

Barfly

COLLABORATORS

	<i>TITLE :</i> Barfly		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		November 28, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Barfly	1
1.1	Barfly.guide	1
1.2	Barfly.guide/BI_CRIGHT	2
1.3	Barfly.guide/BI_REG	2
1.4	Barfly.guide/IN_PURPOSE	3
1.5	Barfly.guide/IN_SYSR	4
1.6	Barfly.guide/IN_INST	5
1.7	Barfly.guide/OT_UPD	5
1.8	Barfly.guide/OT_SUP	5
1.9	Barfly.guide/OT_HIST	6
1.10	Barfly.guide/OT_FUT	6
1.11	Barfly.guide/OT_ACK	7
1.12	Barfly.guide/ASoft	8
1.13	Barfly.guide/BDebugTop	8
1.14	Barfly.guide/UB_COMW	9
1.15	Barfly.guide/UB_REGW	11
1.16	Barfly.guide/REGW_Window	12
1.17	Barfly.guide/REGW_LocalMenus	14
1.18	Barfly.guide/REGW_PublicMenus	15
1.19	Barfly.guide/UB_FPUW	21
1.20	Barfly.guide/FPUW_Window	22
1.21	Barfly.guide/FPUW_LocalMenus	22
1.22	Barfly.guide/UB_DISSW	22
1.23	Barfly.guide/DISSW_Window	23
1.24	Barfly.guide/DISSW_LocalMenus	23
1.25	Barfly.guide/UB_MEMW	25
1.26	Barfly.guide/MEMW_Window	25
1.27	Barfly.guide/MEMW_LocalMenus	25
1.28	Barfly.guide/UB_COPPW	27
1.29	Barfly.guide/COPPW_Window	27

1.30	Barfly.guide/COPPW_LocalMenus	28
1.31	Barfly.guide/UB_STRUCTW	28
1.32	Barfly.guide/STRUCTW_Window	29
1.33	Barfly.guide/STRUCTW_Format	30
1.34	Barfly.guide/STRUCTW_LocalMenus	31
1.35	Barfly.guide/UB_SOURCEW	32
1.36	Barfly.guide/SOURCEW_Window	33
1.37	Barfly.guide/SOURCEW_LocalMenus	33
1.38	Barfly.guide/UB_SNOOPW	34
1.39	Barfly.guide/SNOOPW_Window	34
1.40	Barfly.guide/SNOOPW_LocalMenus	34
1.41	Barfly.guide/UB_BREAKW	35
1.42	Barfly.guide/BREAKW_Window	35
1.43	Barfly.guide/BREAKW_LocalMenus	35
1.44	Barfly.guide/UB_WATCHW	36
1.45	Barfly.guide/WATCHW_Window	37
1.46	Barfly.guide/WATCHW_LocalMenus	37
1.47	Barfly.guide/UB_CHECKW	38
1.48	Barfly.guide/CHECKW_Window	38
1.49	Barfly.guide/CHECKW_LocalMenus	38
1.50	Barfly.guide/UB_ARGUMENTS	39
1.51	Barfly.guide/UB_TECHINFOS	40
1.52	Barfly.guide/UB_CONFIG	41
1.53	Barfly.guide/CO_TOOLTYPES	42
1.54	Barfly.guide/CO_BARFLYFD	42
1.55	Barfly.guide/CO_CONFIGCMDS	42
1.56	Barfly.guide/CC_REGW	43
1.57	Barfly.guide/CC_FPUW	43
1.58	Barfly.guide/CC_DISSW	44
1.59	Barfly.guide/CC_MEMW	45
1.60	Barfly.guide/CC_COPPW	45
1.61	Barfly.guide/CC_STRUCTW	46
1.62	Barfly.guide/CC_SOURCEW	46
1.63	Barfly.guide/CC_BREAKW	47
1.64	Barfly.guide/CC_WATCHW	47
1.65	Barfly.guide/CC_CHECKW	47
1.66	Barfly.guide/CC_SNOOPW	47
1.67	Barfly.guide/CC_INFOW	48
1.68	Barfly.guide/CC_OTHERW	48

1.69	Barfly.guide/CM_MISC	49
1.70	Barfly.guide/CM_TASKSTACK	50
1.71	Barfly.guide/CM_TASKPRI	50
1.72	Barfly.guide/CM_SETBREAK	50
1.73	Barfly.guide/CM_CLICKBREAK	50
1.74	Barfly.guide/CM_SHOWMEM	51
1.75	Barfly.guide/CM_DEFCOMMAND	51
1.76	Barfly.guide/CM_AUTODOCDIR	52
1.77	Barfly.guide/CM_AUTODOCALIAS	52
1.78	Barfly.guide/CM_AREXXPATH	52
1.79	Barfly.guide/CM_AREXXINPUT	52
1.80	Barfly.guide/CM_AREXXOUTPUT	53
1.81	Barfly.guide/CM_AREXXCMD	53
1.82	Barfly.guide/CM_EXECMD	53
1.83	Barfly.guide/CM_LOADINCLUDE	53
1.84	Barfly.guide/CM_ADDSTRUCT	53
1.85	Barfly.guide/CM_CLICK2FRONT	54
1.86	Barfly.guide/CM_CENTERW	54
1.87	Barfly.guide/CM_SCREENFRONT	54
1.88	Barfly.guide/CM_OPENSSCREEN	54
1.89	Barfly.guide/CM_OPENPSCREEN	55
1.90	Barfly.guide/CM_SCREENFONT	55
1.91	Barfly.guide/CM_QUIETEX	55
1.92	Barfly.guide/CM_DISXP	55
1.93	Barfly.guide/CM_TRACEBREAK	55
1.94	Barfly.guide/CM_CRASHEDTASK	56
1.95	Barfly.guide/CM_CATCHHIT	56
1.96	Barfly.guide/CM_CACHEFILE	56
1.97	Barfly.guide/CM_POPPATH	56
1.98	Barfly.guide/CM_AUTOACT	56
1.99	Barfly.guide/CM_NOBREAKTERRORS	57
1.100	Barfly.guide/UB_AREXX	57
1.101	Barfly.guide/UB_HOWTOUSE	62
1.102	Barfly.guide/BAsmTop	64
1.103	Barfly.guide/UA_ASSEMBLER	65
1.104	Barfly.guide/UA_SYNTAX	65
1.105	Barfly.guide/UA_DATATYPES	69
1.106	Barfly.guide/UA_OPERATIONS	71
1.107	Barfly.guide/UA_INST	72

1.108Barfly.guide/AI_HUNK	72
1.109Barfly.guide/HU_SECTION	73
1.110Barfly.guide/HU_CODE	74
1.111Barfly.guide/HU_DATA	75
1.112Barfly.guide/HU_BSS	75
1.113Barfly.guide/HU_CSEG	76
1.114Barfly.guide/HU_DSEG	76
1.115Barfly.guide/HU_IDNT	76
1.116Barfly.guide/HU_IDENTIFY	76
1.117Barfly.guide/HU_BDEBUGARG	77
1.118Barfly.guide/HU_SMALLDATA	77
1.119Barfly.guide/HU_XREF	77
1.120Barfly.guide/HU_XDEF	78
1.121Barfly.guide/HU_GLOBAL	78
1.122Barfly.guide/HU_PUBLIC	78
1.123Barfly.guide/HU_OUTPUT	78
1.124Barfly.guide/HU_OBJFILE	79
1.125Barfly.guide/HU_EXEOBJ	79
1.126Barfly.guide/HU_LINKOBJ	79
1.127Barfly.guide/HU_ORG	79
1.128Barfly.guide/HU_ADDSYM	79
1.129Barfly.guide/HU_DEBUG	80
1.130Barfly.guide/AI_SYMBOL	80
1.131Barfly.guide/SY_CARGS	80
1.132Barfly.guide/SY_RS	80
1.133Barfly.guide/SY_SO	81
1.134Barfly.guide/SY_FO	81
1.135Barfly.guide/SY_RSRESET	82
1.136Barfly.guide/SY_RSSET	82
1.137Barfly.guide/SY_CLRSO	82
1.138Barfly.guide/SY_CLRFO	83
1.139Barfly.guide/SY_SETSO	83
1.140Barfly.guide/SY_SETRS	83
1.141Barfly.guide/SY_SETFO	83
1.142Barfly.guide/SY_RSVAL	83
1.143Barfly.guide/SY_SOVAL	84
1.144Barfly.guide/SY_FOVAL	84
1.145Barfly.guide/AI_DATA	84
1.146Barfly.guide/DA_ALIGN	85

1.147Barfly.guide/DA_CNOP	85
1.148Barfly.guide/DA_PAD	85
1.149Barfly.guide/DA_QUAD	86
1.150Barfly.guide/DA_EVEN	86
1.151Barfly.guide/DA_ODD	86
1.152Barfly.guide/DA_DC	86
1.153Barfly.guide/DA_DB	87
1.154Barfly.guide/DA_DW	87
1.155Barfly.guide/DA_DL	87
1.156Barfly.guide/DA_UB	87
1.157Barfly.guide/DA_UW	87
1.158Barfly.guide/DA_UL	88
1.159Barfly.guide/DA_SB	88
1.160Barfly.guide/DA_SW	88
1.161Barfly.guide/DA_SL	88
1.162Barfly.guide/DA_PB	88
1.163Barfly.guide/DA_PW	89
1.164Barfly.guide/DA_PL	89
1.165Barfly.guide/DA_NB	89
1.166Barfly.guide/DA_NW	89
1.167Barfly.guide/DA_NL	89
1.168Barfly.guide/DA_DS	90
1.169Barfly.guide/DA_DSB	90
1.170Barfly.guide/DA_DCB	90
1.171Barfly.guide/DA_BLK	91
1.172Barfly.guide/DA_ASCII	91
1.173Barfly.guide/DA_CSTRING	91
1.174Barfly.guide/DA_DSTRING	91
1.175Barfly.guide/DA_PSTRING	92
1.176Barfly.guide/DA_ISTRING	92
1.177Barfly.guide/DA_BITSTREAM	92
1.178Barfly.guide/DA_SPRINTX	92
1.179Barfly.guide/AI_LISTING	93
1.180Barfly.guide/LI_LIST	93
1.181Barfly.guide/LI_NOLIST	94
1.182Barfly.guide/LI_PRINTX	94
1.183Barfly.guide/LI_LISFILE	94
1.184Barfly.guide/AI_STRUCTURE	94
1.185Barfly.guide/ST_MACRO	95

1.186Barfly.guide/ST_ENDM	95
1.187Barfly.guide/ST_MEXIT	95
1.188Barfly.guide/ST_FAIL	96
1.189Barfly.guide/ST_END	96
1.190Barfly.guide/ST_IF	96
1.191Barfly.guide/ST_IFD	96
1.192Barfly.guide/ST_IFND	96
1.193Barfly.guide/ST_IFV	97
1.194Barfly.guide/ST_IFNV	97
1.195Barfly.guide/ST_IFMACROD	97
1.196Barfly.guide/ST_IFMACROND	97
1.197Barfly.guide/ST_IFCMACROD	97
1.198Barfly.guide/ST_IFCMACROND	98
1.199Barfly.guide/ST_IFC	98
1.200Barfly.guide/ST_IFNC	98
1.201Barfly.guide/ST_IFCC	98
1.202Barfly.guide/ST_ELSE	98
1.203Barfly.guide/ST_ELSEIF	99
1.204Barfly.guide/ST_ENDC	99
1.205Barfly.guide/ST_ENDIF	99
1.206Barfly.guide/ST_REPEAT	99
1.207Barfly.guide/ST_REPT	99
1.208Barfly.guide/ST_PROCSTART	99
1.209Barfly.guide/ST_PROCEnd	100
1.210Barfly.guide/AI_FILE	100
1.211Barfly.guide/Fl_INCDIR	100
1.212Barfly.guide/Fl_INCPATH	101
1.213Barfly.guide/Fl_INCLUDE	101
1.214Barfly.guide/Fl_INCLUDE2	101
1.215Barfly.guide/Fl_INCBIN	101
1.216Barfly.guide/Fl_INCBIN2	102
1.217Barfly.guide/Fl_IBYTES	102
1.218Barfly.guide/Fl_DSBIN	102
1.219Barfly.guide/Fl_DOSCMD	102
1.220Barfly.guide/Fl_PURE	103
1.221Barfly.guide/AI_MISC	103
1.222Barfly.guide/MI_TRASHREG	103
1.223Barfly.guide/MI_SUPER	103
1.224Barfly.guide/MI_MCXXX	103

1.225Barfly.guide/MI_BOPT	104
1.226Barfly.guide/AI_META	110
1.227Barfly.guide/ME_MB	110
1.228Barfly.guide/ME_MW	110
1.229Barfly.guide/ME_ML	110
1.230Barfly.guide/ME_MQ	111
1.231Barfly.guide/ME_XOR	111
1.232Barfly.guide/ME_XORI	111
1.233Barfly.guide/ME_BHS	111
1.234Barfly.guide/ME_BLO	111
1.235Barfly.guide/UA_MACROS	112
1.236Barfly.guide/UA_HMACROS	115
1.237Barfly.guide/HM_REG	115
1.238Barfly.guide/HM_BRANCH	116
1.239Barfly.guide/HM_FOR	116
1.240Barfly.guide/HM_NEXT	116
1.241Barfly.guide/HM_IF	116
1.242Barfly.guide/HM_ELSE	117
1.243Barfly.guide/HM_ENDIF	117
1.244Barfly.guide/HM_WHILE	117
1.245Barfly.guide/HM_ENDWHILE	118
1.246Barfly.guide/HM_CALL	118
1.247Barfly.guide/HM_RETURN	118
1.248Barfly.guide/HM_DEF	119
1.249Barfly.guide/HM_ENDDEF	119
1.250Barfly.guide/HM_LET	119
1.251Barfly.guide/UA_SYMBOLS	120
1.252Barfly.guide/UA_OPTIMIZING	121
1.253Barfly.guide/OP_DIRECT	121
1.254Barfly.guide/OP_ADDRESS	123
1.255Barfly.guide/OP_OPTIMIZE	125
1.256Barfly.guide/OP_REGISTER	129
1.257Barfly.guide/OP_HOWDOESITWORK	131
1.258Barfly.guide/OP_PROBLEMS	132
1.259Barfly.guide/UA_PRE	133
1.260Barfly.guide/UA_CLI	133
1.261Barfly.guide/UA_AREXX	135
1.262Barfly.guide/UA_COMP	136
1.263Barfly.guide/UA_LITERATURE	137
1.264Barfly.guide/UA_SOFTWARE	138
1.265Barfly.guide/UA_EA	138
1.266Barfly.guide/UA_OPCODES	141

Chapter 1

Barfly

1.1 Barfly.guide

Barfly 1.0

An Intuition controlled Debugger and Optimizing Assembler

Copyright (c) 1989-94 Ralph Schmidt

- Shareware -

Chapters for all users...

Copyright	Your rights.
Registration	How to become a registered user.

Introduction...

Purpose	What is Barfly made for?
System requirements	Which computer can run Barfly?
Installation	How to install Barfly.

Using BDebug...

The Debugger	Debugger Documentation
--------------	------------------------

Using BAsm...

The Assembler	Assembler Documentation
---------------	-------------------------

Includes & Linker...

Additional SW	Where to get the include and Linker
---------------	-------------------------------------

Other topics...

Updates	How to get updates.
Support	How to reach the author.
History	History of Barfly.
Future	Future of Barfly.

Acknowledgements

The author wishes to thank...

1.2 Barfly.guide/BI_CRIGHT

Basic Informations

Copyright and other legal stuff

=====

Copyright (c) 1989-94 Ralph Schmidt

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

No guarantee of any kind is given that the programs described in this document are 100% reliable. You are using this material at your own risk. The author can not be made responsible for any damage which is caused by using these programs.

Permission is granted to include this package in Public-Domain collections, especially in Fred Fishs Amiga Disk Library (including CD-ROM versions of it). The distribution file may be uploaded to Bulletin Board Systems or FTP servers. If you want to distribute this program you must use the original distribution archive Barfly.lha.

None of the programs may be included or used in commercial programs unless by written permission from the author.

None of the programs may be modified in any way. This especially includes changing the copyright notices or removing any of the Shareware restrictions.

None of the programs may be used on any machine which is used for the research, development, construction, testing or production of weapons or other military applications. This also includes any machine which is used for training persons for any of the above mentioned purposes.

None of the programs may be used by more than the registered owner.

1.3 Barfly.guide/BI_REG

Registration

=====

As you may have noticed in the copyright I'm working for five years at Barfly. It has always consumed and will continue to consume a large amount of my time.

I cannot afford just working for fun. Thus, I decided to release Barfly as Shareware. I already tried to release Barfly as a commercial product but the story behind it is more than sad. To sum it...german Amiga software companys aren't worth any time...they suck. Some people may think the price is too high for a Shareware product but i think that BAsm is as powerful as the 2 main available commercial Assemblers...if not more powerful if you compare the speed and the optimize functions;there's no commercial Debugger available that can compete with BDebug. I've used Barfly myself for commercial Amiga applications. Z3-Fastlane device,CDRive,SCSIConfig,...

The unregistered version of Barfly pops up the About requester at the start and has some functions disabled:

Assembler:

- o only 8192Bytes large code possible
- o the Section commands aren't available

Debugger:

- o Only 1 Window per Object
- o Enforcer Catch not available
- o Task Catch not available
- o Crashed Task Catch not available
- o Limited Step count (about 150-200 Steps)

Registered users will be shipped a disk with the newest public release of Barfly, along with a personalized, so-called "keyfile". It enables all the missing features and disables the Shareware reminders. This keyfile will work with all future releases of Barfly, so you can simply download the latest version from your local bulletin board without having to wait weeks for your update passing through the slow mail channels. The keyfile must not be distributed in any way.

The fee for a Barfly registration is

- 70.- DM (D-Mark),
- 70.- SFr (Schweizer Franken),
- 230.- FF (French Francs),
- 50.- US\$ (US Dollar)

The fastest, cheapest and easiest way to register is put the money together with the filled registration form into a letter and send it to

Please allow 2-5 weeks delivery for the registered version.

Ralph Schmidt
Kleiner Hellweg 4
33154 Salzkotten
FR Germany
Phone: +49-5258-5637
E-Mail: laire@uni-paderborn.de
Irc: Laire on #amiga,#amigager

1.4 Barfly.guide/IN_PURPOSE

Introduction

Purpose

=====

BDebug is an Intuition controlled multi-task system Debugger for OS 2.04 and newer.

You can use BDebug to debugging your programs, catching tasks, reseach enforcer hits, follow Source-Level Informations and other advanced functions. The Debugger supports assemblers,SAS C,Dice and GCC.

Some of BDebug's features are:

- * font-sensitive, resizable and Style Guide compliant GadTools GUI
- * Object-Oriented that results in a low learning curve
- * Supports 68000...68060 and the FPU's
- * Can debug multiple tasks at the same time
- * Not limited by the amount of window objects.
- * highly configurable
- * keyboard support

BAsm is a very fast optimizing Assembler for OS 2.04 and newer.

Some of BAsm's features are:

- * 68000-68060,6888x
- * Very Fast
- * Include and Incbin Cache
- * Strong Optimizer with Multi-Pass Optimizing
- * High Level Macros
- * ARexx
- * Supports OS 2.04 and OS 3.0 Hunks
- * SAS D1 Source Level Format

1.5 Barfly.guide/IN_SYSR

System requirements

=====

Barfly requires Amiga operating system version 2.04 or better.

Kickstart 1.3 is not supported; this operating system is considered obsolete.

Barfly requires at least one megabyte of RAM to run. A hardisk or a faster CPU is not required but increase performances and comfort, of course.

1.6 Barfly.guide/IN_INST

Installation
=====

It is really easy to install Barfly:

1. Copy the the binary and the icon (called "BDebug" and "BDebug.info") to any directory.
2. Copy the the binary and the icon (called "BAsm" and "BAsm.info") to any directory.

Then copy the supplied configuration files from "s/Barfly/#?" to "S:" directory of your system partition or create env:Barflypath with the path to a directory that contains Barfly/#?.

1.7 Barfly.guide/OT_UPD

Other topics

Updates
=====

Whenever a new release of Barfly gets released, I will post some information in the appropriate newsgroups of some electronic networks. The new archive will soon be available on many bulletin boards and on all AmiNet FTP servers. Major releases will also come with some PD disks, especially on Fred Fish's collection.

As mentioned above, registered users will neither need a new keyfile nor a special personalized program version. They can use all new features immediately.

1.8 Barfly.guide/OT_SUP

Support

=====

If you have some questions, comments, suggestions or even flames, please feel free to contact me at one of the following addresses. If you send your letter via e-mail, there's a good chance for getting a quick reply.

Snailmail: Ralph Schmidt
Kleiner Hellweg 4
33154 Salzkotten
FR Germany

Phone: +49-5258-5637

E-Mail: laire@uni-paderborn.de
Irc: Laire on #amiga, #amigager

1.9 Barfly.guide/OT_HIST

History

=====

- * 0.0 - 1.0
not released

1.10 Barfly.guide/OT_FUT

Future

=====

Here are some ideas for future versions of BDebug:

- * Full Source Level support for GCC and perhaps SAS.
 - * Better BDebug Arexx support...the current one is a bad excuse.
 - * Mungwall Trace methods.
 - * Automatic Refresh of some Windows(Task Window...)
 - * Amigaguide file mode in the autodocs functions.
 - * Highlight changed registers
 - * Better documentation.
 - * BAsmOption for the easy BAsm options configuration.
-

* Other things i'm too lazy to mention now.

Important:

There is absolutely NO guarantee that these features will ever be implemented. So don't be disappointed, if they aren't in the next version.

1.11 Barfly.guide/OT_ACK

Acknowledgments

=====

Thanks must go to:

- Dirk Leschner, Frank Jakobs

for being my best friends

- Matthias Scheler

for his manual design and Filer.

- S.Schaem, Børge Nøst, Alexis Wilke, Michael B. Smith, Marc Heuler

for their superb betatesting efforts

- Mike Schwartz

for a lot suggestions to improve basm. Sad he has left the Amiga community 2 years ago.

- Stefan Becker

for Toolmanager and being a nice guy.

- Christoph Wolf

for DynamiCache and being a nice guy.

- Brian Cervený (Redwine)

for Grapevine and being a nice guy (Really ?).

- All my IRC friends.

for many great hours. Thanks!

Andrew Denton (Guardian), Kenneth Dyke (Nyx), Bill Coldwell (Cryo), Brian Cervený (Redwine), Joseph Hillenburg (xterm), Scott Ellis (ScottE), Chris Wichura (Caw), John Wiederhirm (John_W), Mike Schwartz (mykes), Markus Illenseer (ill), Petra Zeidler (stargazer), Michael van Elst (mlelstv), Holger Lubitz (holgi), Ralph Babel (rbabel), Seth Harman (Budha) and a lot guys i haven't mentioned here.

- Chris Schneider and Urban D. Mueller
for some suggestions 2-3 years ago and installing Aminet.
- Michael "billy" Böhnisch
for his cleanup on MakeBarflyFD 2-3 years ago.
- Steve Wright
for designing the icons 2 years ago.

And of course to all the other Beta testers and registered users.

1.12 Barfly.guide/ASoft

The V40 Includes can be ftp'ed from the FTP-Server.

ftp.rz.uni-wuerzburg.de: pub/amiga/frozenfish/bbs/com

A superb Linker "lk" by Alex Wilke should be soon availble on Aminet.

1.13 Barfly.guide/BDebugTop

BDebug 1.0

An Intuition controlled Debugger

Copyright (c) 1989-94 Ralph Schmidt

- Shareware -

Using BDebug...

Command Window	Debug Methods
Register Window	Register Window Object
FPU Window	FPU Window Object
Disassembler Window	Disassembler Window Object
Memory Window	Memory Window Object
Copper Window	Copper Window Object
Struct Window	Struct Window Object
Source Window	Source Window Object
Snoop Window	Snoop Window Object
Breakpoint Window	Breakpoint Manager Window Object
Watchpoint Window	Watchpoint Manager Window Object
Checksum Window	Checksum Manager Window Object
Arguments	Requester Arguments
Technicals	Technical Details

Configurations	Configuration Details
Arexx	Arexx
Problem Analysis	Problem Analysis

1.14 Barfly.guide/UB_COMW

Usage of BDebug

The Command Window

=====

Debugger Philosophy

=====

BDebug is a multitasking Debugger that supports the Motorola processors 68000...68060 and 68881...68882. The Debugger allows to debug unlimited tasks parallel. Because of the Debugger's complexity BDebug was designed in an object-oriented way to allow an easy and comfortable way to use it. The register window REGWindow is the Root class of the task object that can be expanded by several subclass windows. Every subclass window has private menus and inherits the public menus of its father object.

Debug Methoden

=====

The Debugger offers a variety of different Debug methods that can be activated by menu or gadget.

Debug Task

.....

is used to select a task you wanna debug. If you doubleclick on a task a REGWindow and a couple of information windows opens. Which type and how many are opened depends on the current configuration. After the task could be stopped the contents of the REGWindow and all other information windows gets refreshed. If the task is in the Wait state the task is stopped when it gets a signal.

Task Listview Layout

Taskaddress	&	Priority	&	Status	&	[!]Name
-------------	---	----------	---	--------	---	---------

A process is marked by ! at the beginning of the name.

You should know what task you can stop and what kind of task should never be stopped. For example the Input.device should never be stopped.

Debug File

.....

is used to load and stop a program. This function is equal to the bdebug cli startup with the exception that you can enter the parameter in a requester. If no error occur all configurated windows are opened and the PC stops at the defined programstart breakpoint that normally points to the first command of the program.

Debug Next Task

.....

is used to debug the next task that is opened. The Debugger waits until another task is created by AddTask and a couple of information windows opens. Which type and how many are opened depends on the current configuration. After the new task was caught the pc points to the beginning of the task and Catch Next Task is disactivated. To catch a program that is started from the WV you have to use Debug Next Task to catch the WB Startup Task WBL that starts the program. Now you have to activate Debug Next Task again and let the current task run. After a short time the task WBL ends and the program's task is caught.

You should avoid to start a new task between the 2 Debug Next Task phases because it's easy to catch the wrong one.

You should notice that AddTask is patched and points to a new routine. Thus you should be careful with programs that also patch AddTask. Furthermore it's useful know in what sequences these patches have to be removed. The Debugger can only be closed when all patched system function that were installed after the Debugger was started are removed.

If you start a program in the shell without c:Run no new task is created. Instead the program is run as a subroutine in shell's task so you can't catch the task that easy.

Debug Crashed Task

.....

is used to catch tasks that crash so you track down the bug location a lot easier. If the system itself doesn't run anymore you shouldn't expect that bdebug still runs because it depends on a working system. If a task crashes and the option Debug Crashed Task is activated a couple of information windows opens. Which type and how many are opened depends on the current configuration. Task Held Requester.

Catch Enforcer Hit

.....

is used to tell the Debugger to stop the task it controlls when an enforcer hit happens. Unfortunately the Debugger can't stop the task at exactly the same location where the hit happened. Mostly the hit command is 1-2 instructions above the stopped task's PC.

This function needs Enforcer V37.x by M. Sinz and it must be installed before BDebug is started. Please read the documentation.

Select Display Mode

This function allows you to choose the Screen Mode for the Debugger.

How to start ?
 =====

If you only want to debug a program you have to start bdebug with the program's name and parameters or by using Debug Program in the command window. Another method is to move a program's icon on the command window or specify BDebug as in the icon as the DefaultTool.

Name:

BDEBUG - The CLI Startup

Synopsis:

BDEBUG [?] [<Program> [Argument]]

Function:

BDebug activates the Debugger, loads and stops the optional entered program. If it can find a local config file with the suffix *.bdebug it loads it.

Inputs

- * ? shows an information message
- * program name If no program name is entered BDebug looks for env:BDebugProgram and loads the program instead that the env: points to.
- * argument line of the program. If there are spaces in parameters you have to enclose the argument with "".

1.15 Barfly.guide/UB_REGW

Usage of BDebug

The Register Window
 =====

The Register Window
 Local Menus

The Register Window
 Local Menus

Public Menus

Public Menus

1.16 Barfly.guide/REGW_Window

Register Window

The register window is the most important control layer of the Debugger and every debugged task has one. You can link unlimited other information windows to the REGWindow or you can tell the Debugger to give up controlling the task. In the title line of the window you can see the ID number of the task, so you can recognize what information window belongs to this task. Furthermore the title line also contain the task address, the state, RUN, WAIT or STOP and the name of the task. The task is in the RUN state if the task has the controll at the moment instead of the traphandler; the task is in the STOP state when the task waits and the traphandler controlls what happens. The task is the WAIT state only when the Debugger has to wait to catch a task by a Debug Task. In the upper area of the window you see the normal data and address registers where the address register also have additional information fields. To change a register or to watch memory where the register points to you only need to doubleclick on the register or on the register's memory contents. Furthermore you may change other usermode registers by this method. Supervisor register can't be changed because that makes not much sense for a system Debugger.

68000 Registers

D0=xxxxxxxx yyyy A0=xxxxxxxx Arg1 Arg2 [!]

D1=xxxxxxxx yyyy A1=xxxxxxxx Arg1 Arg2 [!]

D2=xxxxxxxx yyyy A2=xxxxxxxx Arg1 Arg2 [!]

D3=xxxxxxxx yyyy A3=xxxxxxxx Arg1 Arg2 [!]

D4=xxxxxxxx yyyy A4=xxxxxxxx Arg1 Arg2 [!]

D5=xxxxxxxx yyyy A5=xxxxxxxx Arg1 Arg2 [!]

D6=xxxxxxxx yyyy A6=xxxxxxxx Arg1 Arg2 [!]

D7=xxxxxxxx yyyy A7=xxxxxxxx Arg1 Arg2 [!]

USP=xxxxxxxx SSP=xxxxxxxx PC=xxxxxxxx SR=xxxx

* [!] shows if the address register points on an odd address.

* if the address register points on an illegal memory area the char * is shown in Arg1 and Arg2 to avoid crashes by reading non readable io-addresses. You can config the readable memory areas.

* if the address register points on the following system structures the name of the Node is shown in Arg1 and the Name of the

structure in Arg2.

- * Library
- * Device
- * Port
- * Task
- * Resource
- * MemHead

- * if the address register points to the custom chip area the name of the register is shown in Arg1 and the ID-Word CUSTOM is shown in Arg2. Custom register map is \$dff000-\$dff200.
- * if the address register points to a symbol the symbol and the contents is shown.
- * Otherwise 8Bytes of the memory are shown where the register points to. The 8 Bytes are shown hexadecimal in Arg1 and ascii in Arg2.

68010 Registers

VBR=xxxxxxxx SFC=xxxxxxxx DFC=xxxxxxxx

68020 Registers

VBR=xxxxxxxx SFC=xxxxxxxx DFC=xxxxxxxx

MSP=xxxxxxxx ISP=xxxxxxxx CACR=xxxxxxxx CAAR=xxxxxxxx

68030 Registers

VBR=xxxxxxxx SFC=xxxxxxxx DFC=xxxxxxxx

MSP=xxxxxxxx ISP=xxxxxxxx CACR=xxxxxxxx CAAR=xxxxxxxx

CRP=xxxxxxxxxxxxxxxxxxxxx SRP=xxxxxxxxxxxxxxxxxxxxx

TT0=xxxxxxxx TT1=xxxxxxxx TC=xxxx PSR=xxxxxxxx

68040 Registers

VBR=xxxxxxxx SFC=xxxxxxxx DFC=xxxxxxxx

MSP=xxxxxxxx ISP=xxxxxxxx CACR=xxxxxxxx

URP=xxxxxxxx SRP=xxxxxxxx TC=xxxx PSR=xxxxxxxx

ITT0=xxxxxxxx ITT1=xxxxxxxx DTT0=xxxxxxxx DTT1=xxxxxxxx

68060 Registers

VBR=xxxxxxxx SFC=xxxxxxxx DFC=xxxxxxxx

```

CACR=xxxxxxxx PCR=xxxxxxxx BUSCR=xxxxxxxx

URP=xxxxxxxx SRP=xxxxxxxx TC=xxxx PSR=xxxxxxxx

ITT0=xxxxxxxx ITT1=xxxxxxxx DTT0=xxxxxxxx DTT1=xxxxxxxx

```

Type Information

```

xxxxxxxx [Symbol] Mnemonic operand1[,...]

(EA) : [Address1=Contents]...[Address2=Contents]

```

In the (EA) line you can see the addresses and their contents the current command accesses. The contents of illegal addresses aren't shown.

1.17 Barfly.guide/REGW_LocalMenus

Local Menu

* Close Window

closes the REGWindow, all connected windows and disactivates the Debugger for this task. To deactivate the Debugger you have to choose if the task should keep running so it's the task's business to stop. Furthermore you can end the task by runing the cleanup routine of the task or just removing the task from the list but this can cause sideeffects you can't always oversee. If the task is a process then the Remove option is equal to the Cleanup option. If it's only a task the Cleanup option is equal to the Cleanup option. Beware that the Remove option doesn't free any resources of the task.

You should really know what you're doing if you for example remove a task from the system.

* ZOOM Windows

expands all windows of the task.

* Log File

activates or disactivates the logging of the register and PC changes.

* Big View

shrinks the REGWindow to a 68000 register layout or expands it back to the full layout.

* Open DissWindow

opens a DissWindow with the configured dimensions.

* Open MemWindow

opens a MemWindow with the configured dimensions.

* Open FPUWindow

opens a FPUWindow with the configured dimensions. The menu is only available if a FPU is installed.

* Open BreakWindow

opens a BreakpointWindow with the configured dimensions.

* Open CoppWindow

opens a CoppWindow with the configured dimensions.

* Open StructWindow

opens a StructWindow with the configured dimensions.

* Open SnoopWindow

opens a SnoopWindow with the configured dimensions.

* Open WatchWindow

opens a WatchpointWindow with the configured dimensions.

* Open ChecksumWindow

opens a ChecksumWindow with the configured dimensions.

* Save Window Settings

saves the positions and count of all window the current task controls. The saved file then contains the appropriate commands you have to enter yourself into the configuration file. Because of the Debugger's window concept it makes no sense to save a full configurations file.

1.18 Barfly.guide/REGW_PublicMenus

Public Menus

* Step 1

runs the current command and stops the task afterwards.

* Step X

runs X commands and stops the task afterwards.

* Step Debug Line

runs commands until the PC meets another source line. If the PC is outside the program or if no debug informations are available a single step is used. The command enters subroutines.

* Trace Debug Line

is similar to Step Debug Line with the exception that it runs subroutines.

* Trace over Calls

runs the current command or subroutine and stops the task afterwards. Depending on the configuration and the memory area a breakpoint or single steps are used. If a crash happens in certain program parts you should remove the command Tracebreak from the configuration.

You should avoid to use Tracebreak in Libraries and Devices that are located in the ram. If another task accesses the routine at the same time you can expect an illegal exception.

Some Amiga MMU Setups don't like that programs write to the kickstart rom. For example breakpoints.

* Trace X over Calls

runs X commands or subroutines and stops the task afterwards. Depending on the configuration and the memory area a breakpoint or single steps are used.

* Trace Work

is similar to the command Trace over Calls with the exception that all commands are run. This function is useful to trace loops.

Example:

```
        moveq      #10,d0
0$:     dbra       d0,0$
```

If you use the function on the command dbra the Debugger sets a breakpoint after the dbra and runs the task. It drops back to single step when it hits a Jmp, bra, rts

* Trace over OS-Calls

runs the current command or the OS function and stops the task afterwards.

* Trace on Flow

stops the task when a PC direction occurs. This means the PC is stopped when it hits a Jsr, Jmp, bcc, rts

* Trace on Adress

runs the task until the PC is equal to the entered address. This function is not very fast because the task is running in single step mode and after each instruction the PC is compared with the Adress.

* Trace out of OS

runs the task until the PC is outside of the kickstart. This functions is useful when you catch a task inside the OS and you wanna get as fast as possible back to the program's code. It works simular as Trace on Adress.

You shouldn't use this command if your task only runs in the kickstart.

* (PC)++

jumps over the current command. Useful to jump over Illegal breakpoints that you can use for debugging purposes in your program.

* PC-2

subtract 2 Bytes from the PC.

* Write Nop

overwrites the current command with a Nop.

* Write Illegal

overwrites the current command with an Illegal.

* Run Task

runs the task and only stops on exceptions.

* Run Watched Task

runs the task in trace mode and stops when a WatchPoint condition is true. If there are no watchpoints the command behaves like Run Task.

* Run History Task

runs the task in trace mode and saves the registers each step into the history stack.

* Stop Task

stops the task.

* Send Signal

sends a signal to the task. Default Signal is CTRL-C = 12

* Undo Level

sets the undobuffer's depth.

* Undo

undos the last changes in the registerframe.

* View Refresh

activates and deactivates the copperlist refresh after each trace operation. This function is helpful if you debug programs that install own copperlists.

* Show (EA)

activates and deactivates the output of the address and address contents that are access by the current assembler command.

* Symbol

activates and deactivates the use of symbols in the REGWindow.

* Delete Symbols

erases all symbols of the task.

* Copy Symbols

can copy a symbol list of the task to a different task. This function is helpful if you a task is started from another task and you wanna keep the symbol list.

* Load Symbols

loads the symbols of a program where you can select an alternative process's segmentlist for calculating the symbol and debug informations. Normally you choose the same process but sometimes it's helpful to select a different process. For example if the task you debug is created in a program you have to choose the program's task to get the correct symbol addresses.

* Set Hunklist

sets a new segment list for the SourceWindow and some other hunk related functions. Because the position in the SourceWindow depends on the segments it's sometimes helpful if you load new symbols and debug informations for this task. If you load an alternative Hunklist by selecting a custom task when you use Load

Symbols this routine is called automatically.

* Reset Hunklist

removes the alternative hunklist.

* Show Value

shows the value of an argument.

* Show Last Exception

shows the last exception.

* Open Task Window

opens a window to show the task structure of the task.

* Open System Window

opens a window to show the ExecBase structure.

* Open Proces Window

opens a window to show the process structure of the process.

* Open CLI Window

opens a window to show the cli structure of the process.

* Open Hunk Window

opens a window to show the hunks of the process.

* Open Symbol Window

opens a window to show the symbols of the process. If you doubleclick on a symbol you get the hunk where the symbol is located.

* Open Library Window

opens a window to show the libraries. If you doubleclick on a library entry it opens a FD: window that shows all functions of the library when the library is defined in the Barfly.FD file. Furthermore if you doubleclick on a function you have the choice to see the function in a DissWindow or the autodocs documentation.

* Open Device Window

opens a window to show the devices.

* Open Resource Window

opens a window to show the resources.

* Open Port Window

opens a window to show the public ports.

* Open Resident Window

opens a window to show the resident modules.

* Open Interrupt Window

opens a window to show the interrupts.

* Open AutoDocs Window

opens a filerequester to choose the needed autodocs information of a library. Now a window is opened that shows all function of the chosen autodoc file. If you now click on a function another window is opened that shows the function documentation.

* Open History Window

opens a HistoryWindow that shows the last saved registerframes of the undobuffer. The undobuffer is organized as a stack that the first entry is the last entry in the HistoryWindow. The HistoryWindow isn't updated automatic.

* Stack Check

controlls the stack check. If the register A7 points out of the stack bounds or points on an odd address a warning is shown. The Debugger only checks the task when the task give back the control to the traproutine, so it's not possible to notice every stack problem.

You should be aware that this function doesn't work with a WShell task because the WShell doesn't set the correct stack task values.

* Find Task of Adress

trys to find the task that belongs to the entered address. The command checks if the address is in the task,process,cli, mementry structure and the hunks. It's not safe to assume that the function can check all cases.

* Load Binary

loads a file with an optional length into a memory area. If the Debugger should allocate the memory block automaticly you have to close the memory requester.

* Save Binary

saves a memory area into a file.

* Freeze Task

freezes a selectable task. When bdebug ends the frozen tasks are warmed up again.

* Warm up Task

warms up a frozen task.

* Kill Task

kills a selectable task.

You should know what task you can kill.

* Show Task

shows the task structure of a selectable task.

* Show Prozess

shows the process structure of a selectable process.

* Show CLI

shows the cli structure of a selectable process.

* Show Hunk

shows the hunks of a selectable process.

* Send Task Signal

sends a signal to a selectable task.

* Set Task Priority

sets a priority of a selectable task.

* Refresh Code Cache

refreshes the Code Cache.

* Refresh Data Cache

refreshes the Data Cache

1.19 Barfly.guide/UB_FPUW

Usage of BDebug

The FPU Window

=====

The FPU Window
Local Menu

The FPU Window
Local Menu

1.20 Barfly.guide/FPUW_Window

FPU Window

The FPU Window shows the FPU register FP0 to FP7 in the 96Bit Extended format and the registers FPCR, FPSR and FPIAR in hexadecimal. You can only open this window if a FPU is available.

Register Window Layout

```
FP0=FloatingPoint
FP1=FloatingPoint
FP2=FloatingPoint
FP3=FloatingPoint
FP4=FloatingPoint
FP5=FloatingPoint
FP6=FloatingPoint
FP7=FloatingPoint

FPCR=xxxxxxxx FPSR=xxxxxxxx

FPIAR=xxxxxxxx
```

1.21 Barfly.guide/FPUW_LocalMenus

Local Menu

* Close Window

closes the window.

1.22 Barfly.guide/UB_DISSW

Usage of BDebug

The Disassembler Window

=====

The Disassembler Window
Local Menus

The Disassembler Window
Local Menus

1.23 Barfly.guide/DISSW_Window

Disassembler window

The DissWindow shows the memory contents in assembler mnemonics. The address of the window's view can be absolut or relative. In the absolut mode the window is set to a fixed adress and you can read in the title No Link. In the relative mode the window is connected with a register so the window's view depends on the registers value. You can see this mode by the title string Link to * where * represents the register name. The PC is shown by the colour pen 2. If the linked register value is outside of the window's view area the whole contents of the window will be refreshed. You can change size of the window and scroll through the memory area by using the cursors. In the title you see an ID-String with the format \#x.y where X represents the REGWindow number and y the number of the MemWindow. By a doubleclick in a line of the window you can sets or remove a breakpoint. You can disable this function in the configuration.

1.24 Barfly.guide/DISSW_LocalMenus

Local Menus

* Close Window

closes the window.

* Shrink Window

shrinks the window.

* Expand Window

expands the window to screen size.

* Link to Register

links the window with a register. If you enter the string NO it switches to the absolute mode.

* Change Adress

changes the view address of the window.

* Clear Adress

resets the view address of the window.

* Refresh Window

refreshes the window.

* .W Branches

activates and deactivates the output of the old branch width size.

* Neg. Offsets

activates and deactivates the output of negative values in the indirect address modes with offset.

* Neg. Data

activates and deactivates the output of negative values in the direkt address mode.

* Opcode Data

activates and deactivates the additional output of the command bytes.

* Auto Refresh

activates and deactivates the global refresh of the window after each step.

* Symbols

activates and deactivates the symbol output in the window.

* Show Lib Call

activates and deactivates the symbolic output of library functions so all library functions that are defined in the configuration file <BarflyPath>/Barfly/BARFLY.FD are recognized.

* Guess Lib Call

activates and deactivates the guessing of function call names. It only works in connection with the option Show Lib Call. Fortunately you can't expect that the function names always fit because the the library base register A6 can change until the program counter meets the function.

* Mark Block End

activates and disactivates marking after the instruction
JMP,BRA,RTS,RTE,RTD and RTR to make program
blocks more visible.

* Set/Clear Breakpoint

sets/removes a breakpoint on the first entry in the window.
Breakpoints are shown by changing the pen from colour 1 to colour
3 and the char > at the beginning of a line.

* Pick/Clear Breakpoint

sets/removes a breakpoint through a symbol list.

* Disassemble to File

disassembles a memory area into a file.

1.25 Barfly.guide/UB_MEMW

Usage of BDebug

The Memory Window

=====

The Memory Window
Local Menus

The Memory Window
Local Menus

1.26 Barfly.guide/MEMW_Window

Memory window

The MemWindow shows the memory contents hexadecimal and in ascii.You
can change size of the window and scroll through the memory area by
using the cursors. In the title you see an ID-String with the format
\#x.y where X represents the REGWindow number and y the number of the
MemWindow.

1.27 Barfly.guide/MEMW_LocalMenus

Local Menus

* Close Window

closes the window.

* Shrink Window

shrinks the window.

* Expand Window

expands the window to screen size.

* Link to Register

links the window with a register. If you enter the string NO it switches to the absolute mode.

* Change Adress

changes the view address of the window.

* Clear Adress

resets the view address of the window.

* Refresh Window

refreshes the window.

* Memory Offset Step

defines the data format in the window. The following options can be selected: None, Byte, Word and Long.

* Edit

activates the edit mode of the MemWindow. In the edit mode you can switch between hex and ascii input by the key RETURN. With ESC you can leave the edit mode. Only the cursor right and left are changed to the normal. With these both keys you can access each Byte. In the edit mode you can't change the size of the window.

* Copy

copies a memory area into another memory area. The function uses CopyMem so it doesn't handle memory areas that overlap.

* Fill

fills a memory area with a value of a certain data-width.

* Compare

compares a memory area with another memory area.

* Search

searches a value of a certain data-width in a memory area. If the value is found the address and the value are shown and you can go on with Search Next to find the next address.

* Search Next

Searches the next value. Look at Search

* Pred

sets the address of the window on the preceding entry of the list. If the node points on an odd, illegal or address NULL the command has no effect. The next node is equal to LN_PRED, the second longword of the memory view.

* Succ

sets the address of the window on the next entry of the list. If the node points on an odd, illegal or address NULL the command has no effect. The next node is equal to LN_SUCC, the first longword of the memory view.

1.28 Barfly.guide/UB_COPPW

Usage of BDebug

The Copper Window

=====

The Copper Window
Local Menus

The Copper Window
Local Menus

1.29 Barfly.guide/COPPW_Window

CopperWindow

The CopperWindow shows the memory contents as copper commands. You can change size of the window and scroll through the memory area by using the cursors. In the title you see an ID-String with the format \#x.y where X represents the REGWindow number and y the number of the CoppWindow.

1.30 Barfly.guide/COPPW_LocalMenus

Local Menu

* Close Window

closes the window.

* Shrink Window

shrinks the window.

* Expand Window

expands the window to screen size.

* Link to Register

links the window with a register. If you enter the string NO it switches to the absolute mode.

* Change Adress

changes the view address of the window.

* Clear Adress

resets the view address of the window.

* Refresh Window

refreshes the window.

* Goto Into List

sets the window list on the standard copperlist
GfxBase->gb_copinit.

1.31 Barfly.guide/UB_STRUCTW

Usage of BDebug

The Structure Window

=====

The Structure Window
The Structure Format
Local Menu

The Structure Window
The Structure Format
Local Menu

1.32 Barfly.guide/STRUCTW_Window

StructWindow

opens window that can be connected with a structure. You can use new structure entries by expanding the the <BarflyPath>/Barfly/Barfly.Include file or loading a new custom file. By a doubleclick on a structure window entry you can cause several actions depending on the datatype. Every datatype is connected with an action that is normally started automatically. With the configuration command NoAutoStructAction you can change this behaviour so that an action type requester is opened.

The following datatypes are available.

- * APTR opens a MemWindow.
- * CSTR shows a string.
- * BPTR opens a MemWindow at the address BPTR*4.
- * BSTR shows a string at the address (BPTR*4)+1
- * CPTR opens a DissWindow.
- * FPTR opens a DissWindow.
- * BYTE doesn't cause an action.
- * WORD doesn't cause an action.
- * LONG doesn't cause an action.
- * FLOAT doesn't cause an action.
- * DOUBLE doesn't cause an action.
- * EXTENDED doesn't cause an action.
- * RPTR doesn't cause an action.

The following action types are available.

- * MemWindow opens a MemWindow.
 - * DissWindow opens a DissWindow.
 - * CoppWindow opens a CoppWindow.
 - * StructWindow opens a StructWindow.
 - * NewStruct sets a new structure.
-

1.33 Barfly.guide/STRUCTW_Format

Structure Macro Fileformat

In the beginning you define the root directory entries with the Macro Menudir. The first parameter is the name of the entry, then the address of the parent directory and then the address of the subdirectory. In the root directory the parent address is fortunately NULL. The last entry of the directory is defined by the Macro MENUDIREND.

Label	ListViewMacro	Link
RootDir:	.	
	.	
	MENUDIR	exec,0,Exec_Dir
	.	
	.	
MYCUSTOMETRY:	MENUDIREND	CUSTOM,0,0

The design of a subdirectory only differs from the root directory entries by a parent directory address.

Label	ListViewMacro	Link
Exec_Dir:	.	
	.	
	MENUDIR	nodes.i,RootDir,Nodes_Dir
	.	
	.	
	MENUDIREND	tasks.i,RootDir,Tasks_Dir

to define the structure directory entries you have to use MENUITEM and MENUITEMEND. The first parameter in the Item Macros is the name of the entry and also the name of the structure. The second parameter defines the address of the parent directory.

Label	ListViewMacro	Link
	.	
	.	
Nodes_Dir:	MENUITEM	LN,Exec_Dir
	.	
	.	
	MENUITEMEND	

To define a structure you can use the normal assembler syntax that you probably have to adjust to your custom needs. For example you can tell BDebug more informations about the datatype an entry represents. By redefining APTR to a CSTR you can tell Debugger that the entry is a stringpointer. Or you can tell that APTR points to a structure by APTR LN_SUCC,Node.

Label	IncludeTypeMacro	Name,Link
-------	------------------	-----------

LN_Struct:	STRUCTUREB	LN,0
	APTR	LN_SUCC, LN
	APTR	LN_PRED, LN
	UBYTE	LN_TYPE
	BYTE	LN_PRI
	CCSTR	LN_NAME
	LABEL	LN_SIZE

1.34 Barfly.guide/STRUCTW_LocalMenus

Local Menus

* Close Window

closes the window.

* Shrink Window

shrinks the window.

* Expand Window

expands the window to screen size.

* Link to Register

links the window with a register. If you enter the string NO it switches to the absolute mode.

* Change Adress

changes the view address of the window.

* Clear Adress

resets the view address of the window.

* Refresh Window

refreshes the window.

* Load Custom Struct

loads additional structure files. The new structure entries are place in the CUSTOM directory. The format of custom structure files is equal to the file BARFLY.INCLUDE.

* Select Structure

opens the structure include directory requester where you can select the needed structure. The parent gadget is placed in the upper border.

* Goto Sysbase...

sets the window adress on the ExecBase.

* Goto Gfxbase...

sets the window address on the GFXBase.

* Save Window....

saves the contents of the window in a file.

* Full Address

this switch decides if the StructWindow also shows the address of the entries.

* Offset Address

this switch decides if the StructWindow also shows the offset of the entries.

* Pred

sets the address of the window on the preceding entry of the list. If the node points on an odd, illegal or address NULL the command has no effect. The next node is equal to LN_PRED, the second longword of the memory view.

* Succ

sets the address of the window on the next entry of the list. If the node points on an odd, illegal or address NULL the command has no effect. The next node is equal to LN_SUCC, the first longword of the memory view.

1.35 Barfly.guide/UB_SOURCEW

Usage of BDebug

The Structure Window

=====

The Source Window
Local Menus

The Source Window
Local Menus

1.36 Barfly.guide/SOURCEW_Window

Source window

The SourceWindow shows the source line that belongs to the window address. If the program file doesn't have the needed debug informations the Source window can't be opened. If the address points to an area with no relevant debug information, for example the Kickstart or beyond the program hunks, you only see a small message.

1.37 Barfly.guide/SOURCEW_LocalMenus

Local Menus

- * Close Window

closes the window.

- * Shrink Window

shrinks the window.

- * Expand Window

expands the window to screen size.

- * Link to Register

links the window with a register. If you enter the string NO it switches to the absolute mode.

- * Change Adress

changes the view address of the window.

- * Clear Adress

resets the view address of the window.

- * Refresh Window

refreshes the window.

- * Set Breakpoint

sets a breakpoint on the active line.

- * Show HunkInfo

shows the hunk of the current source line.

1.38 Barfly.guide/UB_SNOOPW

Usage of BDebug

The Snoop Window

=====

The Snoop Window
Local Menus

The Snoop Window
Local Menus

1.39 Barfly.guide/SNOOPW_Window

Snoop Window

The SnoopWindow snoops the task's allocations.

1.40 Barfly.guide/SNOOPW_LocalMenus

Local Menus

* Close Window

closes the window.

* Shrink Window

shrinks the window.

* Expand Window

expands the window to screen size.

* Refresh Window

refreshes the window.

* Auto Refresh

activates/disactivates display refresh by an allocation.

* Snoop Memory

activates/disactivates snooping.

* Snoop Mask

sets the allocation filter mask. Is the Mask 20 only allocations with the size 20 are recorded. Default -1.

* Snoop Max Entries

sets the maximal recorded snoop entries.

1.41 Barfly.guide/UB_BREAKW

Usage of BDebug

The Breakpoint Window

=====

The Breakpoint Window
Local Menu

The Breakpoint Window
Local Menu

1.42 Barfly.guide/BREAKW_Window

Breakpoint window

The BreakWindow handles all breakpoints and contains the functions that are needed with breakpoints. In general breakpoints are addresses in the program where the task should be stopped. The breakpoints are handled global so they aren't deleted when close the window.

1.43 Barfly.guide/BREAKW_LocalMenus

Local Menu

* Toggle

activates and disactivates all breakpoints.

* All

selects all breakpoints.

* Clear

unselects all breakpoints.

* On

activates all selected breakpoints.

* Off

disactivates all selected breakpoints.

* Hit

sets the amount of hits for a breakpoint until it should stop the program. Default is 1.

* ?

shows the hunk where the breakpoint is located and if the breakpoint is equal to a symbol.

* Input

this breakpoint sets and removes a breakpoint.

* Pick

this breakpoint sets and removes a breakpoint using the symbol list.

* Delete

removes every selected breakpoint.

* Goto

opens a DissWindow for every selected breakpoint.

* Run

runs the program until the PC hits a selected breakpoint.

1.44 Barfly.guide/UB_WATCHW

Usage of BDebug

The Watchpoint Window

=====

The Watchpoint Window
Local Menus

The Watchpoint Window
Local Menus

1.45 Barfly.guide/WATCHW_Window

Watchpoint window

The Watchwindow allows to set breakpoints that aren't dependend on a certain PC address but from other conditions. Every watchpoint has a condition, data width and state if it's activated or not. There are 3 types of different watchpoints available now. The Memory watchpoint compares the saved contents of the address with the current contents and dependent on the condition the program is stopped or not. The Register watchpoint compares the saved contents of a register with the current contents and dependent on the condition the program is stopped or not. The Argument watchpoint compares the saved value of an argument with the current contents and dependent on the condition the program is stopped or not. The last watchpoint type is the most powerful because it can simulate the first two types with the cost of a slowdown. The use of watchpoints is very time consuming because the whole program is run in single stepping To use watchpoints you have to run the task with Run Watched Task.

If an error happens during a dynamic argument in the exception handler the screen is blinked.

1.46 Barfly.guide/WATCHW_LocalMenus

Local Menus

- * Toggle

activates and disactivates all watchpoints.

- * All

selects all watchpoints.

- * Clear

unselects all watchpoints.

- * On

activates all selected watchpoints.

- * Off

disactivates all selected watchpoints.

- * Add

opens a requester where the parameter for a watchpoint have to be

adjusted and adds the new watchpoint to the list. If you doubleclick on a watchpoint you can change the parameter.

* Refresh

refreshes the watchpoint arguments.

* Check

checks all selected watchpoints.

* Delete

removes all selected watchpoints.

1.47 Barfly.guide/UB_CHECKW

Usage of BDebug

The Checksum Window

=====

The Checksum Window
Local Menus

The Checksum Window
Local Menus

1.48 Barfly.guide/CHECKW_Window

Checksum Window

The ChecksumWindow controllls all checksum areas that are been checked each time the task stops. Helpful to find illegal random writes bugs. The checkpoints are controlled global so they aren't deleted when you close the window.

1.49 Barfly.guide/CHECKW_LocalMenus

Local Menus

* Toggle

activates or disactivates all checksum areas.

- * All
selects all checksum areas.
- * Clear
unselects all checksum areas.
- * On
activates all selected checksum areas.
- * Off
disactivates all selected checksum areas.
- * Address
adds a checksum area into the list.
- * Hunk
adds a hunk of the current process into the checksum area list.
- * Task
adds a hunk of selectable process into the checksum area list.
- * Refresh
calculate a new checksum for all selected areas.
- * Delete
removes all selected checksum areas.
- * Check
checks all areas for checksum errors.

1.50 Barfly.guide/UB_ARGUMENTS

Usage of BDebug

Requester Arguments
=====

Argument Structur

An argument can use absolut values,symbols and registers as operands and the operators +,-,*,/,|,!,&,<<,>>,~. Additionally to the normal symbols there are some special symbols available.

- * By {Argument}.[b,w,l] you can read from a memory address, that is defined by the argument. If you specify an illegal address that isn't defined in the legal memory space an error is shown.
- * \#d? represents the address of the Disswindows with the ID ?
- * \#m? represents the address of the Memwindows with the ID ?
- * \#c? represents the address of the Coppwindows with the ID ?
- * \#h? represents the address of the hunk ?. Helpful for enforcer Hunk:Offset output
- * \#ea? represents the address of the EA with the number ?. Check Register Window.
- * \#em? represents the contents where the address EA number ? points to. Check Register Window. If the address EA is illegal an error is shown.
- * \#ls represents the start address of a loaded binary file.
- * \#le represents the end address of a loaded binary file.
- * \#ll represents the length address of a loaded binary file.
- * \#p represents the start address of the programs. Only true for a loaded program.

If you have the following Enforcer Hit output Hunk 0:\$11c you can calculate the address by entering the argument #h0+\$11c.

1.51 Barfly.guide/UB_TECHINFOS

Usage of BDebug

Technical Informations
=====

Exceptions

The Debugger can catch all exceptions if the system is still working. If an exception is caused the traphandler catches the exception and tells the Debugger what went on so it can react on the exception. If the exception wasn't caused by the Debugger the type and the possible reason for the exception is shown. The Return-Adress of the debugged task points on an internal ILLEGAL. If the PC points on this ILLEGAL the task is closed and all windows are removed. You should step over this ILLEGAL because it increases possibility of a system crash. If a task is caught by Debug Next Task and notices a finalPC pointer the Return-Adress isn't set on an internal ILLEGAL because the finalPC

pointer is sometimes used for parsing an argument. In this case the Debugger notices that the task ends by the `RemTask()` function.

If the task changes the Return-Adress the Debugger tries to determine the taskend by `RemTask`.

Exception Handler

Every task contains in the task structure a pointer to its exception handler that is named `TC_TRAPCODE`. When an exception happens in this task the exception checks if the Debugger knows this task. If this is not the case something seriously is broken and the Deadend Alert 35000000 will be popped up. If all goes well the registers are saved, the Debugger task gets a message and the exception handler waits for a msg by the Debugger to go on. When the Debugger gets the message it causes the appropriate function. For example refreshing the windows. If the Debugger gets a step command it sends the exception handler the appropriate message and the handler does a step.

Debug Informations

Currently the following formats are supported.

- * `BASM Specialformat` This format allows the Debugger to decide if the code is in the Mainpart, Includes or in a Macro.
- * `SAS D1` This format only allows a Source-Code connection. It doesn't support local variables, Structures and Macros.
- * `GCC STABS` This format is very powerful and offers all a source-level Debugger needs. Unfortunately the Debugger only supports a simple Source-Code connection at the moment. It's planned to support more in the future.

GCC Compiler and BDebug

Unfortunately you can't debug programs that are using the current `ixemul.library` because in `Openlibrary()` `initroutine` the `Task` field `TC_TRAPCODE` is changed. Hopefully there'll be soon an `ixemul.library` available that doesn't change the `traphandler`. If you're using GCC with the link `lib gerlib` that is available on Aminet FTP Servers you shouldn't experience any problems with BDebug.

1.52 Barfly.guide/UB_CONFIG

Usage of BDebug

Configuration

=====

The default configuration file is named BDEBUG.Config and is located in the directory <BarflyPath>/ or s:Barfly/. Obviously it's not optimal to be forced to use the same config file for different programs. Therefore you can also specify a local config file with program name and the suffix .BDebug.

Tooltypes	Available Tooltypes
Barfly.FD	The Barfly.FD format
Commands	Configuration Commands

1.53 Barfly.guide/CO_TOOLTYPES

ToolTypes

The following tooltypes are supported to activate the know functions of the commandwindow.

- * CatchNextTask
- * CatchCrashedTask
- * CatchEnforcerHit

1.54 Barfly.guide/CO_BARFLYFD

Barfly.FD

If you want to create a new Barfly.FD file that contains the library function name you have to follow the following the instructions. First the assign FD: has to point to the directory that contains the FD files that should be contained in the new Barfly.FD. Afterwards you should check every FD file if the Library, Resource, Device name exist in the first line in the following style: * "foobar.library". If this is not the case you have to add the name yourself so that a correct FD database can be build up. If you're more experienced with FD files you can yourself add new entries to the Barfly.FD file because the layout is pretty obvious.

1.55 Barfly.guide/CO_CONFIGCMDS

Configuration Commands

Window Config Commands...

Register Window	Register Window Object
FPU Window	FPU Window Object
Disassembler Window	Disassembler Window Object
Memory Window	Memory Window Object
Copper Window	Copper Window Object
Struct Window	Struct Window Object
Source Window	Source Window Object
Breakpoint Window	Breakpoint Manager Window Object
Watchpoint Window	Watchpoint Manager Window Object
Checksum Window	Checksum Manager Window Object
Snoop Window	Snoop Memory Window Object
Info Window	Information Windows
Other Windows	Other Windows

Misc Commands...

Misc Commands	Miscellaneous Commands
---------------	------------------------

1.56 Barfly.guide/CC_REGW

Register window

RegWindow=x/y/width/height/register

This command defines the position of a REGWindow.

RegFlags=flag[|flags...]

This command defines certain flags in the REGWindows.

- * AUTOVIEWREFRESH
- * SYMBOLS
- * STACKCHECK
- * NOBIGVIEW

1.57 Barfly.guide/CC_FPUW

FPU Window

FpuWindow=x/y/width/height/register

.....

This command defines the position of a FPUWindow.

OpenFPUWindows=Count

.....

This command tells the Debugger to open a FPUWindow.

1.58 Barfly.guide/CC_DISSW

Disassembler window

DissWindow=x/y/width/height/register

.....

This command defines the position, the dimension and linked register of the DissWindow.

Beispiel: DISSWINDOW=0/0/300/100/PC

DissFlags=flag[|flags...]

.....

This command defines certain flags in the DISSWindow.

* AUTOREFRESH

* SHOWLIB

* GUESSLIB

* SHOWEA

* WORDBRANCHES

* NEG OFFSETS

* NEGDI

* OPCODEDATA

* BLANKAFTERJMPBRA

OpenDissWindows=Count

.....

This command tells the Debugger to open a number of DissWindows.

1.59 Barfly.guide/CC_MEMW

Memory window

MemWindow=x/y/width/height/register
.....

This command defines the position, the dimension and linked register of the MemWindow.

Example: MEMWINDOW=0/0/300/100/A0

OpenMemWindows=Count
.....

This command tells the Debugger to open a number of MemWindows.

MemoryOffsetStep=Count
.....

This command defines the Offset-Step of the MemWindows.

- * 0 no Space
- * 1 Space after each Byte.
- * 2 Space after each Word.
- * 4 Space after each Longword.

1.60 Barfly.guide/CC_COPPW

Copper window

CoppWindow=x/y/width/height/register
.....

This command defines the position, the dimension and linked register of the CoppWindow.

Example: COPPWINDOW=0/0/300/100/A0

OpenCoppWindows=Count
.....

This command tells the Debugger to open a number of CoppWindows.

1.61 Barfly.guide/CC_STRUCTW

StructWindow

StructWindow=x/y/width/height/register
.....

This command defines the position, the dimension and linked register of the StructWindow.

Example: StructWINDOW=0/0/300/100/A0

StructFlags=flag[|flags...]
.....

This command defines certain flags for the StructWindow

- * AUTOREFRESH
- * ADDRESSFORMAT
- * OFFSETFORMAT
- * NEWWINDOW

OpenStructWindows=Count
.....

This command tells the Debugger to open a number of StructWindows.

1.62 Barfly.guide/CC_SOURCEW

Source window

SourceWindow=x/y/width/height/register
.....

This command defines the position, the dimension and linked register of the SourceWindow.

Example: SOURCEWINDOW=0/0/300/100/A0

OpenSourceWindows=Count
.....

This command tells the Debugger to open a number of SourceWindows.

1.63 Barfly.guide/CC_BREAKW

Breakpoint window

BreakWindow=x/y/width/height

.....

This command defines the position of a BreakPointWindow.

OpenBreakWindows=Count

.....

This command tells the Debugger to open a BreakPointWindow.

1.64 Barfly.guide/CC_WATCHW

Watchpoint window

WatchWindow=x/y/width/height

.....

This command defines the position of a WatchpoinzWindow.

OpenWatchWindows=Count

.....

This command tells the Debugger to open a WatchPointWindow.

1.65 Barfly.guide/CC_CHECKW

Checksum window

ChecksumWindow=x/y/width/height

.....

This command defines the position of a ChecksumWindow.

ChecksumWindows=Count

.....

This command tells the Debugger to open a ChecksumWindow.

1.66 Barfly.guide/CC_SNOOPW

SnoopMemory window

SnoopMemWindow=x/y/width/height/register

.....

This command defines the position of a SnoopMemWindow.

OpenSnoopMemWindow=Count

.....

This command tells the Debugger to open a SnoopMemWindow.

SnoopMask=Mask

.....

This command defines the snoop mask. The mask defines the length of memory blocks that should be recorded. Default is -1 so everything is recorded.

SnoopMax=Count

.....

This command defines the count of snoop entries. Default is 100.

HistoryEntrys=Count

.....

This command defines the count of history entries. Default is 5.

1.67 Barfly.guide/CC_INFOW

Information Windows

GlobalViewWindow=x/y/width/height

.....

This command defines the position and dimensions of a standard information window. For example the Library Window belongs to this group.

Example: GLOBALVIEWWINDOW=0/0/300/100

GLOBALVIEWWINDOW=0/200/300/100

1.68 Barfly.guide/CC_OTHERW

Other Windows

CommandWindow=x/y/width/height

.....

This command defines the position of the small CommandWindow. This command has no function in local configuration files.

FileShell=<Window Spezifikation>

.....

This command defines the shell that is opened with the loaded program. You should always open the shell on the Debugger's Public Screen. The shell parameters are the same you know from the CLI.

1.69 Barfly.guide/CM_MISC

Misc

Task Stack	Define program's stacksize
Task Priority	Sets the Debugger Priority
SetBreak	Set a breakpoint
ClickBreak	Define DissWindow Action on a Click
Showmem	Define readable Memory areas
DefCommand	Define custom commands
AutoDocDir	Set the autodocdir
AutoDocAlias	Define the library/autodoc alias
Arexxpath	Define the arexxpath
Arexxinput	Define the arexx input handle
Arexxoutput	Define the arexx output handle
Arexxcommand	Define an arexxcommand
Execute Cmd	Define the programs that should be started
LoadInclude	Load struct database file
AddStructFile	Load a custom struct database file
ClickToFront	Activate click to front
CenterWindow	Center requester windows ?
Screeninfront	Screen to front ?
OpenScreen	Define an own screen
OpenPubScreen	Define a Pubscreen
ScreenFont	Define an own screenfont
QuietException	Set certain exceptions quiet
DisableXWindow	Disable Waitpointer
TraceBreak	Define the trace method
CrashedTask	Activate catching crashed tasks
Catch Hit	Activate catching Enforcer hits
Cache File	Activate file caching
Pop Path Request	Disable path requester
Auto Action	Activates the StructWindow action requester
NoBreakErr	Ignores SetBreak config errors

1.70 Barfly.guide/CM_TASKSTACK

TaskStack=Count
.....

This command defines the stack of the loaded program. Default are 4096 Bytes.

1.71 Barfly.guide/CM_TASKPRI

Priority=Count
.....

This command defines the Debugger's priority.

1.72 Barfly.guide/CM_SETBREAK

SetBreak=Argument
.....

This command can be used to define a list of breakpoints that are set before the program is started. This is useful to pass the module Main.c for example. If no breakpoints are defined or if a parsing problem occurs the standard breakpoint, first program instruction, is set.

- * SETBREAK=_main ; SAS C Main Program Start
- * SETBREAK=_main ; GCC C Main Program Start
- * SETBREAK=@main ; DICE C Main Program Start
- * SETBREAK=! ; Programstart (Default)

1.73 Barfly.guide/CM_CLICKBREAK

ClickBreak=State
.....

This command can be used to define the action of the DissWindow on a doubleclick.

State=0
No Action (Default).

State=1

Set/Clear Breakpoint and pop up a Requester for a Set.

State=2

Set/Clear Breakpoint.

1.74 Barfly.guide/CM_SHOWMEM

ShowMem=Start:End

.....

defines the adress areas that are legal to the Debugger so you can look at adress areas that are not in the memorylist or in the rom. Illegal adress areas are shown with * in the windows. You should never define the custom chip areas as legal because a read access on a writeonly register can cause a deadly crash.

Example: SHOWMEM=\$e80000:\$f00000 defines the Zorro 2 area as free.

By this command you can overrule the internal enforcer legal memory areas so you should beware of hits.

1.75 Barfly.guide/CM_DEFCOMMAND

DefCommand=Key,Qualifier[|Qualifier...],Function

.....

This commands allows to connect menu functions with key sequences. Because of the object-oriented concept of the Debugger that allows multiple instances of objects it's not easy to decide what object is meant. Therefore if the object is active it's used and if no object of this type is active the first entry the object-type list is used. As the key parameter every Rawkey can be used with the exception of TAB and the functionkeys that are used internal. The key is searched first in the local and then in the global configuration.

As qualifiers you can use the following keys.

* LSHIFT

* RSHIFT

* CAPSLOCK

* CTRL

* LALT

* RALT

* LCOMMAND

* RCOMMAND

Bespiel: DEFCOMMAND=\$15,CTRL,"Step 1 Position"

Defines CTRL-Z as Step 1 Position

1.76 Barfly.guide/CM_AUTODOCDIR

AutoDocDir=<Path>

.....

This command sets the path for the autodocs directory.

1.77 Barfly.guide/CM_AUTODOCALIAS

AutoDocAlias=Library,File

.....

This command sets an alias for Libraries,Devices or Resources to define the connected Autodocs file. There's no other way because it's not possible to build the autodocs file by knowing the library name.

1.78 Barfly.guide/CM_AREXXPATH

ArexxPath=<rx-path>

.....

This command sets the Arexx-Script Start-Command. In a normal system the path should be <sys:Rexxc/rx>.

1.79 Barfly.guide/CM_AREXXINPUT

ArexxInput=<File>

.....

This command sets the Arexx-Command Input-File. If you don't specify the file, NIL: is used.

1.80 Barfly.guide/CM_AREXXOUTPUT

```
ArexxOutput=<File>  
.....
```

This command sets the Arexx-Command Output-File. If you don't specify the file, NIL: is used.

1.81 Barfly.guide/CM_AREXXCMD

```
ArexxCommand=[1...10], <Pfad>  
.....
```

This command sets the 10 entries in the Arexx-Menu. You specify the number and the path of the Arexx-Script.

```
ArexxCommand=1, "Rexx:Example.rexx"
```

1.82 Barfly.guide/CM_EXECMD

```
ExecuteCommand=<File>  
.....
```

this command can set up a list of programs that should be run before the debugged program's task is started. This parameter only works with programs that are loaded by the Debugger. Furthermore you have to make sure that the loaded programs have to end otherwise the task can't be started. For example you could use these command to set breakpoints with Arexx-Scripts.

1.83 Barfly.guide/CM_LOADINCLUDE

```
LoadInclude  
.....
```

tells the Debugger to load the structure information file Barfly.Include.

1.84 Barfly.guide/CM_ADDSTRUCT

```
AddStructFile=Filename  
.....
```

tells the Debugger to load a custom structure information file and adds it into the CUSTOM/ subtree.

1.85 Barfly.guide/CM_CLICK2FRONT

```
ClickToFront  
.....
```

activates the Debugger's own ClickToFront handler. This function should only be used if you don't use an own Commodity for this task.

1.86 Barfly.guide/CM_CENTERW

```
CenterWindow  
.....
```

activates the centering mode for every stringrequester windows.

1.87 Barfly.guide/CM_SCREENFRONT

```
ScreenInFront  
.....
```

activates the ScreenToFront mode that pops the screen to front after every trace operation.

1.88 Barfly.guide/CM_OPENSREEN

```
OpenScreen[=width,height,depth,mode]  
.....
```

tells the Debugger to open an own screen. If you don't enter dimension parameters the wb screen is cloned. You can use the screenmodes in the mode string that you can see in the Prefs/ScreenMode requester. This command has no function in local configuration files.

```
OPENSREEN=1448,560,2,PAL:HighRes Interlace
```

1.89 Barfly.guide/CM_OPENPSCREEN

OpenPubScreen=Name
.....

tells the Debugger to open on the Pubscreen with the specified name.
This command has no function in local configuration files.

1.90 Barfly.guide/CM_SCREENFONT

ScreenFont=fontname/Height
.....

defines a font for a Debugger screen. This command has no function
in local configuration files.

1.91 Barfly.guide/CM_QUIETEX

QuietException=Exception Nummer
.....

masks off certain exceptions for the exception requester so that
only a DisplayBeep is caused instead of a textrequest. With the value
-1 you can mask off every exception and for example with the value 4
you mask off the Illegal exception.

1.92 Barfly.guide/CM_DISXP

DisableXPointer
.....

disactivate the Wait-Pointer.

1.93 Barfly.guide/CM_TRACEBREAK

TraceBreak
.....

tells the Debugger to use breakpoints in the Subroutine Traces
instead of single steps. The advantage is a speed up and the
disadvantage is that you can cause crashes while you step through
resident/reentry code.

1.94 Barfly.guide/CM_CRASHEDTASK

CrashedTask
.....

activates the CatchCrashedTask mode.

1.95 Barfly.guide/CM_CATCHHIT

CatchEnforcerHit
.....

activates the CatchEnforcerHit mode.

1.96 Barfly.guide/CM_CACHEFILE

DoNotCacheFullFile
.....

tells the Debugger not to cache program files while reading the Symbol/Debug informationen to save memory. Obviously the parsing speed will decrease.

1.97 Barfly.guide/CM_POPPATH

DoNotPopPathRequest
.....

tells the Debugger to ignore errors from opening source files and not to open a path requester.

1.98 Barfly.guide/CM_AUTOACT

NoAutoStructAction
.....

tells the Debugger to open a type-requester by an action in the StructurWindow.

1.99 Barfly.guide/CM_NOBREAKTERRORS

NoBreakpointErrors
.....

tells the Debugger to ignore SETBREAK= errors that cause the Debugger to always set an error on the program start.

1.100 Barfly.guide/UB_AREXX

Usage of BDebug

Arexx
=====

Commands

SIMPLEREQUEST "
.....

* RC: -

* result: 'OK'

TWOGADREQUEST "
.....

* RC: -

* result: 'OK','FALSE'

TRIGADREQUEST "
.....

* RC: -

* result: 'OK','FALSE','RESUME'

NEXT_ROOTWINDOW
.....

* RC: -

* result: 'OK','FALSE'

NEXT_SUBWINDOW
.....

* RC: -

* result: 'OK','FALSE'

FIRST_DISSWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_MEMWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_COPPPWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_FPUWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_BREAKPOINTWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_STRUCTWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_SOURCEWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_SNOOPWINDOW

.....

* RC: -

* result: 'OK', 'FALSE'

FIRST_WATCHWINDOW

.....

* RC: -

* result: 'OK','FALSE'

ACTIVATE_ROOTWINDOW

.....

* RC: -

* result: 'OK','FALSE'

ACTIVATE_SUBWINDOW

.....

* RC: -

* result: 'OK','FALSE'

OPEN_DISSWINDOW '@REG' | 'Argument'

.....

* RC: -

* result: -

OPEN_MEMWINDOW '@REG' | 'Argument'

.....

* RC: -

* result: -

OPEN_COPPWINDOW '@REG' | 'Argument'

.....

* RC: -

* result: -

OPEN_SOURCEWINDOW '@REG' | 'Argument'

.....

* RC: -

* result: -

OPEN_STRUCTWINDOW '@REG' | 'Argument'

.....

* RC: -

* result: -

OPEN_BREAKPOINTWINDOW

.....

```
.....

    * RC: -

    * result: -

OPEN_FPUWINDOW
.....

    * RC: -

    * result: -

OPEN_SNOOPMEMORYWINDOW
.....

    * RC: -

    * result: -

DOMENU 'Menu-String'
.....

    * RC: -

    * result: -

SET_BREAKPOINT 'Argument'
.....

    * RC: -

    * result: 'OK','FALSE'

CLEAR_ICACHE 'Address,Length'
.....

    * RC: -

    * result: 'OK'

CLEAR_ICACHE 'Address,Length'
.....

    * RC: -

    * result: 'OK'

GOTO_ADDRESS 'Address'
.....

    * RC: -

    * result: 'OK','FALSE'

CLEAR_ADDRESS
.....
```

```
* RC: -

* result: 'OK','FALSE'

LINK_REGISTER 'Register'
.....

* RC: -

* result: 'OK','FALSE'

SET_REGISTER 'Register,Value'
.....

* RC: -

* result: 'OK','FALSE'

Read_Byte 'Address'
.....

RC      : 0=Ok      Result: Result-String

Read_Word 'Address'
.....

RC      : 0=Ok      Result: Result-String

Read_Long 'Address'
.....

RC      : 0=Ok      Result: Result-String

Write_Byte 'Address,Value'
.....

RC      : 0=Ok

Write_Word 'Address,Value'
.....

RC      : 0=Ok

Write_Long 'Address,Value'
.....

RC      : 0=Ok

ASL_FileRequester_Save
.....

RC      : 0=Ok      Result: Filepath-String

ASL_FileRequester_Load
.....
```

```
RC      : 0=Ok      Result: Filepath-String
```

```
IS_ADDRESS_LEGAL 'Address'
.....
```

```
* RC: -
```

```
* result: 'OK','FALSE'
```

```
LOAD_BINARY 'Name, Destination, Length'
.....
```

```
SAVE_BINARY 'Name, Source, Length'
.....
```

1.101 Barfly.guide/UB_HOWTOUSE

Usage of BDebug

How to use BDebug ?

=====

Problemanalysis

First you should be sure that all necessary configurations file have been installed because without Barfly.FD file you don't see any function names in the disassembler window; and without Barfly.Include the StructWindow is unusable. Are these preconditions fullfilled you should analyse the problem and anticipate how the Debugger can be used. Because the debugging of programs depends heavily on the situation i can only list some general points. The reality probably looks different... as always:-)

Point of departure:

* Program from the CLI

* 1 Task

The program can be started by bdebug Program [Argument] or can be loaded the command window Debug File. By this method all symbol and debug files are loaded. If the Debugger can't find one source file you can add additional paths if you haven't disabled this function. The standard breakpoint is the first command in the program. Sometimes this uncomfortable and so you can set a different start breakpoint for example to jump over the CStartup code or to set it on an important program position.

* Creates further Tasks

In this case you should be sure how you wanna catch the next

Task. You could catch the Task by Next Task or compile the program with an illegal in the task and catch it with Crashed Task. After you caught the task you probably would like to use see symbole and debug source. These Informationen can be loaded afterwards by using Load Symbols.

* Program from the WB

In this case you should use Next Task and then catch the task WBL. Afterwards you have to activate Next Task again and run WBL. You could also use the illegal strategic. After the right task was caught you can load the symbols again.

* Is it a Handler, Filesystem or something simular.

In this case you should use the illegal strategic and catch the task by Crashed Task. An alternative method would be to catch the waiting task with Debug Task and wait as long as the task gets woken up by a signal.

After you've solved the first stage how you can controll the problem task you should think about how the problem looks like and where it could be located.

Problem Type:

* Enforcer

If an Enforcer Hit is caused Enforcer outputs the hit's program address and mostly also the hunk offset. You can now directly jump to the address by entering the address in the DissWindow by Change Address or you open a HunkWindow, doubleclick on the hunk where the hit is located and then enter the offset Enforcer showed. The Debugger itself can also automatically stop a debugged program if a hit happens.

* Mungwall

These hits aren't that easy to find as Enforcer hits because Mungwall hits aren't showed when the problem happened but only after a FreeMem. In this case you should remember the memoryblock where it happened and determine where the responsible AllocMem is located in the program, so you get an overview between what area the problem is caused. Now you should open a MemWindow that points to this certain memory area and step through the program and look if something changes the mungwall borders in the MemWindow. Mungwall borders are before and after the allocated memory area. If you're more experienced you could also use the WatchpointWindow and set a watchpoint on the certain memory block.

* Crash

If it's just an ordinary crash the error should be pretty easy to find single stepping through the responsible code area. If it's a random crash you should be using Crashed Task and hope that the task can be caught. After the task got caught you should check the instructions that cause the crash. If the PC points to data

fields that don't look like real code the PC is probably set wrong by a stack cleanup error. In this case you should check the next addresses on the stack if these point to legal program code.

* Sideeffects and mysterious

These bugs are the worst to find and there's no general strategic how to find them. In these cases only intuition and patience can help.

Problems that can happen

- * Why does the Debugger react so slow on keyboard commands that controll the tracing ?

This happens if you debug a task with a higher priority as the Debugger's priority. For example. DOS-Handler. Workaround is to higher the priority of the Debugger.

- * Why does the Debugger blockate by opening the filerequester.

This happens when you debug a handler, because the filerequester normally tests every handler with a IsFilesystem(). When you debug a handler it can't reply the IsFilesystem packet and therefore the filerequester is busy.

- * If you debug the following program on a 68040 with 68040.library or 68060.library the instruction rts is run if you cause a Single-Step on the instruction fetox. Because this command isn't implemented in the 68040 and 68060 it has to be emulated. It seems to forget the tracebit.

```
mc68040
fmove.x #1.3,fp0
fmove.x #255,fp1
fetox.x fp0,fp1
rts
```

- * If a program doesn't return from a DOS funktion you could accidently entered a char in shell window.

1.102 Barfly.guide/BAsmTop

BAsm 1.0

A cli/arexx controlled Assembler

Copyright (c) 1989-94 Ralph Schmidt

- Shareware -

Using BAsm...

The Assembler	Some comments about the Assembler
Syntax	Syntax Description
Datatypes	Datatype formats
Operations	Datatype Operations
Instructions	Assembler Instructions
Macros	Assembler Basic Macros
Highlevel Macros	Assembler Highlevel Macros
Defined Symbols	Predefined Symbols
Optimizing	Optimize Methods
Includes	Precompiled Includes and database
CLI	Assembler CLI Convention
ARexx	ARexx Interface
Compatibility	Compatibility to other Assemblers

Addendum...

Literature	Amiga Literature
Software	Amiga Software
Addressmodes	Address Modes
Opcodes	68xxx Opcodes

1.103 Barfly.guide/UA_ASSEMBLER

The Assembler

The assembler understands the commands and addressmodes from the 68000 through the 68060 and both the Floating-Point Units, 68881 and 68882. It supports only the 68851 MMU commands, which are also supported by the 68030. The assembler achieves it's speed by translating the source in a single pass, followed by a backpatch phase which corrects all unresolved references.

1.104 Barfly.guide/UA_SYNTAX

Syntax
=====

Comments

A comment can start in several different ways. In a pure comment it starts either with a ; or *. A comment can only be started after an assembler command or symbol with a ; if an assembler command or symbol exists within that line.

```

;A comment
*A comment

```

```
move.l a0,a0 ;A comment
```

Opcode/Instructions arrangement

```
[label[:]] [opcode] [operand[, operand[, operand...]]]]
```

* Opcodes

An Opcode can be a Motorola Mnemonic, an assembler command, or a Macro call.

* Operations

In a Motorola mnemonic operands are based on legal addressmoes; in assembler privat instructions the parameters depend on the instruction.

Symbol structure

A symbol can represent the following types:

- * Value
- * Program Counter
- * Register
- * Register List
- * Macro

Symbols can only be defined once. The exceptions are local labels and symbols defined by Set.

Structure rules for Symbols.

- * The first letter of a symbol can be one of the following: a...z, A...Z, _, @, . and \.
- * From the second letter on, the symbol can contain the following letters: a...z, A...Z, 0...9, _, @ and ..
- * If a symbol consists of only numbers and ends with \$, then it is a numerical local label.
- * A symbol is ended by an illegal letter.
- * If a symbol begins with a . or \, then it is a local label.
- * A macro symbol should not contain a ..
- * To avoid a conflict, a symbol should not end with .b, .w or .l.

```
ThisIsALabel ;That is a normal label
```

```

ThisIsALabel1.loop:      ;That is a normal label
This@_Is_@A_@Label
1$:                      ;That is a numerical local label
.ThisIsALabel           ;That is a local label
.ThisIsALae11:          ;That is a local label
\ThisIsALabel.loop:     ;That is a local label
ThisIsASymbol=10
ThisIsASymbol = 10
ThisIsASymbol equ 10

```

The relative label

This symbol represents an offset to the start of a program.

```

label
label:
label nop
label: nop

```

The local Label

A local label is only valid between two normal relative labels, thus you cannot reference local labels outside of that scope. Otherwise it works similar as a normal label. There are 2 different prefixes that introduce a local label: `.` and `\` that define 2 different local labels. A special case is the Backward Reference Label that is introduced with `...`. It doesn't depend on a certain define area between normal labels thus you can only access the symbol if it were defined earlier.

```

...:
label_0:
.local:
bra.s .local
label_1:
.local:
bra.s \local
\local:
nop
label_end:
...:
dbra d0,..
...:
dbra d1,..
..Hello:
dbra d1,..Hello

```

The local numerical label

Additionally to the non-numerical local label there are also the numerical labels which are based of 4 digits with the postfix `$`. BASM handles the number as a hash key with the consequence that there's no difference between `001$` and `1$`.

```

label_0:
123$: nop
label_1:
123$: nop

```

The absolute Symbol

The absolute symbol is defined by a direct value initializing that is initiated by =, equ or set. If you define a symbol by set you can change it as often as needed.

```

value1=2
value2 equ value1*2
value3 set value2

```

The Register Symbol

The register symbol is defined by equr or fequr that is used for FPU registers.

```

Ptr equr a1
PI fequr fp2
move.l (Ptr),d0 ;move.l (a1),d0
fmove.x PI,fp0 ;fmove.x fp2,fp0

```

The Register List Symbol

The register list symbol is defined by reg and represents the register mask for Movem and fmovem. You must not mix FPU and Integer registers with each other in a register list.

```

mask reg d0/d2/d4-d7/a0-a4/a6-a7
mask1 reg d0-a6
mask2 reg d0-6
fmask reg fp0-fp2/fp4-fp5

```

The Macro Symbol

By using the command macro or cmacro after the symbol, the symbol is defined as a macro. The macro block is terminated by the command endm. The Macro cmacro is case-insensitive and therefore useful to emulate commands that are missing from the core.

```

Symbol[:] macro
.
.
Symbol[:] endm

```

1.105 Barfly.guide/UA_DATATYPES

Datatypes

The assembler understands 3 distinct datatypes.

- * 32Bit Integer
- * 96Bit Extended Floating Point
- * 96Bit Packed Binary Floating Point

At the moment only integer datatypes are supported in arithmetic arguments so you can only use the FPU datatypes as constants.

Format	Representation
Decimal	1024
Hexadecimal	\$400
Binary	%100000000000
Ascii	"OK", 'OK' or 'OK'

Furthermore you can use symbols or the character '*' that represents the program counter in arguments. There are limitation in the use of symbols in arguments. For example you can only add or subtract constants from an external label, Floatingpoint Values can only be used as simple constants,... By the postfix 'k' after decimal value the value is multiply by \$1000.

Floating Point Format

Format	Representation
Extended	'[+,-]3. 145637848298628[e[+,-]123]
Packed	''[+,-]3. 145637848298628[e[+,-]123]

Datatype Conversion

All commands are performed with these 3 datatypes and then converted into the required datatype. For example a 32Bit integer can be converted into 16Bit and 8 Bit; an extended floating point into a double or single floating point. Floating point datatypes are rounded by a conversion. If a rounding error occurs the parser returns with an error.

Datatype Format

Internal Datatype Structure

* Integer

```
+-----+-----+
| Bit 31 | 30..0   |
+-----+-----+
|      S | Integer |
+-----+-----+
```

* Single Floating Point

```
+-----+-----+-----+
| Bit 31 | Bits 30..23 | Bits 22..0 |
+-----+-----+-----+
|  Sign  | Biased Exponent | Fraction  |
+-----+-----+-----+
```

* Double Floating Point

```
+-----+-----+-----+
| Bit 63 | Bits 62..52   | Bits 51..0 |
+-----+-----+-----+
|  Sign  | Biased Exponent | Fraction  |
+-----+-----+-----+
```

* Extended Floating Point

```
+-----+-----+-----+
| Bit 95 | Bits 94..80    | Bits 62..0 |
+-----+-----+-----+
|  Sign  | Biased Exponent | Mantisse   |
+-----+-----+-----+
```

* Packed Binary Floating Point

```
+-----+-----+-----+-----+-----+-----+-----+
| MEYY | EXP2 | EXP1 | EXP0 | EXP3 | 0000 | 0000 | M016 |
+-----+-----+-----+-----+-----+-----+-----+
| M015 | M014 | M013 | M012 | M011 | M010 | M009 | M008 |
+-----+-----+-----+-----+-----+-----+-----+
```

```

| M007 | M006 | M005 | M004 | M003 | M002 | M001 | M000 |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- * M is the sign (+ or -) of the fraction
- * E is the sign (+ or -) of the exponent
- * Y are the internal flags for infinity and NAN
- * E002-000 are the numbers of the exponent from 2 to 0, EXP3 is used internally.
- * M016-000 are the numbers of the fraction from 16 to 0. Each number lies in the range from 0 to 9..

1.106 Barfly.guide/UA_OPERATIONS

Operations

Operators

Operator	Function
'+'	32Bit signed Addition
'-'	32Bit signed Subtraction
'*'	32Bit signed Multiplication
'/'	32Bit signed Division
' '	32Bit Or
'!'	32Bit Or
'&'	32Bit And
'^'	32Bit Eor
'<<'	logic 32Bit Shift to the left
'>>'	logic 32Bit Shift to the right
'~'	32Bit Not

Basm cares for the operator priorities but be careful while porting Seka Sources because Seka doesn't care for the priorities.

Functions

The following functions are supported.

- * `_bitnum(Argument)` calculates the bit number of the argument. If this is impossible an error occurs.
- * `_bitfield(Argument)` calculates the bit mask of the argument. If this is impossible an error occurs.
- * `_extb(Argument)` equal to the 680xx command `extb`
- * `_extw(Argument)` equal to the 680xx command `extw`
- * `_min(Argument[,Argument,...])` calculate the minimum of the argument.
- * `_max(Argument[,Argument,...])` calculate the maximum of the argument.

1.107 Barfly.guide/UA_INST

Assembler Commands

=====

Instruction Groups...

Hunk/Link Instructions
Symbol Instructions
Data Instructions
Listing Instructions
Structure Instructions
File I/O Instructions
Misc Instructions
68xxx Meta Instructions

1.108 Barfly.guide/AI_HUNK

Hunk/Link Commands

Section
Code
Data
BSS
CSEG
DSEG

```

IDNT
Identify
BDebugArg
Smalldata
XRef
XDef
Global
Public
Output
Objfile
Exeobj
Linkobj
Org
Addsym
Debug

```

1.109 Barfly.guide/HU_SECTION

```

[Label] section Name[, Typ[, [RelocModus, Memtyp]
.....

```

defines a new logical unit so that the DOS-Loader has the opportunity to place smaller hunks into free memory blocks. Another use for this command is to set different memory types for the hunk to load gfx data into the chipmem. If you don't specify the section type it assumes "", Code, Public.

* Name = Hunkname

* Hunk type

```

CODE
    starts a code segment.

```

```

DATA
    starts a data segment.

```

```

BSS
    starts an undefined data segment.

```

```

DEBUG
    starts a custom debug segment.

```

```

CUSTOM
    starts a Custom-Hunk area.

```

* Reloc-Mode defines the width of the hunk relocation. Default 32Bit

```

RELOC16
    sets the reloc width to 16bit.(V37)

```

```

RELOC32
    sets the reloc width to 32bit.(Default)

```

* Memtype defines the memory attributes of the hunk. If you add a `_p`, `_c`, or `_f` upon the type parameter, you cannot use more memtypes. The memtype `DEBUG` does not allow memory attribute suffixes.

`PUBLIC`
loads the hunk into the memory with the highest priority.
Code Suffix `_p`.

`CHIP`
loads the hunk into chip memory. Code Suffix `_c`.

`FAST`
loads the hunk into fast memory. Code Suffix `_f`.

`ADVISORY`
ignores the hunk if the OS doesn't understand the type. A kind of Debug-Hunk that can be used by the OS. (V39)

`ATTR=?`
loads the hunk into the memory with specified memory attributs. (V37)

1.110 Barfly.guide/HU_CODE

[Label] code [Name[, Memtyp]]
.....

defines a new code hunk and is equivalent to the command section `?,code,?`.

* Name = Hunkname

* Memtype defines the memory attributes for the hunk.

`PUBLIC`
loads the hunk into the memory with the highest priority.
Code Suffix `_p`.

`CHIP`
loads the hunk into chip memory. Code Suffix `_c`.

`FAST`
loads the hunk into fast memory. Code Suffix `_f`.

`ADVISORY`
ignores the hunk if the OS doesn't understand the type. A kind of Debug-Hunk that can be used by the OS. (V39)

`ATTR=?`
loads the hunk into the memory with specified memory attributs. (V37)

1.111 Barfly.guide/HU_DATA

```
[Label] data [Name[, Memtyp]]
.....
```

defines a new data hunk and is equivalent to the command section
?,data,?.

* Name = Hunkname

* Memtype defines the memory attributes for the hunk.

```
PUBLIC
    loads the hunk into the memory with the highest priority.
    Code Suffix _p.
```

```
CHIP
    loads the hunk into chip memory. Code Suffix _c.
```

```
FAST
    loads the hunk into fast memory. Code Suffix _f.
```

```
ADVISORY
    ignores the hunk if the OS doesn't understand the type. A
    kind of Debug-Hunk that can be used by the OS. (V39)
```

```
ATTR=?
    loads the hunk into the memory with specified memory
    attributs. (V37)
```

1.112 Barfly.guide/HU_BSS

```
[Label] bss [Name[, Memtyp]]
.....
```

defines a new BSS hunk and is equivalent to the command section
?,bss,?.

* Name = Hunkname

* Memtype defines the memory attributes for the hunk.

```
PUBLIC
    loads the hunk into the memory with the highest priority.
    Code Suffix _p.
```

```
CHIP
    loads the hunk into chip memory. Code Suffix _c.
```

```
FAST
    loads the hunk into fast memory. Code Suffix _f.
```

```
ADVISORY
```

ignores the hunk if the OS doesn't understand the type. A kind of Debug-Hunk that can be used by the OS. (V39)

ATTR=?

loads the hunk into the memory with specified memory attributs. (V37)

1.113 Barfly.guide/HU_CSEG

[Label] cseg [Name[, Memtyp]]

.....

has the same function as the command code.

1.114 Barfly.guide/HU_DSEG

[Label] dseg [Name[, Memtyp]]

.....

has the same function as the command data.

1.115 Barfly.guide/HU_IDNT

idnt Name

.....

defines the name of the HUNK_UNIT hunk in the object file.

* Name = Hunkname

1.116 Barfly.guide/HU_IDENTIFY

identify Name

.....

defines the name of the actual hunk.

* Name = Hunkname

1.117 Barfly.guide/HU_BDEBUGARG

BDebugArg Argument

.....

defines a parameter in env:BDebugProgram. It doesn't activate this function you have to activate by option "-J" in BOPT.

* Argument = Argument Text

1.118 Barfly.guide/HU_SMALLDATA

smalldata [Register]

.....

activates the smalldata mode for the hunk. Optionally, you can also define the smalldata register. Default register is A4. The program itself must initialize the smalldata register with the address of the smalldata data hunk.

```
bopt w2-      ;68020 Addressmode warnings off
mc68020      ;68020 mode activated
```

```
smalldata a3      ;Default is A4!!!
xref _LinkerDB    ;Special linker symbol
```

```
lea.l _LinkerDB,a3      ;Address of the smalldata data segments
move.l #0,(d_test.l,a3)
move.l #"TEST",d_test(a3)
moveq #0,d0
tst.b array(a3,d0.w)
rts
```

```
section "__MERGED",BSS ;The smalldata data segments are defined
                        ;the following way
```

```
d_test:
  ds.l 1
array:
  ds.b 20
```

1.119 Barfly.guide/HU_XREF

xref Symbol[, Symbol...]

.....

imports a symbol so that you can access symbols that were exported by XDef. The linker resolves these reference during the link process and creates a program file. If the assembler finds a XRef in the source

it creates an object file. This decision can be overruled.

* Symbol = Name of the importet symbol.

1.120 Barfly.guide/HU_XDEF

```
xdef Symbol[, Symbol...]  
.....
```

exports a symbol as global so that other object files can import the symbol by XRef. There's no need to define a symbol before you mark them with XDef. If the assembler finds a XRef in the source it creates an object file. This decision can be overruled.

* Symbol = Name of the global symbol

1.121 Barfly.guide/HU_GLOBAL

```
global Symbol[, Symbol...]  
.....
```

has the same function like XDef.

1.122 Barfly.guide/HU_PUBLIC

```
public Symbol[, Symbol...]  
.....
```

has the same function like XDef.

1.123 Barfly.guide/HU_OUTPUT

```
output Name  
.....
```

sets an output filename. If you don't specify a filename the assembler uses the source filename and adds the appropriate filetype suffix.

* Name = Filename

1.124 Barfly.guide/HU_OBJFILE

objfile
.....

has the same function like Output.

1.125 Barfly.guide/HU_EXEOBJ

exeobj
.....

writes a program file if you wanna overrule the assembler.

1.126 Barfly.guide/HU_LINKOBJ

linkobj
.....

writes an object file if you wanna overrule the assembler.

1.127 Barfly.guide/HU_ORG

org Address
.....

activates the absolute mode. All command that refer to hunk related functions aren't allowed. For example: . section, xdef, xref. The parameter address sets the base address of the created code.

* Address = Absolute Address

1.128 Barfly.guide/HU_ADDSYM

addsym
.....

writes a symbol hunk.

1.129 Barfly.guide/HU_DEBUG

```
debug
.....
```

writes a SAS D1 debug hunk to see source level informations while debugging the program through bdebug.

1.130 Barfly.guide/AI_SYMBOL

Symbol Commands

```
CArgs
RS
SO
FO
RSReset
RSSet
Clrso
Clrfo
Setso
Setrs
Setfo
RSVal
SOVal
FOVal
```

1.131 Barfly.guide/SY_CARGS

```
CArgs [#Offset,]Symbol[,Symbol.w[,Symbol.l]
.....
```

defines the symbol offsets for a stack function. The first Symbol starts with the offset 4 but if you like to use a different Offset it's possible to specify one. Then the offset is increased according to the size of the symbol. If the symbol has no size specifier the default size is word. Sorry..i would use a longword here but to be compatible with Devpac i'm forced to use word.

```
cargs Test1.w,Test2.l
move.w Test1(a7),d0 ;Test1=4
move.l Test2(a7),d0 ;Test2=4+2=6
```

1.132 Barfly.guide/SY_RS

```
Symbol rs[.width] Count
.....
```

initializes the Symbol with the value of the counter `__RS` and increases the `__RS` counter afterwards by `Count*Width`. You can use this command as a replacement for the include `exec/types.i` macros to increase the parsing speed.

* Width

B

1 Byte Valuearea: $-\$80 \leq x < \100

W

2 Bytes Valuearea: $-\$8000 \leq x < \10000

L

4 Bytes Valuearea: $-\$80000000 \leq x < \10000000

S

4 Bytes (Single IEEE-Float)

D

8 Bytes (Double IEEE-Float)

X

12 Bytes (Extended IEEE-Float)

P

12 Bytes (Packed BCD-Float)

Q

16 Bytes (Quadword)

1.133 Barfly.guide/SY_SO

```
Symbol so[.width] Count
.....
```

This command has the same function like `rs` with the exception that the Symbol `__SO` is used instead of the `__RS` symbol. Internally both symbols are handled equal. Devpac has introduced the symbol `__RS` and Macro68k knows the functionality by the name `__SO`.

1.134 Barfly.guide/SY_FO

```
Symbol fo[.width] Count
.....
```

decreases the counter `__FO` by `Count*Width` and initializes the Symbol with the new value. Useful to create the negative local stackframe symbols needed by link.

```
* Width

B
    1 Byte Valuearea: -$80 <= x < $100

W
    2 Bytes Valuearea: -$8000 <= x < $10000

L
    4 Bytes Valuearea: -$80000000 <= x < $100000000

S
    4 Bytes (Single IEEE-Float)

D
    8 Bytes (Double IEEE-Float)

X
    12 Bytes (Extended IEEE-Float)

P
    12 Bytes (Packed BCD-Float)

Q
    16 Bytes (Quadword)
```

1.135 Barfly.guide/SY_RSRESET

```
rsreset
.....
```

initializes the counter `__RS` to 0.

1.136 Barfly.guide/SY_RSSET

```
rsset Value
.....
```

initializes the counter `__RS` with the Value

```
* Value = New Index
```

1.137 Barfly.guide/SY_CLRSO

```
clrso
.....
```

has the same function like rsreset

1.138 Barfly.guide/SY_CLRFO

```
clrfo
.....
```

has the same function like foreset.

1.139 Barfly.guide/SY_SETSO

```
setso Value
.....
```

has the same function like rsset

1.140 Barfly.guide/SY_SETRS

```
setrs Value
.....
```

has the same function like rsset

1.141 Barfly.guide/SY_SETFO

```
setfo Value
.....
```

initializes the counter __FO with the Value

* Value = New Index

1.142 Barfly.guide/SY_RSVAL

Symbol rsval
.....

initializes the Symbol with the value of the __RS counter.

1.143 Barfly.guide/SY_SOVAL

Symbol soval
.....

has the same function like rsval.

1.144 Barfly.guide/SY_FOVAL

Symbol foval
.....

initializes the Symbol with the value of the __FO counter.

1.145 Barfly.guide/AI_DATA

Data Commands

Align
CNoP
Pad
Quad
Even
Odd
DC
DB
DW
DL
UB
UW
UL
SB
SW
SL
PB
PW
PL
NB
NW

```
NL
DS
DSB
DCB
Blk
Ascii
CString
DString
PString
IString
Bitstream
SPrintx
```

1.146 Barfly.guide/DA_ALIGN

```
align Value
.....
```

aligns the program counter to an address that can be divided by the value. Useful because certain DOS structures have to be aligned on 4 Byte boundaries. For example FileInfoBlock. Furthermore it's also useful to align subroutines on longword boundaries that they fit better into the cache structure.

* Value = Align Mask

1.147 Barfly.guide/DA_CNOP

```
cnop Offset,Align
.....
```

aligns the program counter to an address that can be divided by the Align value and adds the value onto the address. Internally only Align values < 16 are supported.

1.148 Barfly.guide/DA_PAD

```
pad.Width Align[,Value]
.....
```

aligns the program counter to an address that can be divided by the Align*Width and fills the aligned area by the optional mask value.

1.149 Barfly.guide/DA_QUAD

```
quad
....
```

aligns the program counter to a 16 Byte address.

1.150 Barfly.guide/DA_EVEN

```
even
....
```

aligns the program counter to an even address. This function is useful if you define an odd sized data area and you need a word aligned for OS data structures or assembler instructions.

1.151 Barfly.guide/DA_ODD

```
odd
...
```

aligns the program counter to an odd address.

1.152 Barfly.guide/DA_DC

```
dc[.width] Value[,Value...]
.....
```

inserts data of the Width into the code.

* Width

B

1 Byte Valuearea: $-\$80 \leq x < \100

W

2 Bytes Valuearea: $-\$8000 \leq x < \10000

L

4 Bytes Valuearea: $-\$80000000 \leq x < \100000000

S

4 Bytes (Single IEEE-Float)

D

8 Bytes (Double IEEE-Float)

X
12 Bytes (Extended IEEE-Float)

P
12 Bytes (Packed BCD-Float)

1.153 Barfly.guide/DA_DB

```
db Value[,Value,...]  
.....
```

inserts a byte with a value in the valuearea $-\$80 \leq x < \100 .

1.154 Barfly.guide/DA_DW

```
dw Value[,Value,...]  
.....
```

inserts a word with a value in the valuearea $-\$8000 \leq x < \10000 .

1.155 Barfly.guide/DA_DL

```
dl Value[,Value,...]  
.....
```

inserts a longword with a value in the valuearea $-\$80000000 \leq x < \100000000 .

1.156 Barfly.guide/DA_UB

```
ub Value[,Value,...]  
.....
```

inserts a byte with a value in the valuearea $-\$80 \leq x < \80 .

1.157 Barfly.guide/DA_UW


```
uw Value[,Value,...]  
.....
```

inserts a word with a value in the valuearea $-\$8000 \leq x < \8000 .

1.158 Barfly.guide/DA_UL

```
ul Value[,Value,...]  
.....
```

inserts a longword with a value in the valuearea $-\$80000000 \leq x < \80000000 .

1.159 Barfly.guide/DA_SB

```
sb Value[,Value,...]  
.....
```

inserts a byte with a value in the valuearea $-\$80 \leq x < \100 .

1.160 Barfly.guide/DA_SW

```
sw Value[,Value,...]  
.....
```

inserts a word with a value in the valuearea $-\$8000 \leq x < \10000 .

1.161 Barfly.guide/DA_SL

```
sl Value[,Value,...]  
.....
```

inserts a longword with a value in the valuearea $-\$80000000 \leq x < \100000000 .

1.162 Barfly.guide/DA_PB

```
pb Value[,Value,...]  
.....
```

inserts a byte with a value in the valuearea $0 \leq x < \$80$.

1.163 Barfly.guide/DA_PW

```
pw Value[,Value,...]  
.....
```

inserts a word with a value in the valuearea $0 \leq x < \$8000$.

1.164 Barfly.guide/DA_PL

```
pl Value[,Value,...]  
.....
```

inserts a longword with a value in the valuearea $0 \leq x < \$80000000$.

1.165 Barfly.guide/DA_NB

```
nb Value[,Value,...]  
.....
```

inserts a byte with a value in the valuearea $-\$80 \leq x < 0$.

1.166 Barfly.guide/DA_NW

```
nw Value[,Value,...]  
.....
```

inserts a word with a value in the valuearea $-\$8000 \leq x < 0$.

1.167 Barfly.guide/DA_NL

```
nl Value[,Value,...]  
.....
```

inserts a longword with a value in the valuearea $-\$80000000 \leq x < 0$.

1.168 Barfly.guide/DA_DS

```
ds[.width] Count[,Value]
.....
```

defines a memory area with the length $\text{Count} * \text{Width}$ and fills the area with an optional Value. Default fill value is 0. Is the Count 0 a cnop 0,Width is run.

* Width

B

1 Byte Valuearea: $-\$80 \leq x < \100

W

2 Bytes Valuearea: $-\$8000 \leq x < \10000

L

4 Bytes Valuearea: $-\$80000000 \leq x < \100000000

S

4 Bytes (Single IEEE-Float)

D

8 Bytes (Double IEEE-Float)

X

12 Bytes (Extended IEEE-Float)

P

12 Bytes (Packed BCD-Float)

* Count = Length of the memory area.

* Value = optional Fill Value.

1.169 Barfly.guide/DA_DSB

```
dsb[.width] Count[,Value]
.....
```

has the same function like ds

1.170 Barfly.guide/DA_DCB

```
dsb[.width] Count[,Value]
.....
```

has the same function like ds

1.171 Barfly.guide/DA_BLK

```
blk[.width] Count[,Value...]
.....
```

has the same function like ds

1.172 Barfly.guide/DA_ASCII

```
ascii String1[,String2,...]
.....
```

inserts Strings.

1.173 Barfly.guide/DA_CSTRING

```
cstring String1[,String2,...]
.....
```

inserts C-Strings.

1.174 Barfly.guide/DA_DSTRING

```
dstring dtype1,dtype2,dtype3
.....
```

inserts the current date string.

Datentypen

* "w" WeekDay

* "d" Date

* "t" Time

```
dc.b " ("
dstring w,d,t
dc.b ")"
dc.b $a,$d,0
;=> (Thursday 14-Okt-93 15:32:06)
```

1.175 Barfly.guide/DA_PSTRING

```
pstring String[,String,...]
.....
```

inserts a BCPL string.

1.176 Barfly.guide/DA_ISTRING

```
istring String[,String,...]
.....
```

inserts strings that terminate with a char that has Bit 7 set.

1.177 Barfly.guide/DA_BITSTREAM

```
bitstream Mask
.....
```

inserts a bitmask for an image object for example. The bits are aligned to bytes.

* Mask = Mask is a string that is based only of 0 and 1.

```
bitstream "01001000001"
```

1.178 Barfly.guide/DA_SPRINTX

```
sprintx "Formatstring"[,Value[,...]]
.....
```

inserts the resulting string into the code. The string isn't terminated by a 0 so that you can add other strings rather easy.

* FormatString - is a string in C-Notation so you can use the known C char types n,t,...

The following options are allowed.

* FormatSyntax - %[flags][width][.limit][length]type

* flags - '-' deactivates left side layout.

* width - Field Length. If the first char is '0' the field is filled by '0' on the left side.

- * limit - defines the maximal count of char that can be inserted from a string. Only legal for %s and %b.
- * length - The size of the datatype. Default is 16-bit for the types %d,%u and %x. %l is long (32Bit). Attention! The Assembler always pushes a longword on the stack so always use %l if you don't know what you're doing.
- * type - The following types are supported.
 - b - BSTR, a 32-bit BPTR Pointer on a bytelength string. A NULL BPTR is handled like an empty string.
 - d - signed decimal
 - u - unsigned decimal
 - x - hexadecimal in lower case.
 - X - hexadecimal in upper case.
 - s - String, a 32-bit Pointer on a NULL-terminated Byte-String. A NULL BPTR is handled like an empty string.
 - c - Char
- * Value - is an argument that has to be resolvable.

1.179 Barfly.guide/AI_LISTING

Listing I/O Commands

```
List
Nolist
Printx
Lisfile
```

1.180 Barfly.guide/LI_LIST

```
list
....
```

activates the listing output. Has no function if the global listing output wasn't activated.

Listing Format:

```
LINE ADDRESS[Flag1] COMMAND-BYTES[Flag2] SOURCE
```

* Flag1

- * + shows that the line was created by a macro.
- * > shows that the Assembler searches the closing ENDC.
- * < shows that the Assembler searches the closing ENDM.

* Flag2

* + shows a line overflow and that Bytes are ignored. Can often happen during data definitions.

1.181 Barfly.guide/LI_NOLIST

nolist
.....

disactivates the listing output. Has no function if the global listing output wasn't activated.

1.182 Barfly.guide/LI_PRINTX

printx "Formatstring"[,Value[,...]]
.....

outputs the string to the current Stdout and works similar as the known C-Printf function. Look at SPRINTF

errfile Name
.....

defines the filename for the error output.

* Name = Filename

1.183 Barfly.guide/LI_LISFILE

lisfile Name
.....

defines the filename for the listing output. If no error file was defined the error output is also written into the listing file.

* Name = Filename

1.184 Barfly.guide/AI_STRUCTURE

Structuring

```
Macro
Endm
MExit
Fail
End
If
Ifd
Ifnd
Ifv
Ifnv
Ifmacrod
Ifmacrond
Ifcmacrod
Ifcmacrond
Ifc
Ifnc
If[cc]
Else
Elseif
Endc
Endif
Repeat
Rept
Procstart
Procend
```

1.185 Barfly.guide/ST_MACRO

```
symbol[:] macro
.....

    starts a Macro block.
```

1.186 Barfly.guide/ST_ENDM

```
endm
....

    ends a macroblock.
```

1.187 Barfly.guide/ST_MEXIT

```
mexit
.....

    ends a macro call.
```

1.188 Barfly.guide/ST_FAIL

```
fail
....

    creates an error.
```

1.189 Barfly.guide/ST_END

```
end
...

    ends the assembling.
```

1.190 Barfly.guide/ST_IF

```
if Symbol
.....

    checks if the symbol value is not NULL and assembles the block
    depending on the success.
```

1.191 Barfly.guide/ST_IFD

```
ifd Symbol
.....

    checks if the Symbol exists and assembles the block depending on the
    success.
```

1.192 Barfly.guide/ST_IFND

```
ifnd Symbol
.....

    checks if the Symbol doesn't exist and assembles the block depending
    on the success.
```

1.193 Barfly.guide/ST_IFV

ifv String
.....

This is a privat command that is used for internal functionality and subject to change. Touch an burn!

1.194 Barfly.guide/ST_IFNV

ifnv String
.....

This is a privat command that is used for internal functionality and subject to change. Touch an burn!

1.195 Barfly.guide/ST_IFMACROD

ifmacrod Macro
.....

checks if the Macro exists and assembles the block depending on the success.

1.196 Barfly.guide/ST_IFMACROND

ifmacrond Macro
.....

checks if the Macro doesn't exist and assembles the block depending on the success.

1.197 Barfly.guide/ST_IFCMACROD

ifcmacro CMacro
.....

checks if the CMacro exists and assembles the block depending on the success.

1.198 Barfly.guide/ST_IFCMACROND

```
ifcmacrond CMacro  
.....
```

checks if the CMacro doesn't exist and assembles the block depending on the success.

1.199 Barfly.guide/ST_IFC

```
ifc Symbol,Symbol  
.....
```

compares the first string with the second string and if they are equal the block is assembled.

1.200 Barfly.guide/ST_IFNC

```
ifnc 'String','String'  
.....
```

compares the first string with the second string and if they differ the block is assembled.

1.201 Barfly.guide/ST_IFCC

```
if[condition] Symbol=Symbol  
.....
```

compares the first symbol with the second symbol and decides according to the condition if the block is assembled.

* Condition = Normal Bcc-Condition Syntax

* Symbol = Normal Symbol

1.202 Barfly.guide/ST_ELSE

```
else  
....
```

activates the condition block if the block above wasn't assembled.

1.203 Barfly.guide/ST_ELSEIF

```
elseif  
.....
```

activates the condition block if the block above wasn't assembled.

1.204 Barfly.guide/ST_ENDC

```
endc  
....
```

defines the end of a condition block.

1.205 Barfly.guide/ST_ENDIF

```
endif  
.....
```

defines the end of a condition block.

1.206 Barfly.guide/ST_REPEAT

```
repeat Count  
.....
```

repeats the blocks that is located between repeat and endr by the number Count.

1.207 Barfly.guide/ST_REPT

```
rept Count  
.....
```

has the same function like Repeat

1.208 Barfly.guide/ST_PROCSTART

```
procstart
.....
```

defines a function in a Dice-C assembler output and is used to optimize Link and Unlk. This optimize method isn't working yet.

1.209 Barfly.guide/ST_PROCEND

```
procend
.....
```

defines a function in a Dice-C assembler output and is used to optimize Link and Unlk. This optimize method isn't working yet.

1.210 Barfly.guide/Al_FILE

File I/O Commands

```
Incdir
Incpath
Include
Include2
Incbin
Incbin2
IBytes
DSBin
Doscnd
Pure
```

1.211 Barfly.guide/FI_INCDIR

```
incdir Dir[,Dir[,...]]
.....
```

adds directories to the include path list. BASM uses 2 internal path lists and the current directory to find the include and incbin files. First BASM checks for a : character in the filename and if it finds a volume the file is loaded direct instead of searching it through the pathlists. The first path list contains the paths that were defined in the commandline or BOPT by the option -i or through incdir. The second path list contains the paths that were defined in global configuration file ENV:BASMOption. The entries of the second list will be removed when the assembler is closed so that the paths are still correct in the ARexx-Mode. The first list is removed every pass.

* Dir = Name of the Include-Path.

1.212 Barfly.guide/FI_INCPATH

Incpath
.....

has the same function like incdir.

1.213 Barfly.guide/FI_INCLUDE

include Name
.....

loads the external include file, for example the OS-Includes. If the file is a precompiled include file it's detected automaticly. Includes are loaded from the editor or cachefile.library.

* Name = Filename

1.214 Barfly.guide/FI_INCLUDE2

include2 Name
.....

has the same function like include with the exception that the cachefile.library isn't ignored.

* Name = Filename

1.215 Barfly.guide/FI_INCBIN

incbin Name[,size]
.....

inserts the file with the optional length at the current address into the code. Normally used for sounds and graphics.

1.216 Barfly.guide/FI_INCBIN2

```
incbin2 Name[,size]
.....
```

has the same function like incbin with the exception that the `cachefile.library` isn't used.

* Name = Name of the data file.

1.217 Barfly.guide/FI_IBYTES

```
ibytes Name[,Length]
.....
```

has the same function like incbin

1.218 Barfly.guide/FI_DSBIN

```
dsbin Name[,Length]
.....
```

defines a memory area with the length of the file specified by the file. Optionally you can define the maximal file length.

* Name = Filename

* Length = maximal file length

1.219 Barfly.guide/FI_DOSCMD

```
doscmd Name
.....
```

runs the program Name.

```
dc.b 0, "$VER: Fubar 1.0 by Joe User"
doscmd "c:date >ram:Temp"
incbin ram:Temp
doscmd "c:delete ram:Temp"
```

1.220 Barfly.guide/FI_PURE

```
pure
....
```

sets the Pure Bit while writing a program file.

1.221 Barfly.guide/AI_MISC

```
Miscellaneous
-----
```

```
Trashreg
Super
MC[Type]
Bopt
```

1.222 Barfly.guide/MI_TRASHREG

```
trashreg Reglist
.....
```

defines the registers that are available to the optimizer.

* RegList = A normal Registerlist known by Movem.

1.223 Barfly.guide/MI_SUPER

```
super
.....
```

disactivates Supervisor warnings.

1.224 Barfly.guide/MI_MCXXX

```
mc[Type]
.....
```

defines the processor type to allow certain commands and addressmodes.

Processor-Type

- * 68000 Default Mode
- * 68010
- * 68020
- * 68030
- * 68040
- * 68060
- * 68881
- * 68882

1.225 Barfly.guide/MI_BOPT

```
bopt [opt[,...],...]  
.....
```

sets the assembler options.

Options

m1[+,-]
activates/disactivates the 68010 mode.

m2[+,-]
activates/disactivates the 68020 mode.

m3[+,-]
activates/disactivates the 68030 mode.

m4[+,-]
activates/disactivates the 68040 mode.

m6[+,-]
activates/disactivates the 68060 mode.

mf[+,-]
activates/disactivates the 68881/2 mode.

ue[+,-]
activates/disactivates writing an executable file.

uo[+,-]
activates/disactivates writing an object file.

ua[+,-]
activates/disactivates writing an absolut file.

un[+,-]

activates/disactivates writing file.

p[+,-]
activates/disactivates writing a preassembled Include file.

g[+,-]
activates/disactivates adding the prefix _ to each exported symbol.

sx[+,-]
activates/disactivates writing all XRef/XDef symbols to a symbol hunk.

sl[+,-]
activates/disactivates writing all normal symbols to a symbol hunk.

sa[+,-]
activates/disactivates writing all symbols to a symbol hunk.

sd[+,-]
activates/disactivates writing a BASM custom format Debug Hunk.
Makes only sense as a program file and it needs a lot of space
because it includes all sources.

sl[+,-]
activates/disactivates writing a SAS D1 compatible Debug Hunk.

sf[+,-]
activates/disactivates writing the full sourcefile path into the
debug hunk. You should only use this for your own development
system because other users may have different HD layouts. This
option has only a meaning within a SAS D1 Debug Hunk.

j[+,-]
activates/disactivates setting the PURE Bit for a program file.
The PURE Bit tells the Shell that this program can be loaded
resident.

J[+,-]
activates/disactivates creating the file ENV:BDebugProgram that
contains the assembled filename for BDebug.

a[+,-]
activates/disactivates creating of an .info file for each program.
Useful if you use the assembler through the WB.

A[+,-]
activates/disactivates the ARexxmode Option. Only allowed in the
commandline.

i<DirName>
defines the include path.

o<FileName>
defines the object filename

P<Priority>
sets the task priority.

`c[+,-]`
activates/disactivates that the assembler interpretes Upper and Lower case as 2 different chars.

`f[+,-]`
activates/disactivates a faster mode that resolves all references in the 2nd pass. Fortunately this mode needs more memory and has some disadvantages like uncorrect values during the listing. This option has no effect during optimizing.

`M<Bytes>`
defines the max macro expansion size. If you get a macromemerror you should increase the size. Default 1000 Bytes.

`Z<Address>`
tells the assembler that the source is starts in the memory at the defined address. Useful for ARexx scripts. Option is only available in the commandline.

`x[+,-]`
uses the cachefile.library to load resident Includes/Incbins or add unknown files to the cachefile.library database.

`X[+,-]`
erases all files that are controlled by the cachefile.library.

`y[+,-]`
shows all files that are controlled by the cachefile.library.

`l[+,-]`
activates/disactivates the listing output.

`l0[+,-]`
activates/disactivates the listing macro expansion.

`L<Listingfile>`
defines the Listing filename.

`h[+,-]`
activates/disactivates the symbol listing output.

`H[+,-]`
activates/disactivates the unused symbol output.

`v[+,-]`
outputs a statistic after assembling.

`V[+,-]`
as little status output as possible

`e[+,-]`
creates an error list.

`es[+,-]`
outputs the error list in the Barfly shell. This option has no meaning in BASM.

wo[+,-]
 activates/disactivates Optimizing warnings.

ws[+,-]
 activates/disactivates Supervisor warnings.

wm[+,-]
 activates/disactivates Move16 warnings because the use of the
 move16 command is dangerous if you don't know the problems.

w2[+,-]
 activates/disactivates 68020 Addressmode warnings.

w4[+,-]
 activates/disactivates 64k-Access warnings. It's useful if you
 accidently avoid to forget the address register. Example: move.l
 8,d0 instead of move.l 8(an),d0

b0
 sets the Default Branch Length to 8-Bit. .b

b1
 sets the Default Branch Length to 16-Bit. .w

b2
 sets the Default Branch Length to 32-Bit. .l

B0
 sets the Default BaseDisplacement-Width to 8 Bit. .b

B1
 sets the Default BaseDisplacement-Width to 16 Bit. .w

B2
 sets the Default BaseDisplacement-Width to 32 Bit. .l

n0
 sets the Default OuterDisplacement-Width to 16 Bit. .w

n1
 sets the Default OuterDisplacement-Width to 32 Bit. .l

q[+,-]
 activates/disactivates align long after each rts, bra or jmp to
 align blocks to the cache structure.

O[+,-]
 activates/disactivates the Optimizer. Without this option no
 optimizing will happen besides the addressmode converting.

OG[+,-]
 activates/disactivates Forward Reference Optimizing to use every
 possibility. In this mode the source is assembled until no further
 optimizing method is found. First the source is assembled
 normally. This is shown by the Output Pass 1. Afterwards the
 optimize passes are started and continued until no further symbol

changes and length errors occur. This can take a while and depends on the source size.

OT[+,-]
activates/disactivates Time Optimizing.

Addressmode Converting

OC0[+,-]
bdwan

OC1[+,-]
bdwpc

OC2[+,-]
anxn

OC3[+,-]
pcxn

OC4[+,-]
bdw

OC5[+,-]
bd1

OC6[+,-]
an

OC7[+,-]
pc

ODD[+,-]
activates Direct Addressmode Optimizing

Direct Optimizing

OD0[+,-]
move

OD1[+,-]
clr

OD2[+,-]
add

OD3[+,-]
sub

OD4[+,-]
lea

OD5[+,-]
cmp

OD6[+,-]
bcc

```
OD7[+,-]
    jsr

OD8[+,-]
    jmp

OD9[+,-]
    asl

ODa[+,-]
    or ( This Optimizing is disactivated internal )

ODb[+,-]
    eor ( This Optimizing is disactivated internal )

ODc[+,-]
    and

ODd[+,-]
    mulu

ODE[+,-]
    muls

ODf[+,-]
    jsr+rts

ODg[+,-]
    jmp+rts

ODh[+,-]
    MovemNoRegister

ODi[+,-]
    MovemOneRegister

ODj[+,-]
    Link

OAP[+,-]
    activates PC-Relative Optimizing

OAS[+,-]
    activates Smalldata Optimizing

OAL[+,-]
    activates long nach word Optimizing

OAX[+,-]
    activates x(An) to (An) Optimizing

OAY[+,-]
    activates 68020++ An-EA Optimizing

OAZ[+,-]
    activates 68020++ PC-EA Optimizing
```

OAR[+,-]
 activates Register Optimizing

You should be careful with the command BOPT when you activate Global-Optimize. In every parse the default config is set and therefore you should define all global options in the commandline or in the configuration file.

1.226 Barfly.guide/AI_META

680xx Meta Commands

```
mb
mw
ml
mq
xor
xori
bhs
blo
```

1.227 Barfly.guide/ME_MB

mb Operand1,Operand2

.....

has the same function as move.b.

1.228 Barfly.guide/ME_MW

mw Operand1,Operand2

.....

has the same function as move.w.

1.229 Barfly.guide/ME_ML

ml Operand1,Operand2

.....

has the same function as move.l.

1.230 Barfly.guide/ME_MQ

mq Operand1,Operand2
.....

has the same function as moveq.

1.231 Barfly.guide/ME_XOR

xor.? Operand1,Operand2
.....

has the same function as eor.?.

1.232 Barfly.guide/ME_XORI

xori.? Operand1,Operand2
.....

has the same function as eori.?.

1.233 Barfly.guide/ME_BHS

bhs.? Label
.....

has the same function as bcc.?.

1.234 Barfly.guide/ME_BLO

blo.? Label
.....

has the same function as bcs.?.

1.235 Barfly.guide/UA_MACRO

Assembler Macros

=====

Macros are meta commands that can be based of many assembler instructions to achieve an abstracter source layout. In a macro you can use several different pattern that are replaced by appropriate parameters when the macro is called. The parameter that are passed during a macro call are represented by the following patterns: \0,...,\9, \a,...,\z, \A,...,\Z. The pattern ids are using the hexadecimal format. If a pattern is used with no related parameter an empty string is inserted. Furthermore if a parameter contains tabulators or spaces it has to be placed between <...>. When a macro needs relative labels and is should be called more than one time you should use the special pattern @. This pattern is replaced by a number that is based of 4 digits and that is increased after each call. The pattern \# is replaced by the value of the symbol narg that represents the count of macro parameters. Besides the standard patterns there are some more advanced pattern functions supported that look like *Function-Name. These functions don't belong to the motorola standard thus they aren't supported by every assembler. Another important point is that you can also call macros from from macros but you can't define macros in macros.

The standard macro pattern

```
Label & [. string] & [, string] & [, string] & [,...]          & [\0]
& [\1|] & [\2|] & [\3|]...[\n|]
```

The advanced macro pattern functions

- * \ (Argument) inserts the string of the macroparameter with the number the argument defines.

```
\(1) = \1
\ (1+3+4) = \8
```

- * *upper(String) inserts the string in upper case.
- * *lower(String) inserts the string in lower case.
- * *valof(Argument) inserts the decimal value of the argument as a string.
- * *strlen(Symbol) inserts the length of a symbol as a string.
- * *right(String,n) inserts n chars of the right side of the string. If the string contains less than n chars the whole string is inserted.
- * *left(String,n) inserts n chars of the left side of the string. If the string contains less than n chars the whole string is inserted.

* *mid(String,n,m) inserts chars from position n to m from the string. If the position is outside of the string length the chars till the end of the string is inserted.

```
openwind MACRO
    move.l    intbase,a6
    lea.l     \1,a0
    jsr       OpenWindow(a6)
ENDM
```

```
start:
    openwind    newwindow
```

```
movewind MACRO
    move.l    intbase,a6
    move.l    \1,a0
    moveq     #0,d0
    move.\0   \2,d1
    IFC      '\0','b'
    ext.w     d1
    ENDC
    jsr       MoveWindow(a6)
ENDM
```

```
start:
    move.b    #10,d2
1$:
    movewind.b newwindow,d2
    addq.b    #1,d2
    cmp.b     #100,d2
    bne.s     1$
```

```
wait MACRO
    moveq     #-1,d0
wait\
    dbra     d0,wait\
ENDM
```

```
start:
    wait
    wait
    wait
```

```
test MACRO
    move.l    #\*upper(Hello),d0
    move.l    #\*lower(Hello),d0
    move.l    #\*strlen(1234567890123456),d0
    move.l    #\*valof(value),d0
    rts

    cstring   "\*left(abcdefgh,4) "
    even
```

```

        cstring    "\*left (abcdefgh,10) "
        even
        cstring    "\*right (abcdefgh,4) "
        even
        cstring    "\*right (abcdefgh,10) "
        even
        cstring    "\*mid (abcdefgh,2,4) "
        even
        cstring    "\*mid (abcdefgh,2,8) "
        even

        ENDM

value = 123456789
hello:
        test

        value = 123456789

hello:
        move.l     #HELLO,d0
        move.l     #hello,d0
        move.l     #16,d0
        move.l     #123456789,d0
        rts
        cstring    "abcd"
        even
        cstring    "abcdefgh"
        even
        cstring    "efgh"
        even
        cstring    "abcdefgh"
        even
        cstring    "cdef"
        even
        cstring    "cdefgh"
        even

PUTTAG MACRO
        IFC        "\2", ""
PUTTAG_COUNT set 0
        ENDC

        IFNC        "\2", ""
        move.l     \2,-(a7)
PUTTAG_COUNT SET PUTTAG_COUNT+4
        ENDC

        move.l     \1,-(a7)
PUTTAG_COUNT SET PUTTAG_COUNT+4

        IFC        "\1", "#TAG_END"
PUTTAG_COUNT SET 4
        ENDC

```

```

                                ENDM

CLEARTAG MACRO
    lea.l        PUTTAG_COUNT(a7),a7
                                ENDM

                                PUTTAG    #TAG_END
                                PUTTAG    #WA_Width,#100
                                PUTTAG    #WA_ScreenTitle,#Title
                                .
                                .
                                move.l    a7,a1
                                sub.l     a0,a0
                                jsr       OpenWindowTagList(a6)
                                CLEARTAG

```

1.236 Barfly.guide/UA_HMACROS

Highlevel Macros
=====

In highlevel macros the operands are based of legal addressmodes. Arguments are based of operands and the operators +,-,<<,>>. Conditions are based of !,=, <,>,<=,>=,<>. By using highlevel macros you can make the programming of non critical source areas easier and more abstract. Blame Mike Schwartz for this idea...he forced me to do it:-B

```

.Reg
.Branch
.For
.Next
.If
.Else
.Endif
.While
.Endwhile
.Call
.Return
.Def
.Enddef
.Let

```

1.237 Barfly.guide/HM_REG

```

.REG
....

```

sets the accumulator register that is used to calculate arguments.
Default register is D0.

1.238 Barfly.guide/HM_BRANCH

```
.BRANCH b|w|l
.....
```

sets the length of branch commands that are used in the highlevel macros. Standard length is .b.

1.239 Barfly.guide/HM_FOR

```
.FOR Operand[.b|w|l] = Operand TO Operand STEP Operand
.....
```

creates code for a for loop. The optional width you define after the first operand sets the width for all operations in the for loop.

```
.FOR    d0.w = #1 to  STEP #2
addq.w #1,d1
.NEXT

;Compiled Code

move.w #1,d0
__for1:
addq.w #1,d1
add.w  #2,d0
cmp.w  ,d0
blt.b  __for1
```

1.240 Barfly.guide/HM_NEXT

```
.NEXT
.....
```

closes the outer .FOR loop.

1.241 Barfly.guide/HM_IF

```
.IF [Argument] =,!, < , > , <> Operand
.....
```

creates code for an IF-Operation. You can remove the first argument if you wanna test the operand. For example `.IF <>`

```
.IF      (a0) + #0 <> d1
moveq    #0,d0
.ELSE
moveq    #1,d0
.ENDIF
```

;Compiled Code

```
move.l   (a0),d7
add.l    #0,d7
cmp.l    d1,d7
beq.b    __else1
moveq    #0,d0
bra.b    __endif1
__else1:
moveq    #1,d0
__endif1:
```

1.242 Barfly.guide/HM_ELSE

```
.ELSE
.....
```

starts an alternative IF-Block.

1.243 Barfly.guide/HM_ENDIF

```
.ENDIF
.....
```

closes the outer `.IF` block.

1.244 Barfly.guide/HM_WHILE

```
.WHILE [Argument] =,!, < , > , <> Operand
.....
```

creates code for a while loop The optional width you define after the first operand sets the width for all operations in the while loop.

```

        .WHILE d0 <> #0
        addq.w #1,d1
        .ENDWHILE

;Compiled Code

__while1:
    cmp.l    #0,d0
    beq.s    __endwhile1
    addq.w   #1,d1
    bra.s    __while1
__endwhile1:

```

1.245 Barfly.guide/HM_ENDWHILE

```

.ENDWHILE
.....

```

closes the outer while loop.

1.246 Barfly.guide/HM_CALL

```

.CALL Function [, Argument [, Argument [,...]]]
.....

```

calls a C-Funktion by parsing the arguments through the stack.
Arguments are calculated in the accumulator register.

```

.CALL func , test + 0 - #20 , #test

;Compiled Code

move.l test,d7
add.l 0,d7
sub.l #20,d7
move.l d7,-(a7)
move.l #test,-(a7)
jsr func
ifnc "8","0" ;Were there any parameters ?
lea.l __CALLSize(a7),a7
endc

```

1.247 Barfly.guide/HM_RETURN

`.RETURN` Argument

returns a result value in the accumulator register.

```
.return d1 + d2 + #$100
```

;Compiled Code

```
move.l d1,d7
add.l d2,d7
add.l #$100,d7
```

1.248 Barfly.guide/HM_DEF

`.DEF` func [, Operand [, Operand [...]]]

defines a C-Stack function and loads the defined parameters into the operands.

```
.DEF func , d0.w , d1 , (a0)
.ENDDEF
```

;Compiled Code

```
XDEF func
link a5,#0
move.w $0a(a5),d0
move.l $0c(a5),d1
move.l $10(a5),(a0)
unlk a5 ;.ENDDEF
rts
```

1.249 Barfly.guide/HM_ENDDEF

`.ENDDEF`

closes a function that was started by `.DEF`

1.250 Barfly.guide/HM_LET


```
.LET [ Operand =] Argument
.....
```

calculates an argument in an accumulator or moves the value to a defined operand.

```
.LET + 4 - #LN_SIZE << #7
.LET d1 = (a1) - (a0)
```

```
;Compiled Code
```

```
add.l 4,d7
sub.l #LN_SIZE,d7
lsl.l #7,d7
```

```
move.l (a1),d7
sub.l (a0),d7
move.l d7,d1
```

1.251 Barfly.guide/UA_SYMBOLS

Predefined Symbols
=====

NARG
....

represents the macro parameter count in a macro.

BARFLY
.....

represents the assembler version.

680xx
.....

represents the CPU processor type.

6888x
.....

represents the FPU processor type.

_MOVEMBYTES
.....

represents the byte count the last movem transfer used.

```
lea _MOVEMBYTES(a7),a7 ;frees the stack
```

`_MOVEMREGS`

.....

represents the last movem register mask.

```
movem (a7)+,_MOVEMREGS
```

`__RS`

....

represents the RS-Counter.

`__SO`

....

represents the RS-Counter.

`__FO`

....

represents the FO-Counter.

1.252 Barfly.guide/UA_OPTIMIZING

Optimizing

=====

Optimize Methods...

```
Direct Address Optimizing
Address Optimizing
#x Optimizing
Register Optimizing
How it works ?
Problems
```

1.253 Barfly.guide/OP_DIRECT

Direct Addressmode Optimizing

The assembler can direct optimize certain 68020...60 Addressmodes if a faster 68000 addressmode exists. This optimizing method should always be activated because of compatibility reasons.

```
*      (bd.w,an) can be optimized to x(an) that removes 1 word and
      some cycles.
```

Addressmode	Optimizing	Option
move. l (1000.w,an),dn	move.l 1000(an),dn	-OC0

- * (bd.w,pc) can be optimized to x(pc) that removes 1 word and some cycles.

Addressmode	Optimizing	Option
move. l (1000.w,pc),dn	move.l 1000(pc),dn	-OC1

- * (bd.w) can be optimized to bd.w that removes 1 word and some cycles.

Addressmode	Optimizing	Option
move. l (bd.w),dn	move.l bd.w,dn	-OC4

- * (bd.l) can be optimized to bd.l that removes 1 word and some cycles.

Addressmode	Optimizing	Option
move. l (bd.l),dn	move.l bd.l,dn	-OC5

- * (an) can be optimized to (an) that removes 1 word and some cycles. The addressmode (an) can be interpreted as a subgroup of (bd,an,xn). Because (an) is a normal 68000 addressmode you should never switch off this optimizing method.

Addressmode	Optimizing	Option
move. l (an),dn	move.l (an),dn	-OC6

- * (pc) can be optimized to (pc) that removes 1 word and some cycles. The addressmode (pc) can be interpreted as a subgroup of (bd,pc,xn). Because (pc) is a normal 68000 addressmode you should never switch off this optimizing method.

Addressmode	Optimizing	Option
-------------	------------	--------

move.l (pc),dn	move.l (pc),dn	-OC7
----------------	----------------	------

1.254 Barfly.guide/OP_ADDRESS

Address Optimizing

* Long

Addressmode	Optimizing	Note	Option
x.l,EA	x.w,EA	\$ffff8000<=x<=\$7fff	-OAL
EA,x.l	EA,x.l	\$ffff8000<=x<=\$7fff	

* x(an)

Addressmode	Optimizing	Note	Option
x(an),EA	(an),EA	x=0	-OAX
EA,x(an)	EA,(an)	x=0	

* PC-Relative

Addressmode	Optimizing	Note	Option
label,EA	label(pc),EA	\$ffff8000<=x<=\$7fff	-OAP

* A4-Smalldata

Addressmode	Optimizing	Note	Option
x.l,EA	x(a4),EA	\$ffff8000<=x<=\$7fff	-OAS
EA,x.l	EA,x(a4)	\$ffff8000<=x<=\$7fff	

* 68020-An

--	--	--	--

Addressmode	Optimizing	Note	Option
(x.l, an)	(an)	x=0	-OAY
(x.l, an)	x(an)	\$ffffff80<=x<=\$7f	
(x, an, xn)	0(an, xn)	x=0	
(x, an, xn)	(x.b, an, xn)	\$ffffff80<=x<=\$7f	
(x, an, xn)	(x.w, an, xn)	\$ffff8000<=x<=\$7fff	
([?], x)	([?])	x=0 ? is also optimized	
([?], x)	([?], x.w)	\$ffff8000<=x<=\$7fff ? is also optimized	
([?], xn, x)	([?], xn)	x=0 ? is also optimized	
([?], xn, x)	([?], xn, x.w)	\$ffff8000<=x<=\$7fff ? is also optimized	

* 68020-PC

Addressmode	Optimizing	Note	Option
(x.l, pc)	(pc)	x=0	-OAZ
(x.l, pc)	x(pc)	\$ffffff80<=x<=\$7f	
(x, pc, xn)	0(pc, xn)	x=0	
(x, pc, xn)	(x.b, pc, xn)	\$ffffff80<=x<=\$7f	
(x, pc, xn)	(x.w, pc, xn)	\$ffff8000<=x<=\$7fff	
([?], x)	([?])	x=0 ? is also optimized	
([?], x)	([?], x.w)	\$ffff8000<=x<=\$7fff ? is also optimized	
([?], xn, x)	([?], xn)	x=0 ? is also optimized	
([?], xn, x)	([?], xn, x.w)	\$ffff8000<=x<=\$7fff ? is also optimized	

1.255 Barfly.guide/OP_OPTIMIZE

#x Optimizing

* Move

Addressmode	Optimizing	Note	Option
move.l #x,dn	moveq #x,dn	\$ffffff80<=x<=\$7f	-OD0
move.? #0,an	suba.l an,an	? = w or l	
move.l #x,dn	moveq #y,dn swap dn	\$10000<=x<=\$7f0000	
move.l #x,dn	moveq #y,dn swap dn	\$ff80ffff<=x<=\$fffEffff	
move.l #x,dn	moveq #y,dn neg.b dn	\$80<=x<=\$ff	
move.l #x,dn	moveq #y,dn neg.w dn	\$ffff<=x<=\$ff81	
move.l #x,dn	moveq #y,dn neg.w dn	\$ffff0080<=x<=\$ffff0001	
move.? #0,EA	clr.? EA	? = w or l. See Trashreg optimizing. I also check if it accesses the HW	
move.b #\$ff,EA	st EA		
movea.l -4(an),an	movea.l -(an),an		

* Clr

Addressmode	Optimizing	Note	Option
clr.l dn	moveq #0,dn		-OD1

* Add

Addressmode	Optimizing	Note	Option
add.? #x,EA	addq.? #x,EA	1<=x<=8	-OD2

add.? #x,EA	subq.? #x,EA	-8<=x<=-1	
+-----+	+-----+	+-----+	+-----+
add.? #x,an	lea.l x(an),an	\$ffff8000<=x<=\$7fff	
+-----+	+-----+	+-----+	+-----+
add.? #0,EA	tst.? EA	legal EA	
+-----+	+-----+	+-----+	+-----+
add.? #0,an	removed		
+-----+	+-----+	+-----+	+-----+

* Sub

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
sub.? #x,EA	subq.? #x,EA	1<=x<=8	-OD3
+-----+	+-----+	+-----+	+-----+
sub.? #x,EA	addq.? #x,EA	-8<=x<=-1	
+-----+	+-----+	+-----+	+-----+
sub.? #x,an	lea.l -x(an),an	\$ffff8000<=x<=\$7fff	
+-----+	+-----+	+-----+	+-----+
sub.? #0,EA	tst.? EA	legal EA	
+-----+	+-----+	+-----+	+-----+
sub.? #0,an	removed		
+-----+	+-----+	+-----+	+-----+

* Lea

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
lea x(an),an	addq.w #x,an	1<=x<=8	
+-----+	+-----+	+-----+	+-----+
lea x(an),an	subq.w #x,an	-8<=x<=-1	-OD4
+-----+	+-----+	+-----+	+-----+

* Cmp

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
cmp.? #0,EA	tst.? EA		-OD5
+-----+	+-----+	+-----+	+-----+

* Bcc

The assembler tries to optimize the branch on the smallest possible length so that can win max 2 words and some cycles.

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
Bcc.l label	Bcc.w label	\$8000<=label<=\$7fff	-OD6
+-----+	+-----+	+-----+	+-----+

Bcc.l label	Bcc.s label	\$80<=label<=\$7f	
+-----+	+-----+	+-----+	+-----+
Bcc.w label	Bcc.s label	\$80<=label<=\$7f	
+-----+	+-----+	+-----+	+-----+

Attention! This optimizing method is unsafe when you use
BRANCH-Tables. You should switch off the optimize method over
this area.

* Jsr

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
jsr label	bsr.w label	\$8000<=Offset<=\$7fff	
+-----+	+-----+	+-----+	+-----+
jsr label	bsr.s label	\$80<=Offset<=\$7f	-OD7
+-----+	+-----+	+-----+	+-----+

Attention! This optimizing method is unsafe when you use
JSR-Tables. You should switch off the optimize method over this
area.

* Jmp

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
jmp label	bra.w label	\$ffff8000<=Offset<=\$7fff	
+-----+	+-----+	+-----+	+-----+
jmp label	bra.s label	\$ffffff80<=Offset<=\$7f	-OD8
+-----+	+-----+	+-----+	+-----+

* Asl

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
asl.? #1,dn	add.? dn,dn		-OD9
+-----+	+-----+	+-----+	+-----+

* Or This optimizing method isn't safe because of the changed
condition flags.

Addressmode	Optimizing	Note	Option
+-----+	+-----+	+-----+	+-----+
or.? #x,dn	bset #y,dn	x=y^2	-ODa
+-----+	+-----+	+-----+	+-----+

* Eor This optimizing method isn't safe because of the changed

condition flags.

Addressmode	Optimizing	Note	Option
eor.? #x,dn	bchg #y,dn	$x=y^2$	-ODb

* Mulu

Be very careful with this optimizing.

Addressmode	Optimizing	Note	Option
mulu.w #x,dn	swap dn	$x=2^y$	-ODc
	clr.w dn	$y=y1+y2$	
	swap dn	$y=1, \text{add.l dn,dn}$	
	lsl.l #y1,dn		
	lsl.l #y2,dn		
mulu.l #x,dn	lsl.l #y1,dn	$x=2^y$	
	lsl.l #y2,dn	$y=y1+y2$	
		$y \geq 16$	
		swap dn , y-16	

* Muls

Be very careful with this optimizing.

Addressmode	Optimizing	Note	Option
muls.w #x,dn	ext.l dn	$x=2^y$	-ODd
	asl.l #y1,dn	$y=y1+y2$	
	asl.l #y2,dn	$y=1, \text{add.l dn,dn}$	
muls.l #x,dn	asl.l #y1,dn	$x=2^y$	
	asl.l #y2,dn	$y=y1+y2$	
		$y \geq 16$	
		swap dn , y-16	

* Jsrr+Rts

Addressmode	Optimizing	Note	Option
jsr EA	jmp EA	No optimizing if there's	-ODE
rts		a label before RTS	

+-----+-----+-----+-----+			
* Bsr+Rts			
+-----+-----+-----+-----+			
Addressmode	Optimizing	Note	
+-----+-----+-----+-----+			
Option			
+-----+-----+-----+-----+			
jmp EA	jmp EA	No optimizing if there's	-ODf
rts		a label before RTS	
+-----+-----+-----+-----+			

* MovemNoRegister			
+-----+-----+-----+-----+			
Addressmode	Optimizing	Note	Option
+-----+-----+-----+-----+			
movem.l ,EA	Removed		
			-ODh
movem.l EA,	Removed		
+-----+-----+-----+-----+			

* MovemOneRegister			
+-----+-----+-----+-----+			
Addressmode	Optimizing	Note	Option
+-----+-----+-----+-----+			
movem.l Xn,EA	Move.l Xn,EA	Alter the status flags!	
			-ODi
movem.l EA,Xn	Move.l EA,Xn	Alter the status flags!	
+-----+-----+-----+-----+			

1.256 Barfly.guide/OP_REGISTER

Register Optimizing

- * #xxx is switched off.
- * An address register is set free by trashreg.

+-----+-----+-----+-----+↔			
Addressmode	Optimizing	Note	Option ↔
+-----+-----+-----+-----+↔			
+-----+-----+-----+-----+↔			

move.? EA, label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	move.? EA, (an)	-OAR	↵
+-----+-----+-----+-----+			↵
tst.? label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	tst.? (an)		↵
+-----+-----+-----+-----+			↵
not.? label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	not.? (an)		↵
+-----+-----+-----+-----+			↵
neg.? label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	neg.? (an)		↵
+-----+-----+-----+-----+			↵
negx.? label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	negx.? (an)		↵
+-----+-----+-----+-----+			↵
nbcd label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	nbcd (an)		↵
+-----+-----+-----+-----+			↵
scc label	lea.l label(pc), an	\$ffff8000<=label<=\$7fff	↵
	scc (an)		↵
+-----+-----+-----+-----+			↵

* #x Optimizing on.

* An address register is set free by trashreg.

+-----+-----+-----+-----+↵			
Addressmode	Optimizing	Note	Option↵
+-----+-----+-----+-----+↵			
+-----+-----+-----+-----+↵			
move.l #x,EA	moveq #x,dn	\$ffffff80<=x<=\$7f	↵

		move.l dn,EA		-OAR	↵
+-----+	+-----+	+-----+	+-----+		↵
ori.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	or.l dn,EA				↵
+-----+	+-----+	+-----+	+-----+		↵
eori.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	eor.l dn,EA				↵
+-----+	+-----+	+-----+	+-----+		↵
andi.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	and.l dn,EA				↵
+-----+	+-----+	+-----+	+-----+		↵
addi.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	add.l dn,EA				↵
+-----+	+-----+	+-----+	+-----+		↵
subi.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	sub.l dn,EA				↵
+-----+	+-----+	+-----+	+-----+		↵
cmpi.l #x,EA	moveq #x,dn		\$ffffff80<=x<=\$7f		↵
	cmp.l EA,dn				↵
+-----+	+-----+	+-----+	+-----+		↵
move.? #0,EA	moveq #0,dn		Time optimizing		↵
	move.l dn,EA		must be on		↵
+-----+	+-----+	+-----+	+-----+		↵

1.257 Barfly.guide/OP_HOWDOESITWORK

How does Optimizing work ?

In single-pass Optimizing the assembler can only optimize commands where it can resolve the reference in the first pass. This means the

label or symbol has to be known before. In multi-pass Optimizing it can optimize every command without bothering where the label is defined. The Assembler keeps all labels each pass but increases a change counter if the old contents differs with the new contents. The exception is that the assembler can't optimize commands that depend on symbols that are defined after the command. The reason is that the Assembler has to remove each pass every symbol to avoid problems with IFD and IFND that can cause that certain areas aren't assembled in multi-pass mode. You probably noticed that I assume that nobody used IFD or IFND with labels because that would also break multi-pass.

An example for a construct that can't be optimized.

```
    move.l #1, NULL(a0)
NULL=0
```

1.258 Barfly.guide/OP_PROBLEMS

Problems...

You should always be careful with optimizing because it can cause bugs in certain source areas. Branch optimizing for example has to be switched off if you use JMP-Towers.

```
    lsl.w #1,d0
    jmp   Tower(pc,d0.w)
```

```
Tower:
    bra.w func1
    bra.w func2
    bra.w func3
```

could be optimized to

```
Tower:
    bra.w func1
    bra.s func2
    bra.s func3
```

that leads to program bugs.

Solution:

bopt OD6-

```
Tower:
    bra.w func1
    bra.w func2
    bra.w func3
```

bopt OD6+

1.259 Barfly.guide/UA_PRE

Preassembled Includes
=====

If you wanna assemble a program that needs to load a lot includes it's useful to preassemble the includes and load one file because the real slowdown factor is the need loading time. You can only use absolut symbols and macros in a preassembled file. All relative and symbols defined by set aren't written into a preassembled file. The created file isn't compress to avoid any slowdown but if the file size is critical you can compress the file by xpk and load it through the xfh filesystem.

basm -p Source.S creates the preassembled file Source.p

An error location could be absolut symbols that are calculated by relative symbols. You should avoid these symbols.

Symbol=Label1-Label2

Resident includes
=====

BASM can control an Include and Incbin database by the library cachefile.library to get rid of the loading delays. The files in the database can be shown and deleted.

1.260 Barfly.guide/UA_CLI

Basm Assembler System
=====

Cli Calling Convention

Format:

BASM [-Option] Name

This is the commandline version of the assembler and can be easy integrated in own development system, for example Make and CED. An assembler error is indicated by the result 20 and the result 10 is used if no source file were specified.

* Option

The same options are accepted that are described in the assembler command BOPT. The following options are accepted additionally.

- * A[+,-] Turns ARexx mode on/off
 - * C <Configuration> loads a configuration file
 - * d <Symbol=Value> defines a symbol
 - * Standard-Optionen
- ;All other options are deactivated.
- c+,e+,m1000,wo+,ws+,wm+,w2+,w4+
 b1+,B0+,n1+
 OC0+,OC1+,OC2+,OC3+,OC4+,OC5+,OC6+,OC7+,
 ODD+,OD0+,OD1+,OD2+,OD3+,OD4+,OD5+,OD6+,OD7+,OD8+,
 OD9+,ODc+,ODe+,ODf+,ODg+
 OAP+,OAL+,OAX+,OAY+,OAZ+,OAR+

Configuration

Global

.....

You can define the global configuration through the file ENV:BASMOption. The internal standard configuration is not replaced but can only be changed.

File ENV:BASMOption

```
-v
-f
-c-
-iASM:
```

If a line starts with - it's ignored.

WB Tooltypes

.....

Additionally you can also define the above described configuration options in the tooltypes of the source file icon. Furthermore BASM allows a special tooltype to define an output window.

o Window= <Window Defintion>

1.261 Barfly.guide/UA_AREXX

ARexx
=====

The BASM ARexx Port Name is BASM_rexx and the ARexx Script suffix .basm. To activate the BASM ARexx mode you have to start BASM with the option -A.

BASM
....

BASM [-Option] Name

This ARexx command starts the assembler and coincides with the CLI-syntax structure.

BEND
....

BEND

This ARexx command closes the ARexx port and shuts down the assembler.

BGETERROR
.....

BGETERROR

With this ARexx command you will receive an explanation of the actual errors. If no errors exist it will return a status code 20.

Error Format String

OFFSET:FILE:<Error Description>

BNEXTERROR
.....

BNEXTERROR

This ARexx command will cause a jump to the next error in the list. If there are no further errors in the list it will return a status code 20.

BINITERROR
.....

BINITERROR

This ARexx command will cause a jump to the first entry in the error list. If no error exists it will return a status code 20.

1.262 Barfly.guide/UA_COMP

Compatibility

=====

to other assemblers...

Fortunately BASM can't be 100% compatible to every assembler on the amiga market. Thus you can expect problems with different sources. In general you can expect problems with commands that don't belong to a standard like option commands. Furthermore you should also be careful with sources that directly depend on the assembler implementation. Because BASM is a 1-Pass Assembler in the normal mode with an additional backpatch phase you shouldn't define symbols that can't be resolved at once. An ideal example for this practice is the Xoper2.2 Source that was developed with the PD Assembler A68k. While assembling with BASM the assembler detects that a not defined symbols is accessed through the SET command. Generally this should cause an error at once but unfortunately A68k doesn't show anything and uses the last value of cmdnum.

```

ADDCMD      MACRO
cmdnum      set      cmdnum+1
            dc.b      \1,0
            ENDM

            .
            .
            .
;Here it's using 'cmdnum' although
;the symbol wasn't defined yet
            addq      #1,d2
            cmp.w     #cmdnum,d2
            bne.s     1$
            .
            .
            .

;Here the cmdnum is first defined
cmdnum      set      0
commds      ADDCMD   'time'
            ADDCMD   'taskpri'
            ADDCMD   'info'
            ADDCMD   'pri'
            ADDCMD   'flush'

```

Because the A68k is a 2-Pass Assembler he can assemble this Source without problems.

Another Problem is that the assembler argument parser doesn't detect Overflow because of speed reasons. I don't think it's worth

it...if you differ tell me your opinion.

The assembler doesn't support the following motorola syntax bugs because of the internal structure of the parser it would cause major problems in the multi-pass mode.

```
symbol: equ 0
symbol: equ d0
```

C-Compiler Assembler

Dice

....

If you use Basm as a DASM replacement you have to run Basm with the option -OAS to activate the Smalldata mode. If you wanna emulate the advanced Link,UnLink, Movem optimizing DASM supports you have to use the options -O,-OG,-ODh,-ODi,-ODj. The option -OG is needed because the link stackframe register list symbols are defined after the commands so the assembler doesn't know them in the 1 pass mode. Sorry...i had to disable this mode because i later detected that i have to keep track of the used registers. I'll try to fix this in a later version

1.263 Barfly.guide/UA_LITERATURE

Literature

- * [Addison Wesley] RKM Libraries 2.04,CATS
- * [Addison Wesley] RKM Devices 2.04,CATS
- * [Addison Wesley] RKM Autodocs\&Includes 2.04,CATS
- * [Addison Wesley] RKM Hardware 2.04,CATS
- * [Addison Wesley] RKM Styleguide,CATS
- * [Addison Wesley] RKM Libraries 1.1,CATS
- * [Addison Wesley] RKM Intuition 1.1,CATS
- * [Addison Wesley] RKM Exec 1.1,CATS
- * [Addison Wesley] RKM Hardware 1.1,CATS
- * [Edotronik] Kommentiertes Rom-Listing 1,Dr. Ruprecht
- * [Edotronik] Kommentiertes Rom-Listing 2,Dr. Ruprecht
- * [Edotronik] Kommentiertes Rom-Listing 3,Dr. Ruprecht

* [Ralph Babel] Guru Book, Selbstvertrieb

1.264 Barfly.guide/UA_SOFTWARE

Software

For the development of Barfly the following programs were used:

* [CATS] Developer CD V2.0

* [B.Hawes] WShell V2.0

* [M.Sinz] Enforcer

* [C.Scheppner] Mungwall

* [SAS Institute] SAS/C

* [GNU] GCC

* [ASDG] CED

* [Georg Hessmann] PasTex

* [Stefan Stuntz] MFR 2.0d

* [Mathias Scheler] Filer

* [Matthew Dillon] DNet

1.265 Barfly.guide/UA_EA

Assembler Addressmodes

Notation	Description
EA	Effective Address
Dn	D0...D7
An	A0...A7
Xn	D0...D7, A0...A7
.b	Operand Width 8Bit
.w	Operand Width 16Bit
.l	Operand Width 32Bit
size	w,l
Size	b,w,l

Scale	1,2,4 or 8
Xn.size*Scale	68000-10 only Scale 1.

Data register direct

Syntax: Dn

Address register direct

Syntax: An

Address register indirect

Syntax: (An)

Address register indirect with postincrement

Syntax: (An)+

Address register indirect with predecrement

Syntax: -(An)

Address register indirect with offset

Syntax: bd.w(An)

Address register indirekt with index and offset

Syntax: bd.b(An,Xn{.Size*Scale})

Address register indirect with index and offset

Syntax: (bd,An,Xn{.Size*Scale})

Address register indirect with index and offset

Syntax: (bd.b,An,Xn{.Size*Scale})

Address register indirect with index and base displacement

Syntax: ({bd.size{,An{,Xn{.Size{*Scale}}}}})

Indirekter Memory Addressierung mit postindex

Syntax: ({[{bd.size{,An}}]}{Xn{.Size{*Scale}}{,od.size}}})

Indirekter Memory Addressierung mit preindex

Syntax: ({[{bd.size{,An}}]{,Xn{.Size{*Scale}}}{,od.size}}})

PC Indirect

Syntax: (PC)

PC Indirect with offset

Syntax: bd.w(PC)

PC Indirect with index and offset

Syntax: bd.b(PC,Xn{.Size*Scale})

PC Indirect with index and offset

Syntax: bd.b(ZPC,Xn{.Size*Scale})

PC Indirect with index and base displacement

Syntax: ({bd.size{,PC{,Xn{.Size{*Scale}}}}})

PC Indirect with index and base displacement

Syntax: ({bd.size{,ZPC{,Xn{.Size{*Scale}}}}})

PC Indirect memory Addressing with post-index

Syntax: ({[{bd.size{,PC}}]},{Xn{.Size{*Scale}}{,od.size}}})

PC Indirect memory Addressing with post-index

Syntax: ({[{bd.size{,ZPC}}]},{Xn{.Size{*Scale}}{,od.size}}})

PC Indirect memory Addressing with pre-index

Syntax: ({[{bd.size{,PC}}]},{Xn{.Size{*Scale}}{,od.size}}})

PC Indirect memory addressing with pre-index

Syntax: ({[{bd.size{,ZPC}}]},{Xn{.Size{*Scale}}{,od.size}}})

Absolut short

Syntax: bd.w

Absolut long

Syntax: bd[.l]

Immediate Data

Syntax: #xxx

Addressmode Examples

=====

To avoid some problems here are some small examples how addressmode have to build up.

```
x=$40
y=$400
move.b (x,A0,D2.W),D0
move.b x(A0,D2.W),D0

;Both lines are correct
;(x,a0,d2.w) is optimized internal to (x,a0,d2.w).
;For more information please check the chapter about
;Optimizing Direct Addressmodes.

move.b (y,A0,D2.W),D0
move.b y(A0,D2.W),D0

;Now you get 2 errors, because y is not an 8bit word.
;These 2 lines shows the correct version.

move.b (y.w,A0,D2.W),D0
move.b (y.w,A0,D2.W),D0

;or

move.b (y.l,A0,D2.W),D0
move.b (y.l,A0,D2.W),D0
```

1.266 Barfly.guide/UA_OPCODES

680xx Opcode Overview

Opcode	Size	68000	68010	68020	68030	68040	68060	6888x
abcd	b	x	x	x	x	x	x	
add	b,w,l	x	x	x	x	x	x	
addq	b,w,l	x	x	x	x	x	x	
adda	w,l	x	x	x	x	x	x	
addi	b,w,l	x	x	x	x	x	x	
addx	b,w,l	x	x	x	x	x	x	
and	b,w,l	x	x	x	x	x	x	
andi	b,w,l	x	x	x	x	x	x	
asr	b,w,l	x	x	x	x	x	x	
asl	b,w,l	x	x	x	x	x	x	
bcc	b,w,l	x	x	x	x	x	x	
bchg	b,l	x	x	x	x	x	x	
bclr	b,l	x	x	x	x	x	x	
bfchg	unsized			x	x	x	x	
bfclr	unsized			x	x	x	x	
bfext	unsized			x	x	x	x	
bfffo	unsized			x	x	x	x	
bfins	unsized			x	x	x	x	
bfset	unsized			x	x	x	x	
bftst	unsized			x	x	x	x	
bkpt	unsized		x	x	x	x	x	
bset	b,l	x	x	x	x	x	x	
btst	b,l	x	x	x	x	x	x	
callm	unsized			x				
cas	b,w,l	x	x	x	x	x	x,2	
cas2	b,w,l	x	x	x	x	x	x,2	
chk	b,w,l	x	x	x	x	x	x	
chk2	b,w,l			x	x	x	2	
cinvs\$^1\$	unsized					x	x	
clr	b,w,l	x	x	x	x	x	x	
cmp	b,w,l	x	x	x	x	x	x	
cmpa	w,l	x	x	x	x	x	x	
cmpi	b,w,l	x	x	x	x	x	x	
cmpm	b,w,l	x	x	x	x	x	x	
cmp2	b,w,l			x	x	x	2	
cpush\$^1\$	unsized					x	x	
dbcc	w	x	x	x	x	x	x	
divs	w,l	x	x	x	x	x	x,2	
divsl	l			x	x	x	x	
divu	w,l	x	x	x	x	x	x,2	
divul	l			x	x	x	x	
eor	b,w,l	x	x	x	x	x	x	
eori	b,w,l	x	x	x	x	x	x	
eori/ccr	b	x	x	x	x	x	x	
eori/sr\$^1\$	w	x	x	x	x	x	x	
exg	l	x	x	x	x	x	x	
ext	w,l	x	x	x	x	x	x	

extb	1		x	x	x	x	
fabs					x	x	x
fsabs					x	x	
fdabs					x	x	
facos					2	2	x
fadd					x	x	x
fsadd					x	x	
fdadd					x	x	
fasin					2	2	x
fatana					2	2	x
fatanh					2	2	x
fbcc					x	x	x
fcmp					x	x	x
fcos					2	2	x
fcosh					2	2	x
fdbcc					x	2	x
fdiv					x	x	x
fsdiv					x	x	
fddiv					x	x	
fetox					2	2	x
fetoxml					2	2	x
fgetexp					2	2	x
fgetman					2	2	x
fint					2	x	x
fintrz					2	x	x
flogn					2	2	x
flognp1					2	2	x
flog2					2	2	x
flog10					2	2	x
fmod					2	2	x
fmove					x	x	x
fsmove					x	x	
fdmove					x	x	
fmovecr					2	2	x
fmovem					x	x,2	x
fmul					x	x	x
fsmul					x	x	
fdmul					x	x	
fneg					x	x	x
fsneg					x	x	
fdneg					x	x	
fnop					x	x	x
frem					2	2	x
frestore\$^1\$						x	x
fscc					2	2	x
fsub					x	x	x
fssub					x	x	
fdsub					x	x	
fsave\$^1\$						x	x
fscale					2	2	x
fsglmul					2	2	x
fsgldiv					2	2	x
fsin					2	2	x
fsinh					2	2	x
fsincos					2	2	x
fsqrt					x	x	x
fssqrt					x	x	

fdsqrt							x	x	
ftan							2	2	x
ftanh							2	2	x
ftentox							2	2	x
ftrap							x	2	x
ftst							x	x	x
ftwotox							2	2	x
illegal	unsized		x	x	x	x	x	x	
jmp	unsized		x	x	x	x	x	x	
jsr	unsized		x	x	x	x	x	x	
lea	l		x	x	x	x	x	x	
link	w,l		x	x	x	x	x	x	
lpstop								x	
lsl	b,w,l		x	x	x	x	x	x	
lsr	b,w,l		x	x	x	x	x	x	
move	b,w,l		x	x	x	x	x	x	
movea	w,l		x	x	x	x	x	x	
moveq	l		x	x	x	x	x	x	
movec\$^1\$	l			x	x	x	x	x	
movem	w,l		x	x	x	x	x	x	
movep	w,l		x	x	x	x	x		
moves\$^1\$	b,w,l			x	x	x	x	x	
move16							x	x	
muls	w,l		x	x	x	x	x	x	
mulu	w,l		x	x	x	x	x	x	
nbcd	b		x	x	x	x	x	x	
neg	b,w,l		x	x	x	x	x	x	
negx	b,w,l		x	x	x	x	x	x	
nop	unsized		x	x	x	x	x	x	
not	b,w,l		x	x	x	x	x	x	
or	b,w,l		x	x	x	x	x	x	
ori	b,w,l		x	x	x	x	x	x	
pack	unsized				x	x	x	x	
pea	l		x	x	x	x	x	x	
pflush\$^1\$	unsized					x	x	x	
pflusha\$^1\$	unsized					x			
plpa\$^1\$	unsized								x
pload\$^1\$	unsized					x			
pmove\$^1\$	w,l,q					x			
ptest\$^1\$	unsized					x	x		
reset\$^1\$	unsized		x	x	x	x	x	x	
rol	b,w,l		x	x	x	x	x	x	
ror	b,w,l		x	x	x	x	x	x	
roxl	b,w,l		x	x	x	x	x	x	
roxr	b,w,l		x	x	x	x	x	x	
rtd	unsized			x	x	x	x	x	
rte\$^1\$	unsized		x	x	x	x	x	x	
rtr	unsized		x	x	x	x	x	x	
rts	unsized		x	x	x	x	x	x	
rtm	unsized				x				
sbcd	b		x	x	x	x	x	x	
scc	b		x	x	x	x	x	x	
stop\$^1\$	unsized		x	x	x	x	x	x	
sub	b,w,l		x	x	x	x	x	x	
subq	b,w,l		x	x	x	x	x	x	
suba	w,l		x	x	x	x	x	x	
subi	b,w,l		x	x	x	x	x	x	

subx	b,w,l	x	x	x	x	x	x	
swap	w	x	x	x	x	x	x	
tas	b	x	x	x	x	x	x	
trap	unsized	x	x	x	x	x	x	
trapcc	? ,w,l			x	x	x	x	
trapv	unsized	x	x	x	x	x	x	
tst	b,w,l	x	x	x	x	x	x	
unlk	unsized	x	x	x	x	x	x	
unpk	unsized			x	x	x	x	
+-----+-----+-----+-----+-----+-----+-----+-----+								

\$^1\$ Supervisor instruction
2 These are software-supported instructions on the 68040 and 68060