DLDBTT

COLLABORATORS			
	TITLE : DLDBTT		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 4, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

DLDBTT

1.1 Dreamline Designs' Bugs, Tips and Tricks guide.

```
                                   Dreamline Designs presents
      The bugs, tips and tricks guide v1.2

                Released 21-07-95.


Introduction
  What is this?

News
  News subjects/changes since v1.1

Disclaimer

Contacting us
  How to reach us.

Missing info
  List of missing information.

Authors
  List of the authors of this document.

LCR-Project
  Info on the Library-Call-Registers project.

Bugs reports.

Autodocs
  Bugs in autodocs.

Includes
  Bugs in includes.

Libraries
  Bugs in libraries.

RKM
```

1.2 Introduction to this document.

This Document has it's roots in Dreamline Designs internal NewsLetter. Here we, among other things, report found errors in the Autodocs, the includes, the ROM Kernel Manuals and other books. I have decided that we should share this information with other developers. Furthermore this document contains some tips and tricks.

If You have knowledge of any bugs we haven't discovered, feel free to

contact
us.

If anyone should have any public information on electronical media about some of the blank areas such as the mfm.device or the dos.library please mail them to me and I'll include them in the next version of this document. See the
list
for a more specific list of undocumented areas.

From v1.2 the archive comes with the LCR-Project. (LCR=Library-Call-Registers) This project is a set of includes which can be used by assembler-coders when using librarycalls. See more info in the
LCR-Project
section.

1.3 Updates since v1.1

-Rewrote the Tips for the dos.library/Info()

-Reported some bugs in The Amiga Guru Book, seems like the biggest problem is the index. :)

-A couple of bugs found in the Autodocs.

-A couple of bugs found in the RKM: Libraries.

-Added the LCR-Project to the archive.
 This is a help for assembler-programmers when calling library-functions.
 Read more in the docs for the project.

-Rewrote RKM: Devices/Wrong labels a bit.

-Corrected a couple of bugs in this guide, sorry guys. :)

1.4 List of undocumented areas of the AmigaOS.

This is a list of undocumented areas of the AmigaOS. If you have ANY public information stored on electronic media from the list below, please mail them to me so I can include this in the next version of this document.

 -Devices: Includes, autodocs and general information on the mfm.device and the ramdrive.device.

 -dos.library: General information on the dos.library. The dos.library is completely left out in the RKMs Third Edition.

C/PASCAL/E/OBERON/AMOS/BASIC-PROGRAMMERS WANTED FOR WRITING!!!
This document is mostly for assembler-coders, we need some other coders to report bugs in their books.

Please share your knownledge with the rest of the developers.

1.5 Disclaimer

Dreamline Designs cannot in any way be held responsible for any problems of any kind the use of this information can cause or has caused.

You may freely distribute this document. If you have any kind of comments, suggestions or new bugs please report them to us instead of just add them and spread the modified document.

1.6 How to reach Dreamline Designs.

You can reach us at the following places:

 Stargate, Dreamline Designs BBS

 Address

1.7 Stargate, Dreamline Designs BBS.

NOTE: From the 8/8/95 Stargate will be online 24h/day.

1.8 The homeaddress of this document.

Rasmus K. Ursem 2:238/28.10 (FidoNet)
10:2/9.10 (HOANet)
39:140/109.10 (AmyNet)

Snail-Mail

Rasmus K. Ursem, Dreamline Designs.
Gudrunsvej 8 3. TH
8220 Brabrand
Denmark
+45 86755309 (Voice)

1.9 The authors of this document.

The following people has reported bugs or written parts of this document.

Rasmus K. Ursem, Dreamline Designs.
Karsten T. Niemeier, Dreamline Designs.
Jakob V. Hansen, Dreamline Designs.
Dennis Franck.

1.10 The Library-Call-Registers project from Dreamline Designs

Info on the LCR-project (Library-Call-Registers) from Dreamline Designs.
Idea and concept by Rasmus K. Ursem, Dreamline Designs.

Introduction
Introduction to the LCR-Project.

Copyrights
Copyrights, disclaimer and distribution.

Current Status
Current status of the project.

Using the includes
How to use the includes.

Future
Future development of the LCR-Project.

1.11 Introduction to the LCR-Project.

INTRODUCTION
This is a set of includes used by assembler-programmers when accessing
libraries. The include contains all register referring for all the
libraryoffsets. An Example:

```
*** _LVOAllocMem() ***
_LVOAllocMem_Size       equr d0
_LVOAllocMem_ByteSize   equr d0
_LVOAllocMem_Attributes equr d1
_LVOAllocMem_Att        equr d1
_LVOAllocMem_Requirements equr d1
_LVOAllocMem_Result     equr d0
_LVOAllocMem_RFail      = 0
_LVOAllocMem_RU         = 2
```

What can this be used to? Well, the 2 major advantages.

-You don't have to open your autodocs every time you want to get the

```
 registers used for a library-call or the fail-code.

-Your code will be easier to read and understand.
```

1.12 Copyrights, disclaimer and distribution.

```
COPYRIGHT
The LCR-Project is copyrighted ©1995 by Rasmus K. Ursem, Dreamline Designs.
All rights reserved. It is distributed with the Bugs, Tips'n'Tricks guide
from Dreamline Designs.

DISCLAIMER
Dreamline Designs cannot in any way be held responsible for any errors caused
by using these includes. However, they have been tested and should be ok.

DISTRIBUTION
You may freely spread the archive on any media, put on coverdisks or in PD-
libraries as long as nothing is changed.
The LCR-Project comes with the tips'n'tricks guide from Dreamline Designs,
and should be distributed with this.
```

1.13 Current Status of the project.

```
CURRENT STATUS
LCR_Exec.i      100% done
LCR_Dos.i       Not complete but the most used functions is supported.
```

1.14 How to use the includes.

```
USING THE INCLUDES

You simply include them as all other includes.

        incdir  includes:
        include LCR/LCR_Exec.i

Say you want to allocate some memory, without using LCR_Exec.i you have to

        ...
        move.l  #1000,          d0
        move.l  #MEMF_PUBLIC,   d1
        move.l  4,              a6
        jsr     _LVOAllocMem(a6)
        ...

Well it is shorter than

        ...
        move.l  #1000,          _LVOAllocMem_Size
        move.l  #MEMF_PUBLIC,   _LVOAllocMem_Attributes
```

```
        move.l  4,              a6
        jsr     _LVOAllocMem(a6)
        ...
```

But you don't have to check your autodocs for the registers and the source is, IMHO, easier to understand.

1.15 The future of the LCR-Project.

FUTURE
The LCR-project will be updated along with new versions of the Bugs, tips'n'tricks guide. The Next version will contain a 100% done LCR for the dos.library.

If you want to help us with this project you are VERY welcome. (it's kinda boring converting an autodoc to a LCR-include.)
You don't have to be a coder to help us, anyone who can use a texteditor can do it.

If you are coding libraries other coders can use you can support us a lot by sending us the LCR for your library. We'll of course include it in the next version of the guide.

1.16 Known bugs in autodocs.

Known bugs in the Autodocs.

General bugs
General bugs in the autodocs from Commodore.

System-libraries.

exec.library

dos.library

mathffp.library

mathtrans.library
User-libraries.

xpkmaster.library

1.17 General bugs in the autodocs.

General bugs which can be found in many of the autodocs.

```
                Missing registers
                 Missing registers under the SYNOPSIS-line.
```

1.18 Missing registers in the autodocs.

```
Some of the autodocs from CBM miss some registers below the SYNOPSIS-line.
Especially the Result register is missing. This is probably because most
of the results is recieved in D0.

Example of the missing result-register.

mathtrans.library/SPAcos()

SYNOPSIS
                fnum2 = SPAcos(fnum1);
                               d0.l
                  ^
                   \_ d0 is missing.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.19 Bugs in the autodocs for the exec.library

```
                Known bugs in the Autodocs for the exec.library.


                 AddTask()

                 CreateIORequest()

                 ObtainQuickVector()
```

1.20 Missing result register.

```
Here is missing the mentioned result-register d0. The line should look like
this:

SYNOPSIS
        ioReq = CreateIORequest( ioReplyPort, size );
        d0                       a0           d0

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.21 Missing result register.

```
Here is missing the mentioned result-register d0. The line should look like
this:

SYNOPSIS
        AddTask(task, initialPC, finalPC)
        d0      A1    A2         A3


                                        Rasmus K. Ursem, Dreamline Designs.
```

1.22 ObtainQuickVector()

```
First of all, some versions of the autodocs dosn't have this entry.
The bug is not really a bug, but more a matter of standards in the autodocs.
If your autodocs have this entry, the NAME section probably looks like this:

NAME
     Function to obtain an install a Quick Interrupt vector            (V39)

According to the standards of the autodocs the entry should look like this.

NAME
     ObtainQuickVector -- Function to obtain and install a Quick Interrupt vector. ↩
         (V39)

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.23 Bugs in the autodocs for the dos.library

```
                Known bugs in the Autodocs for the dos.library.


                 WriteChars()

                 GetPacket()

                 QueuePacket()
```

1.24 Dos.library/WriteChars

```
Here is missing a D2 under the buflen.

SYNOPSIS
        count = WriteChars(buf, buflen)
        D0                 D1    D2

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.25 Dos.library/GetPacket()

```
This node is completely left out. Probably because the function is replaced
by other dos functions. You should use WaitPkt() instead.
Information can be found in the Amiga System Programmers Guide at page 347.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.26 Dos.library/QueuePacket()

```
This node is completely left out. Probably because the function is replaced
by other dos functions. You should use SendPkt() instead.
Information can be found in the Amiga System Programmers Guide at page 347.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.27 Bugs in the autodocs for the mathffp.library

```
                Known bugs in the Autodocs for the mathffp.library.


                 SPTst()

                 SPDiv()
```

1.28 SPTst

```
RESULT

   ....

Integer functional result as:

        +1 => fnum > 0.0
        -1 => fnum < 0.0
         0 => fnum = 0.0

This isn't correct. SPTst() will never return -1. As the normal integer tst
it only tests the operand against Zero. The correct output should be like this:

I recently found this in another book and decided to check up on the subject,
unfortunately I am wrong, SPTst does really return -1 if the value is negative.
Sorry about that.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.29 SPDiv

```
Here is missing some very important information. The authors forgot to
specify which number was devided by which.

SPDiv -- Divide two floating point numbers.

....

SYNOPSIS
        fnum3 = SPDiv(fnum1, fnum2)
        D0            D1     D0

The missing part:

fnum3 = fnum2/fnum1

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.30 Bugs in the autodocs for the mathtrans.library

```
Known bugs in the Autodocs for the mathtrans.library.

In all the node the result register is missing. However I assume that the
results can be recived in D0.

                                        Rasmus K. Ursem, Dreamline Designs..
```

1.31 Bugs in the autodocs for the xpkmaster.library

```
                Bugs found in the autodocs for xpkmaster.library.


                 XpkExamine()
```

1.32 XpkExamine()

```
This can be found in the autodocs for the xpkmaster.library/XpkExamine()

        ....

INPUT
        tags    - Pointer to an array of struct TagItem. You may use
                  either a XPK_InBuf, a XPK_InName, XPK_InFH or XPK_InHook
                  tag.


The author left out the fib input. The entry should look like this.
```

```
INPUT
        fib     - Pointer to a XpkFib structure.
        tags    - Pointer to an array of struct TagItem. You may use
                  either a XPK_InBuf, a XPK_InName, XPK_InFH or XPK_InHook
                  tag.

                                Rasmus K. Ursem, Dreamline Designs.
```

1.33 Bugs in the includes.

```
                General errors in the includes.

                 Libraryoffsets
                Specific errors in the includes.

                 Libraryoffsets/dos_lib
```

1.34 Bugs in the libraryoffsets for dos.library.

```
Because the GetPacket() and QueuePacket() is no longer supported these are
left out.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.35 Huge problem in the libraryoffsets

```
In most of the includes for libraryoffsets for assembler is missing the
lines that prevents double assembling of includes.

E.g from Dos/dos.i these lines are at the top.

IFND DOS_DOS_I
DOS_DOS_I SET 1

... rest of include ...

and 'ENDC' at the last line.

When including a libraryoffset-include twice the assembler will resond
with the error "Double Symbol". To fix your includes you have to add
some of these lines yourself. We suggest the following names.

LVO_<Libraryname>_I

For dos.library these lines will be as follows:

IFND LVO_DOS_I
LVO_DOS_I SET 1
```

```
... rest of include ...

ENDC

                    Rasmus K. Ursem and Karsten Niemeier, Dreamline Designs.
```

1.36 Bugs in the libraries.

```
No non-mentioned bugs in the libraries from CBM is found.
```

1.37 Bugs in the RKMs.

```
                Bugs and missing parts of the RKMs.


                 RKM Libraries
                 Third Edition.

                 RKM Devices
                 Third Edition.
```

1.38 Bugs in the RKM Libraries Third Edition

```
                 What about the dos.library?
                Page number


                 401

                 500
```

1.39 What about the dos.library?

```
CBMs "Amiga ROM Kernel Reference Manual: Libraries" has a huge bug.
Somehow the authors of this book completely forgot the dos.library.

This is very annoying because neither the autodocs or the includes say much
about e.g. the packets.

Anyway, A friend of mine told me that Commodore has "leased out" writing
about dos.library to another company, AFAIR called Bantam.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.40 Gadtools.library

Just below the subtitle "GADGET REFRESH FUNCTIONS" there is 3 lines. The last sentence is:
"Alternately, they may be added to a window after it is open by using the functions AddGList() and RefreshGList()."

Using AddGList() will, in this connection, crash the computer.

Jakob V. Hansen, Dreamline Designs.

1.41 Exec messages and ports.

At the top of page 500 this C-structure is placed.

```
struct          Node mp_Node;
UBYTE           mp_Flags;
UBYTE           mp_SigBit;
struct task     *mp_SigTask;
struct          List mp_MsgList;
```

According to the includes (exec/ports.h and in asm, exec/ports.i) the currect structure should look like this:

```
struct          Node mp_Node;
UBYTE           mp_Flags;
UBYTE           mp_SigBit;
void            *mp_SigTask;
struct          List mp_MsgList;
```

Rasmus K. Ursem, Dreamline Designs.

1.42 Bugs in the RKM Devices Third Edition

	Page number
Missing devices	
Wrong labels	
	310
	311

1.43 A couple of missing devices.

CBMs "Amiga ROM Kernel Reference Manual: Devices" has left out at least 2 more or less important devices. This is the mfm.device used by PCx: and

and the ramdrive.device used by RAD: However in general you can say that
the mfm.device works allmost the same way as the trackdisk.device, but
no function is documented, and AFAIK no includes or autodocs exists.
The ramdrive.device used by RAD: isn't documented either, however such
IO-commands as TD_GETNUMTRACKS or any non-standard device IO commands cannot
be used.

Rasmus K. Ursem, Dreamline Designs.

1.44 Case sensitive labels.

Though the book the authors uses labels with cases that dosn't match the
case of some (my) version of the includes. example:

```
In the book:            io_Data
In the includes:        IO_DATA
```

Case sensitive assemblers can of course not assemble these labels.
(I haven't checked this for the C-includes.)

NOTE: This should only be important to assembler programmers because
the book is written with C in mind, and the labels in the book is
"C-Labels".

Rasmus K. Ursem, Dreamline Designs.

1.45 Controlling the drive motor.

Just below the middle of this page just above the example is a sub-chapter
starting with this:

You control the drive motor by passing an IOExtTD to the device with CMD_MOTOR
or ETD_MOTOR.

The bug:

CMD_MOTOR does not exsist this should be replaced with TD_MOTOR.

Rasmus K. Ursem, Dreamline Designs.

1.46 Formatting a track.

At the middle of this page just above the example is a sub-chapter
starting with this:

You format a track by passing an IOExtTD to the device with CMD_FORMAT
or ETD_FORMAT.

The bug:

CMD_FORMAT does not exsist this should be replaced with TD_FORMAT.

Rasmus K. Ursem, Dreamline Designs.

1.47 Bugs in other books.

Amiga System Programmers Guide
©Data-Becker, 1988.

Amiga ordbogen
©Sall Data, 199?.

Commodore Amiga Maskinsprog
©Data-Becker, 1987.

The Amiga Guru Book (English)
©Ralph Babel, 1993.

Mastering Amiga Assembler
©Paul Overaa, 1992.

1.48 Amiga system programmers guide

Page number

212
Introduction to programming the amiga.

244
Task functions.

266
The Allocmem() and FreeMem() functions.

339
Output().

342
Examine().

344
Info().

347
QueuePacket().

353
Disk layout.

1.49 Introduction to programming the amiga.

```
At approx line 7 this code is listed:

        Lea.l   mytask,a0
        move.l  #01,8(a0)               ;set type = task

This is wrong because the type in the Nodestructure only leaves space for
a byte. The lines should be changed to:

        Lea.l   mytask,a0
        move.b  #01,8(a0)               ;set type = task

The same mistake is made at approx the middle of the page.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.50 Task functions.

```
At the AddTask() the registers below is like this:

   Addtask(task,initialPC,finalPC)
            A0      A1      A2

This should be corrected to:

   Addtask(task,initialPC,finalPC)
            A1      A2      A3

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.51 The Allocmem() and FreeMem() functions.

```
Here is missing a -198 just below the "AllocMem"

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.52 Output()

```
Here is missing a d0 and the libraryoffsets below the Synopsis. The Output should
look like this.

 Handle = Output()
 d0        -60

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.53 Examine()

Vital information about the alignment of the infoblock is missing here.
The infoblock MUST be longword-aligned. (Starting at an address divideable
by 4.)

Rasmus K. Ursem, Dreamline Designs.

1.54 Info()

The libraryoffset just below "Info" should be corrected from -104 to -114.

Rasmus K. Ursem, Dreamline Designs.

1.55 QueuePacket()

Here is missing a d0, a d1 and the libraryoffsets below the Synopsis. The
QueuePacket should look like this.

```
 Status = QueuePacket(Packet)
 d0        -168         d1
```

Rasmus K. Ursem, Dreamline Designs.

1.56 Disk Layout.

This part of the book contains a lot of useful info about disk layout,
however at the beginning of all the figures this line is written.

Word Name Contents Description

This should be corrected for all the figures at the following pages to this.

LongWord Name Contents Description

Just above the figure for the Root-block the authors say that all values
are longwords. One should keep in mind that only the numbers below the "Word"
column should be mulitplied by 4.

Rasmus K. Ursem, Dreamline Designs.

1.57 Amiga ordbogen.

This book written in danish is simply a bug. It is supposed to be a dictionary
for the Amiga, but most of the information in the book is simply misguiding
and useless. The book is more useful as a gag-book.

Rasmus K. Ursem, Dreamline Designs.

1.58 Commodore Amiga Maskinsprog Dansk/norsk udgave.

```
 General errors in the book
Page Number


  69

  95

 118

 119

 129

 136
```

1.59 General errors in this book.

```
-The author has forgotten a tailing : on some of the labels. AFAIR the
 Seka-assembler (and maybe other assemblers) cannot handle labels
 without :
-The assembler directive called "even" is not placed at the proper position
 in the syntax. Most assemblers today define labels as the first chars of
 a line, and if this is space or tab there is no label here. In the book
 the "even" is written at the first position of a source line. Assemblers
 will not assemble these sources.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.60 Hexlong conversion.

```
Just after the Loop: label at the

        rol     #4, d1

This should be longword like

        rol.l   #4, d1

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.61 Dosname.

```
Just after the label dosname is this line
```

```
        dc.b    'dos.library',0,0,even
```

because even is a socalled assembler directive it probably will cause trouble when assembling is attempted. This should be changed to:

```
        dc.b    'dos.library',0,0
        even
```

The same 'error' is at page 98.

Rasmus K. Ursem, Dreamline Designs.

1.62 Felt:

At the lower part of the page is a label called felt: This line looks like this in the book.

```
felt:   blk.b                   ;reserver 100 bytes
```

The line should look like this

```
felt:   blk.b   100,0           ;reserver 100 bytes
```

Rasmus K. Ursem, Dreamline Designs.

1.63 link

At the upper part of this page there is a routine with a label called readdata: The code looks like this.

```
        move.l  dosbase,a6
        move.l  filehd,d1
        move.l  #$ffffff,d3             ;Læs et vilkårligt antal bytes
        jsr     Read(a6)                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        rts                                     \/
                                                /
```

This is saying "Read any amount of bytes". However the routine "only" reads 16.777.215 bytes (Max). The line should look like this:

```
        move.l  #$ffffffff,d3           ;Læs et vilkårligt antal bytes
```

Which specifies the maximum amount of data the Read() can read.

Rasmus K. Ursem, Dreamline Designs.

1.64 Calculating the offsets.

At the upper part of the page the following is written.

```
offset=(Sektornummer-1)*512
```

```
This is not correct. Normally we define the first sector on a disk as sector
0, using the above formula will set our offset equal -512 which is wrong
for the first sector (0). The line should look like this

offset=Sektornummer*512

And of course assume the first sector on the disk to be sector 0.

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.65 Detail-pen.

```
In this structure is written

detail_pen:     dc.w    0

this entry is one byte-size so it should be replaced by

detail_pen:     dc.b    0

                                        Rasmus K. Ursem, Dreamline Designs.
```

1.66 The Amiga Guru Book.

```
                This book is a MUST for the utility-coder who want to use the dos. ↩
                   library.
The book contain, among other things, vital info on most of the packets.
(No 3.0+ packets, but that almost dosn't matter since there is only a couple
of them. (ACTION_EXAMINE_ALL_END and ACTION_SET_OWNER))

                                        Rasmus K. Ursem, Dreamline Designs.


                 Chapter 17.1 The dos.library
                Page Number


                 705

                 710

                 711
```

1.67 Chapter 17.1 The dos.library.

```
In this chapter on some of the pages is the result sometimes described as

   BOOL
```

```
   D0
     [0...15]
```

I have been doing some testing on this subject and found out that this
is not always true. (Testing is done on v39.23 = KS3.0) I have done the
testing on the AssignLock() and AssignPath() and they both return a
longword success/failure. However is is only relevant for assembler-freaks
because C probably takes care of this.

Rasmus K. Ursem, Dreamline Designs.

1.68 Leftout in the index.

Well this is not excatly a bug, but more an annoying "left out".

In the index the author has left out the ACTION_DISK_CHANGE packet.
The reason is, AFAIK, because ACTION_DISK_CHANGE is an internal packet,
and not used to check for disk-changes. Anyway you can add the following
line in your index.

ACTION_DISK_CHANGE page 630.

Here is also descibed other internal packets.

Rasmus K. Ursem, Dreamline Designs.

1.69 Error in the index.

The Entry "checksum 355 f., 369, 550" should be "checksum 356 f., 369, 550".
The subchapter about calculating the checksum is placed on page 356 not 355.

Rasmus K. Ursem, Dreamline Designs.

1.70 Error in the index.

The entry "ComHandler.c, 629 f., 633" has a minor bug. The Source is placed
at page 643, not 633.

Rasmus K. Ursem, Dreamline Designs.

1.71 Mastering Amiga Assembler.

I havn't used this book much but a quick browse though the book revealed
a couple of bugs.

Rasmus K. Ursem, Dreamline Designs.

Page number

56

62

64

65

77

78

109

1.72 Solving simple problems.

Quick instructions:

At the top of the page is written that moveq is an 16-bit instuction that sign-extends the contens to a longword. This is written very tricky, to cause total confusion this could be replaced by something like this.

The moveq command always affects the data-registers in 32-bit/longword.

It is in other words not possible to do a

```
moveq.w #10,   d0
```

At the middle of page these two lines is written

```
move.l  #RESULT,A1              Load a1 with address of RESULT
addq    #4,(A1)                 Add 4 to the contents of the byte
                                                          ^^^^
                                This is wrong, should be "word"
                                because the default size of addq
                                is word, not byte.
```

The same mistake is made at the next example.

The whole subchapter leaves the reader with the impression that quick instruction only can be used on words, which is wrong.

Rasmus K. Ursem, Dreamline Designs.

1.73 Solving simple problems.

At the top page in the example the line after the LOOP label.

```
        addq.1  #1,     a0
             ^
        This should be the letter l
```

The same error is in the next example and at page 63.

Rasmus K. Ursem, Dreamline Designs.

1.74 Solving simple problems.

At the second example (CH3-18.s) this line is after the LOOP-label.

```
        subq.b  d0
```

Should be

```
        subq.b  #1,     d0
```

Rasmus K. Ursem, Dreamline Designs.

1.75 Solving simple problems.

Example CH3-19.s the 3 first lines look like this.

```
        lea     TEXT,   a0      Put address f string in a0
        move.b  (a0)+,  d0      Copy count and increment pointer
        sub.b   #1,     d0      reduce count by 1 for dbra
```

One should keep in mind that Dbra works with 16 bits, and if d0 the lower middle byte isn't 0 (e.g $ff00) dbra will loop more than the requested. The example should look like this

```
        lea     TEXT,   a0      Put address f string in a0
        moveq   #0,     d0      Clear d0 for dbra use.
        move.b  (a0)+,  d0      Copy count and increment pointer
        sub.b   #1,     d0      reduce count by 1 for dbra
```

The same error is in example CH3-20.s at page 66.

Rasmus K. Ursem, Dreamline Designs.

At least one bug pr page! Hmmmm, the subchapter sure looks like fast work or a sleepy author. :)

1.76 Subroutines and parameter passing.

```
Below the middle of this page is 3 boxes the first and the last contains
this line

a7 a6 a5 a4 a3 a2 a1 a0 d7 d6 d5 d4 d3 d2d d1 d0
                                       ^^^
                                       This is of course wrong. should be d2

                                       Rasmus K. Ursem, Dreamline Designs.
```

1.77 Subroutines and parameter passing.

```
Just below the middle of this page is these lines

        movem.l d0-d7/a0-a3,    -(sp)

This line is ok.

        move.l  (sp)+,          d0-d7/a0-a3

Here is missing a m in the movem.l instruction, the line should look like
this.

        movem.l (sp)+,          d0-d7/a0-a3

The same error is in some of the lines at this page and the next. All lines
with a (sp) must be movem.l and not move.l. (NOTE: Only on page 78 and 79)

                                       Rasmus K. Ursem, Dreamline Designs.
```

1.78 Program design issues.

```
At the top of this page is the following code listed

        move.l  #INDIRECTION_TABLE,a5
        asl.w   #2,d0                   ;Multiply by 4
        move.l  (a5,d0.l),a0
        jsr     (a0)

        asl.w   #2,d0                   ;Multiply by 4
        move.l  (a5,d0.l),a0

These 2 lines can result in some very spooky bugs, say the highword of d0
contains anything else but 0, the last line will get a totally wrong address.
What if d0 contains anything larger than $4000? Wierd and wrong addresses
occur. The asl.w should be a

        lsl.l   #2,     d0

or at least a

        asl.l   #2,     d0
```

Rasmus K. Ursem, Dreamline Designs.

1.79 Tips for Assembler-programmers.

Some Tips for the Assembler-programmer.

Forgetting to cmp
Optimizing Assembler. (This section is for the demo-coder.)

Faster Anding

Memory accesses

Using tables

Faster Multiplication

Faster Floating points

Saving registers

Some words when using Ax's

1.80 Forgetting to cmp before address-registers

The MC680x0 has a tricky thingy when leaving out a cmp.l #0,Ax instruction
in connection with Address-registers. The tricky thing is:

When working on Address-registers (Ax) the processor dosn't set any flags,
however this is only when using Ax in the following way.

```
        move.l  Testarea,       a0
```

This dosn't set the Zeroflag if Testarea contains 0. Example:

```
        move.l  DosBase,        a1
        beq     .NoDosLibOpen           ;This instruction will never jump
                                        ;to .NoDosLibOpen because the zero-
                                        ;flag isn't set if DosBase == 0
                                        ;a cmpi.l #0,a1 before the beq is
                                        ;needed.
        move.l  4,              a6
        jsr     _LVOCloseLibrary(a6)

.NoDosLibOpen
```

Rasmus K. Ursem, Dreamline Designs.

Why addressregisters doesn't affect the condition flags:

```
The reason is very simple. The Motorola Enginneers' actually knew
what they were doing. To ensure interrupt-transparency, the pro-
cessor has to save all the conditionflags, the programcounter and
the data\address registers on the stack, when an interrupt occurs.
When an interrupt occurs, the first thing that happens is :
  1) A word containing the condition flags is pushed onto the stack
  2) The programcounter (PC) is pushed on the stack as a longword
If pushing the anything anything onto the stack would set the
conditionregisters, then you couldn't relay on them, since the status
of these registers would be dependent on what you actually pushed
onto the stack! Imagine this :

You have a subroutine, where you are executing this instruction :
      cmp.l #0,d0
Now an interrupt occurs. You don't know whats happening in this
interrupt, and honestly, you don't care!
Now the interrupt is finished, and gives the CPU Power back to you
subroutine, which executes this command :
      beq.s regd0positive
This command depends on the status of one of the condition flags!!!
So the condition flags better have to be the same, as before the
interrupt occured! Otherwise, you're fucked!!! Perhaps the Motorola
engenieers could have let a7 be the only register not to affect
the condition registers, but this would cause a lousier performance
of the CPU.
I sure hope this is the solution to all your confusion...

                                        Karsten Niemeier, Dreamline Designs.
```

1.81 A way to do faster anding.

```
Here is a little hint that can be used when anding small values.

Say we want to copy a dataregister to another dataregister and AND this
with a value below $7f. The usual way to do this is:

        move.l  d0,     d1
        And.l   #$7f,   d1

This can be done like this instead.

        moveq   #$7f,   d1
        and.l   d0,     d1

This is faster and the upcode is 2 words smaller. (1 word if we are using
words.)

                                        Karsten Niemeier, Dreamline Designs.
```

1.82 Memory accesses.

A good thing to keep in mind when coding fast assembler-routines is to
limit the amount of memory accesses to an absolute minimum. Memory accesses,
and especially Chip-memory accesses is extremely slow compared to accessing
the data-registers of the MC680x0.

An example: We are coding something that has to set a lot of pixels
continuous though the chip-memory, this could e.g. be a CPU-Filled-Vector.

Unoptimized way:

```
        ...
        moveq.l #31,      d6      ;Counter for Dbra
        lea     WritePos, a0      ;Somewhere in Chip-mem.
.loop:                            ;Loop that writes 32 continuous pixels
        Bset    d6,       (a0)    ;Set a pixel in the screen.
        dbra    d6,       .loop
        ...
```

This routine accesses the chip-memory 32 times. (One for each pixel set)
It will be approx 32 times (on a 68020+) slower than the following routine
because chip-mem accesses eats a lot of time.

```
        ...
        moveq.l #31,      d6      ;Counter for Dbra
        moveq.l #0,       d0
.loop:                            ;Loop that writes 32 continuous pixels to d0
        Bset    d6,       d0      ;Set a pixel in d0.
        dbra    d6,       .loop

        lea     WritePos, a0      ;Somewhere in Chip-mem.
        move.l  d0,       (a0)
        ...
```

This routine only accesses the chip-memory once. Approx 32 times faster than
the routine above. (On a mc68020+) The pixelsetting routine will be loaded
to the cache and will thus take almost no time.

The example above is of course very simple and, in this case, it would be
much faster just to move.l $ffffffff directly to the memory.

Rasmus K. Ursem, Dreamline Designs.

1.83 Precalculated tables.

Precalculated tables is often used to speed up routines. The main purpose is
to avoid complicated calculation when the routine is running.

An example: We are coding something which requires a mulu with 320.

Unoptimized:

```
.Loop:
        ...                     ;A nice loop that does a lot of cool stuff.
```

```
        mulu.w  #320,d1         ;Here is a slow command this might be
                                ;replaced by a table
```

```
 Optimized:
  If you know anything about the limits of d0 and d1, you could make an
  precalculated multiplication table:

   lea.l        Mulu320Table,a0
   move.w       (a0,d1.w*2),d1

  Also see the special
               multiplication
                section!

                                        Karsten Niemeier, Dreamline Designs.
```

2. Example: We are coding a routine that sets pixel on the screen. A sinus affects the y-coord, which means that we have to multiply the sinus with the width of the screen.

```
Unoptimized table
.SinusTable     dc.w    0,0,0,1,1,2,3,3,4,4,4,3,3,2,1,1
```

When you are using this you have to do a

```
        ...
        mulu    #40,    d0
        ...
```

before you can set the pixel. Instead the table could just look like this.

```
.SinusTable     dc.w    0*40,0*40,0*40,1*40,1*40,2*40,3*40,3*40,4*40,4*40,4*40
                dc.w    3*40,3*40,2*40,1*40,1*40
```

and you don't have to do the

```
         mulu #40,      d0
```

Rasmus K. Ursem, Dreamline Designs.

1.84 Multiplication.

```
 NOTE: On the mc68020+, shifting 1 or 8 times takes exactly the same time,
       because the CPU has a shifter, just as the blitter has!

 Generally all multiplications can be written as a combination of some
 arithmetic operators. Sometimes it's slower to write the multiplication
 as a combination, than using the mulu command. In these cases, you might
 use a precalculated table, as stated in section 1, example 2.
 Consider these examples:

 Example 1, Unoptimized:

  mulu.w  #256,d0
  muls.w  #256,d1
  mulu.w  #1024,d2
```

```
 muls.w  #1024,d3

Example 1, Optimized:
 lsl.w   #8,d0
 asl.w   #8,d1

 moveq.l #10,d4
 lsl.w   d4,d2
 asl.w   d4,d3

Example 2, Unoptimized:
 mulu.w  #320,d0

Example 2, Optimized:
 move.w  d0,d1
 lsl.w   #6,d0
 lsl.w   #8,d1
 add.w   d1,d0

 The bitcombination of #320 is %101000000, thats why we shift 6 and 8
 times.

                                        Karsten Niemeier, Dreamline Designs.
```

1.85 Fast self-invented floating points.

```
Floating point:

 Many people uses floating point in their routines. Also many people
 don't know how to make floating point operations on the processor. The
 following example is just my way of making it. It uses 16 bit precision,
 but still some people prefer to use fx. 7 bit precision, since it's a
 little faster! This example only works on mc68020+, since it uses divs.l!!!

 As an example, we could make a linedraw routine:

 LineDraw:
 PRE:  X1Coordinate in d0
       Y1Coordinate in d1
       X2Coordinate in d2
       Y2Coordinate in d3
 NOTE: The SetDot routine should of course be implemented in the LineDraw
       routine, but I placed it as a Routine, to make the source easier to
       read.

        cmp.w   #319,d0   ; First, check if any of the coordinates
        bhi     .out      ; exceeds the screen. If any of the coordinates
        cmp.w   #319,d2   ; does, the we just skip the whole routine.
        bhi     .out
        cmp.w   #255,d3
        bhi     .out
        cmp.w   #255,d4
        bhi     .out

  sub.w   d0,d2     ; Calc delta x
```

```
  sub.w   d1,d3     ; Calc delta y
  cmp.w   d2,d3
  bge     .d3g
  move.w  d2,d7     ; # Dots to draw!
  bra     .cont1
.d3g: move.w  d3,d7     ; # Dots to draw!
.cont1: tst.w   d7
        beq     .out      ; If delta is zero, then skip!
        ext.l   d7
  swap    d2
  swap    d3
  clr.w   d2
  clr.w   d3
  divs.l  d7,d2     ; delta x float
  divs.l  d7,d3     ; delta y float

  and.l   #$ffff,d0 ; Clear float part of d0 (xcoord)
  and.l   #$ffff,d1 ; Clear float part of d1 (ycoord)

.loop:  jsr SetDot    ; SetDot(d0,d1)
  swap    d0
  swap    d1
  add.l   d2,d0
  add.l   d3,d1
  swap    d0
  swap    d1
  dbra    d7,.loop
.out:   rts

SetDot:
PRE:  XCoordinate in d0
      YCoordinate in d1

      movem.l   d0-d2/a0,-(sp)
      lea.l     Screen,a0
      move.b  d0,d2
      not.b     d2
      lsr.w     #3,d0
      mulu.w    #320,d1
      add.w     d0,d1
      bset  d2,(a0,d1)
      movem.l   (sp)+,d0-d2/a0
      rts
```

Comments:
None of this code is optimized, nor assembled! Well, let me explain the theory behind these floating point operations: We use a longword for the floatingpoint number. The lowword is the point part of the number, and the hiword is the number. Fx. the number 1 would look like this : $00010000. Fx. the number 1.5 would looke like this : $00018000 etc. To calculate the delta we first scale our input by 2^16, and so we does by using the swap command. Then we clear the point part of the number, this we do by clearing the lowword, using the clr.w command. If we fx. have to divide our number by 15, to get a floating point number, then we just use the divs.l command. Consider this example:

```
  moveq.l   #30,d0
  swap      d0
```

```
   clr.w     d0      ; could be skipped, since we know the lowword is zero!
   divs.l    #15,d0
 Register d0 no yields $00008000, which is a half!
 As you can see, administrating floating points in this way, always works,
 even with negative numbers, since the swap keeps the sign, and in divs.l
 also keeps the sign! Well, have fun...

                                          Karsten Niemeier, Dreamline Designs.
```

1.86 Saving registers

```
Saving registers:

A typical way of saving registers, could be on dbras. Consider this
example:

Example 1, UnOptimized:
        moveq.l   #10,d6
 .lop0: moveq.l   #20,d7
 .lop1: ...
        ... (some code)
  ...
        dbra      d7,.lop1
        dbra      d6,.lop0
        rts

  Example 2, Optimized:
        moveq.l   #10,d7
 .lop0: swap      d7
  move.w    #20,d7        ;D7 is now $000A0014
 .lop1: ...
        ... (some code)
        ...
        dbra      d7,.lop1
        swap      d7
        dbra      d7,.lop0
        rts

 Naturally, this loop takes a little longer than the other, but it's a lot
 faster than using a memory word, just because you need a single register!
 You also have to consider the swaps is only done each 20.th time!

                                          Karsten Niemeier, Dreamline Designs.
```

1.87 Some words about Ax's

```
When working on Address-registers the following can be used to optimized
code.

The mc680x0 autoexpands operations on Ax's to longwords, it also signextends
to longword.
The following examples will illustrate this.
```

```
Example 1: Say we have $ffff in a0 and want to add 2 the code could look
like this.

        move.l  #$ffff, a0
        add.l   #2,     a0

The result will of course be $10001, with the above in mind the optimized
version would look like this

        move.l  #$ffff, a0
        add.w   #2,     a0

Because the mc680x0 autoexpands this to longword the result will be the same
as above, and the upcode is a word smaller. This is very useful in loops
e.g. when using tables.

This can also give some problems when using Ax's for calculations.
Say we want A0 to contain $0000ffff (= -1.w) we'll have to do a

        move.l  #$0000ffff,     a0

instead of just

        move.w  #-1,         a0

because the signbit indicates a negative number the mc680x0 will signextend
to longword. (a0 is of course cleared before this.)

                Karsten Niemeier and Rasmus K. Ursem, Dreamline Designs.
```

1.88 Tips for the dos.library

```
                 Info()
```

1.89 Tips when using info()

```
When getting info about a disk by using the dos.library/Info() the following
is useful to know. In the include dos/dos.i this can be found.

*      Disk states
ID_WRITE_PROTECTED EQU 80 * Disk is write protected
ID_VALIDATING  EQU 81 * Disk is currently being validated
ID_VALIDATED  EQU 82 * Disk is consistent and writeable

This is a part of the structure filled in by Info(). The data received in
id_DiskState is a result of the dos.library internal measuring about the
disk in the drive. The dos.library first checks the disks writeprotection,
second is the validation, and if the disk wasn't write protected or
validating, it returns ID_VALIDATED and you can write to the disk.
```

```
In other words:
-If a disk is write protected id_DiskState will contain ID_WRITE_PROTECTED
 even if it isn't validated.
-If a disk is NOT write protected and NDOS id_DiskState will contain
 ID_VALIDATING.
-If a disk is NOT write protected and DOS id_DiskState will contain
 ID_VALIDATED.

Info() will return ID_WRITE_PROTECTED and not ID_VALIDATING if they're both
positive, ID_VALIDATING if the disk is NDOS and not write protected and
only ID_VALIDATED if the disk is DOS and not write protected.
Hope this makes sense.

                                        Rasmus K. Ursem, Dreamline Designs.

Readers has let me know this part was hard to understand in the previous
versions, especially if you don't know your includes by numbers, the
numbers I was referring to in the privious versions was the numbers
returned in id_DiskState. Sorry about that. °)
```

1.90 Tips for the mathffp.library

```
SPFix()
Problems when rounding off numbers.
```

1.91 Rounding problems with the SPFix()

```
The problem with SPFix() is that it returns the truncated value instead of the
rounded value. Examples:

FFP-value       Returned by SPFix()
12.3            12
12.5            12
12.9999         12
-3.4            -3
-3.9            -3

This problem can be corrected by adding 0.5 if the value is positive and
subtracting 0.5 if the value is negative. This gives the following values in
the table above.

FFP-value       Returned by SPFix()
12.3+0.5        12
12.5+0.5        13
12.9999+0.5     13
-3.4-0.5        -3
-3.9-0.5        -4

                                        Rasmus K. Ursem, Dreamline Designs.
```