen Prozessortyp, aber nicht für
   den gerade eingestellten. Benutzen Sie MACHINE um dies zu ändern.

93 Ungültiger Scale Faktor
   Beispiel:    move.w  (a1,d2*3)    -> Error 93
   Erlaubt sind die Faktoren 1, 2, 4 und 8.

94 Vermisse einen Operanden
   Beispiel:    move.l  (a0)+        -> Error 94

95 Diese Section existiert nicht
   Dieser Fehler wird durch die Angabe einer falschen Sections-Nummer in
   der NEAR-Direktive verursacht.

96 Ungültiger RORG Offset
   Der relative Offset darf keine Adresse vor der gerade übersetzten Adresse
   sein.

97 Immediate-Operand ist außerhalb seines erlaubten Bereichs
   Beispiel:    move.b   #$1234,d0   -> Error 97

98 Vermisse ein ENDR
   Offene REPT-Schleife beim Verlassen des Quelltextes, einerInclude-Datei
   oder eines Makros, entdeckt.

99 Überflüssiges ENDR
   Für dieses ENDR existiert keine passende REPT-Direktive.

100 Maximale REPT-Verschachtelungstiefe überschritten
```

Die maximale Verschachtelungstiefe ist 255.

101 Name existiert schon als Direktive
Es wurde ein Macro mit einem Namen definiert, der bereits von einer eingebauten Direktive oder Instruktion verwendet wird.

1.91 Entstehungsgeschichte / Literatur

Nachdem ich sechs Jahre lang mit Assemblern wie SEKA, AS (Aztec-C) und A68k gearbeitet habe, kam ich im Dezember 1991 zu dem Entschluß, daß ich einen neuen mächtigeren Assembler benötige. Ich habe erst einige Zeit darüber nachgedacht, ob ich mir nicht O.M.A. oder Devpac kaufe, aber eigentlich mag ich diese modernen Assembler mit einem integrierten Editor nicht besonders. Weitere Gründe für den Startschuß zur Entwicklung von PhxAss waren mein chronischer Geldmangel (da ich Student bin) und die Möglichkeit einen Assembler nach meinen persönlichen Wünschen zu schaffen (meistens unterstützt ein Assembler ja immer gerade das nicht, was man unbedingt benötigt :-).

Die Arbeiten an der ersten lauffähigen Version, V1.00, waren am 28.01.92 abgeschlossen. Von nun an konnte ich PhxAss dazu benutzen sich selber zu übersetzen (vorher mußte ich A68k verwenden). Es kostete mich mehr als ein Jahr und 23 Versionen um V3.00 zu erreichen und beinahe weitere zwei Jahre und 52 Versionen für V4.00 (natürlich war PhxAss in diesem Zeitraum nicht mein einziges Projekt).

Es folgt eine Liste meiner Hardware und Literatur, wodurch die Entwicklung von PhxAss ermöglicht wurde:

Hardware: Mein guter, alter A1000 (erste Version von '85) mit 68010 CPU, 2 MB Fast-RAM und eine 33 MB Harddisk.
(seit Dezember '93:) A4000, 68040, 6 MB RAM, 250 MB Harddisk.

Literatur: Motorola MC68000/68008/68010/68HC000 8-/16-/32-Bit Microprocessor 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)

1.92 Danksagungen

Den folgenden Personen, die PhxAss intensiv getestet und durch konstruktive Bug-Reports die Entwicklung beschleunigt haben, möchte ich meinen Dank aussprechen:

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.wwb.sub.de
Dave Dustin (NZ)	dave@eclipsnz.manawatu.gen.nz
Gunther Nikl	gnikl@informatik.uni-rostock.de
Wolf Wolfswinkel (NL)	W.Wolfswinkel@PObox.ruu.nl
Richard Körber	Shred@tfh.dssd.sub.org
Gregor Copoix	Logical@indigo.oche.de
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@goober.mbhs.edu

Schwedische Catalogs von:

Marcus Geelnard (S)	e4geeln@etek.chalmers.se

Italienische Catalogs von:
Simone Tellini (I)

Ein weiterer Dank geht, auch wenn sie jetzt liquidiert sind, an Commodore:

Danke für den einzigen Computer der heutigen Zeit, mit dem das Arbeiten wirklich noch Spaß macht :-)

1.93 Bekannte Fehler in Version V4.14

o Wenn die Instruktion xxxx während der Optimierung vollständig entfernt wird, erzeugt PhxAss einen illegalen Byte-Branch mit Null-Displacement:

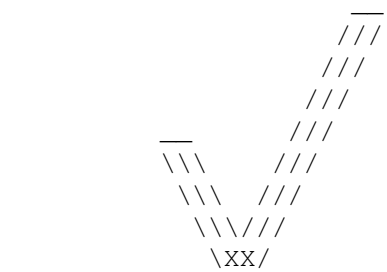
```
    Bcc.s  label
    xxxx
  label:
```

Dies geschieht jedoch nur, wenn der Optimierungsschalter 'M' gewählt wurde und xxxx ein 'MOVEM' ohne Register ist, oder wenn der Optimierungsschalter

```
  'S' gewählt wurde und xxxx ein 'ADDA/SUBA #0,An' oder 'LEA 0(An),An' ist.

o Die Vorwärts-Branch Optimierung (T-Schalter) kann nicht sämtliche Zeilen-
  adressen im Listing File, die sich dadurch verschoben haben könnten,
  korrigieren.

o Die folgenden Zeilen aus der original Commodore Include-Datei
  "exec/types.i" können nicht fehlerfrei übersetzt werden, und sollten
  daher wie folgt geändert werden:
  \@BITDEF  SET     1<<\3
            BITDEF0 \1,\2,F_,\@BITDEF
  Ändern in:
  BITDEF\@  SET     1<<\3
            BITDEF0 \1,\2,F_,BITDEF\@
  Ich würde wirklich zu gerne wissen, wer die großartige Idee hatte, ein
  Symbol zu definieren, das mit einer Ziffer beginnt :-(

o Vorsicht bei der Definition eines Labels direkt vor einer CNOP Anweisung!
  label1:
            CNOP     0,4
  label2:
  Leider kann PhxAss nur in Pass 1 zwichen label1 und label2 unterscheiden,
  während es in Pass 2 passieren kann, daß sich label1 mitverschiebt!
  Ich sehe momentan leider keine Lösung für dieses Problem... :((


Wenn noch irgendwelche Fehler oder Fragen auftauchen sollten, schreiben
Sie an :

                        Meine Adresse
```

1.94 Meine Adresse


```
SMail:                  Frank Wille
                        Auf dem Dreische 45
                        32049 Herford
                        DEUTSCHLAND

EMail:                  Frank@Phoenix.owl.de
                        (wird täglich nachgesehen)

                        fwille@TechFak.Uni-Bielefeld.de (gültig bis 10/95?)
                        (momentan nur ein oder zweimal im Monat nachgesehen)
```

```
A M I G A   F O R E V E R !
```