MANUAL

SEKA Assembler

Commodore
AMIGA

K-SEKA
EDITOR/ASSEMBLER

for
the

Commodore
AMIGA
Microcomputer

IMPORTANT

SERIAL NUMBER _____

The serial number in your manual is the same as the serial number on your registration card.

In order to obtain technical support it is essential to complete and return the registration card enclosed in this package.

This serial number must be quoted when in correspondence with the technical support service.

Should you need to return your program disc in the unlikely event of a disc failure or loading error please return the disc ONLY.

For major upgrades i.e. from version 1 to 2 BOTH the manual and the disc must be returned.

For minor upgrades i.e. from version 1.5 to 1.6 the disc ONLY should be returned.

Kuma Computers Ltd.
12 Horseshoe Park
Pangbourne
Berkshire RG8 7JW

Tel. No. (07357) 4335
Telex 846741 KUMA G
Fax. No. (07357) 4339

K-SEKA
AMIGA

CONTENTS

K-SEKA

ISBN 07457 0182-5

Published by:-
Kuma Computers Ltd.,
12 Horseshoe Park,
Pangbourne,
Berks. RG8 7JW
U.K.
Telex: 846741 KUMA G Tel: 07357 4335

CHAPTER 1

INTRODUCTION

SEKA is a 68000 Native code Assembler for the **AMIGA**, and other 68000 Based micros. Standard Motorola Mnemonics are used, producing either absolute or relocatable code at a rate of 25,000 lines per minute for both large and small files. Executable code is produced as standard, but linkable code can be produced if the 'L' option is specified on assembly. SEKA contains a built in Linker, which runs 5-10 times faster than assembly. SEKA allows instant debugging using the built-in symbolic debugger. Code can be entered using the built-in editor, or can be loaded from another source - e.g. a text editor or wordprocessor.

All functions can be used without accessing disk - source, object, and even optional link buffers are in RAM - thus ensuring a very fast edit -assemble -(link) -debug turnaround.

Features:
* Text Editor
* Full 68000 Assembler
* Symbolic debugger
* Line Disassembler
* Formatted Listing output
* Absolute, relocatable or linkable code
* Built in Linker
* Conditional assembly
* Macro facility

This manual covers the operation of the Editor/Assembler/Disassembler/Symbolic Debugger, and the Assembler pseudo ops.

NOTE: The user is expected to be familiar with Motorola 68000 Assembly Mnemonics, or have a suitable book available.

This manual refers to **version 1.4** of SEKA.

Due to a policy of continual improvement the version supplied may be more recent. Any improvements or changes will be included in a file called READOC.ME, on the disk.

SEKA has been designed to work in 80 column mode, and although it may be used in either 60 or 80. It will look neater in 80 columns.

Running SEKA

SEKA must be run from a command line, since the disk appears empty if you look at it from the workbench. If you are in workbench, select the icon marked CLI (command line interpreter) and double click on it.

Now run SEKA, just type "seka". It may be necessary to prefix it with the disk name, e.g. "df0:seka". df0: is the built in disk drive.

Workspace

SEKA uses an area of memory as a workspace. It will prompt for WORKSPACE KB> when it is run, and will expect a size in K bytes. Ideally this should be all the memory, which is not required for other tasks, usually about 150K if you are running nothing else.

Type 150, followed by the Return key.

e.g. WORKSPACE KB>150

Entering Commands

When SEKA runs, the prompt SEKA> is displayed.

From this mode, editor, debugger, and assemble commands may be entered.

All command lines must end in <RET> (the Return, or Enter, key).

An exception to this is ending with <Ctrl-P>, (press the <Ctrl> key and the P key simultaneously), which redirects the output to the printer.

Also <ESC> is used to exit frominsert mode, modify mode, immediate assembly mode, and the screen editor. (see I, M, and A commands).

Finally, <Ctrl-C> will abort the current command, and return to SEKA command level.

e.g. The P command 'prints' lines of text to the screen, you suffix it with the number of lines to print, so...

SEKA>P10<RET> prints 10 lines to the screen

SEKA>P10<Ctrl-P> prints 10 lines to the printer

Intra Line Editor

When typing a line of text (before pressing <RET>) it is possible to delete the last character typed by pressing <BS> (the BACKSPACE key), as is common with most systems.

In addition SEKA allows;
- cursor keys <-- and --> to be used to move left and right.
- <DEL> deletes the character under the cursor.

<RET> or <Ctrl-P> terminates Intra-line editing, returning the whole line regardless of the cursor position.

In addition, ↑ recalls the previous line of text, which can then be edited, and entered by typing <RET> (or <Ctrl-P>) again.

```
e.g. type TEST LNE          display TEST LNE
          hit <RET>
          hit ↑                     TEST LNE
          type <-- <-- I TEST LINE
          hit <RET>
```

! - Exiting SEKA

To exit, use the ! (Exclamation mark) command. SEKA will prompt "Exit to system, are you sure?". Replying Y or Yes to this will exit, loosing anything you forgot to save.

CHAPTER 2

THE PROGRAM EDITOR

SEKA provides two editors for editing the program (source). The screen editor is easier to use for most purposes, and that is described first. For functions such as locating text, use a command described under LINE EDITOR.

SCREEN EDITOR

The screen editor is very simple and it is easier to use than the line editor. Note that all editing is done in the Source Buffer in memory - there is no concept of editing a disk file.

Type <ESC> in response to SEKA> prompt, to enter screen edit mode.

All text typed is inserted, cursor keys allow movement up, down, left, or right. <DEL> deletes the character under the cursor, and <BS> deletes the character to the left, i.e. the character just typed.

<Ctrl-O> will open a blank line before the current one.

Pressing <ESC> at any time from within the screen editor will exit to command level (the SEKA> prompt).

CUT & PASTE

In the screen editor, it is possible to cut out a section of text, and Paste it in zero or more times.

Initially place the cursor at the BEGINning of the piece to cut out, and mark it using <Ctrl-B>. Move the cursor to the end, and type <Ctrl-C> to CUT it out.

To PASTE, just type <Ctrl-P>. The block last cut out will be pasted in at the current cursor position.

Note that leaving screen edit mode, returning to SEKA> command level, will erase the paste buffer.

This is useful for:
a) deleting text, just cut it out. No paste.
b) moving text, cut it out, reposition cursor & paste it in.
c) copying text, cut it out, paste in immediately, reposition cursor and paste in another copy.

THE LINE EDITOR

This section describes the whole line editor commands. For other commands, see other sections. Note that within a line, the Intra-Line editor may be used - see Chapter 1.

Some commands take numeric parameters, which if omitted default to 1. This parameter is indicated as n in the below text. The Locate command takes a textual parameter, terminated by <RET>.

```
I  - Insert text
E  - Edit line Tn - Target to line n (default=top)
B  - Bottom of buffer
Un - Up n lines
Dn - Down n lines
Pn - Print (to screen) n lines
Zn - Zap (delete) n lines
Lt - Locate text t
L  - Locate next occurrance
KS - Kill source
O  - Old, opposite of KS
H  - Howbig are buffers
```

Note that all editing is done in the Source Buffer in memory - there is no concept of editing a disk file.

I - Insert text

The Insert command opens a line at the current line for more text to be inserted into the source buffer. A line number will appear, and text can then be entered. Typing <RET> will end this line, and the next line will open. This is Insert Mode, and can be exited by typing <ESC> at the start of a line.

The opened line occurs immediately above the current line, and has the same number. The old current line, and all subsequent lines, are renumbered to make room for the new line or lines.

! -_EDIT LINE

Edit allows you to edit the current line. Editing may be done with the intra-line editor, and exited by typing <RET>.

Tn - Target to line n

The concept of the current line is very important in this editor, and this command allows the current line to be set.

Just typing T will take the default n=1, or the top line, and thus T is an easy way to the top of the buffer.

If you are unsure which line you want - use the Locate command, or Up and Down commands to move relative to your current position.

B - Bottom of buffer

This command will take you to the bottom of the source buffer.

Un - Up n lines

The Up and down commands are useful to move around the buffer relative to your current position. Up will take the current position up (towards the top of the file, or lower numbered lines) by n lines. The line reached is displayed.

Dn - Down n lines

Down is like up, only you move down n lines.

Pn - Print (display) n lines

Print displays n lines on the screen, starting at the current line. The last line becomes the new current line, and thus the current line moves n-1 lines down the buffer. The P command alone (n defaults to 1) displays the current line.

To print this on the printer, use the P command with suffix <Ctrl-P>, see Chapter 1.

Zn - Zap (delete) n lines

The Zap command will delete n lines, starting at the current line. The current line and subseqent lines will be deleted, and the remaining lines renumbered to fill the gap. If the number of lines is small (<20), then they will be displayed as they are deleted. Otherwise "Sure?" will be asked, and Y or Yes should be typed to Zap the lines. Note: Use this command with caution, since there is no provision for recovering lost lines!

t - Locate text t

The locate command allows text to be located (found). The command will only find one occurence of the text, but it may be repeated by the command L, with no text. The search starts at the line following the current line, so use T first to go to the top of the buffer if you want to search the entire buffer. If the text cannot be located, Not Found is displayed, otherwise the line in which the text occurs is displayed, and becomes the current line.

L - Locate next

The L command with no text following, will locate the next occurrence of the text previously specified.

KS - Kill Source

This command erases the source buffer. The prompt "Sure?" appears, and typing Y or Yes will kill it, otherwise "** Not done" is displayed. Since this command works just by moving the EOF pointer, and setting an EOF mark, the file can be recovered using OLD.

O - Old

This command recovers a source buffer if one was deleted recently. The first character of the buffer is forced to ";". The size of source buffer is displayed if succesful (see HowBig).

H - HowBig are Buffers

This command displays the size of the:

WorkSpace (WORK),
Linker input buffer (LINK),
Source text buffer (SRC),
Relocation stream code (RELC),
Relocation stream data (RELD),
Object output code (CODE), and
Object output data (DATA).

Each is on a line of its own, and is displayed as start of file pointer and the end of file pointer in Hex, and the size of the file in decimal.

Note that the buffers (except SRC) are only displayed if they are non-zero size. This is done to simplify the display.

```
e.g. SEKA>H
Work 047B04 057B04     65536
Src  047C82 047D78       246
```

CHAPTER 3

THE ASSEMBLER

The Assembler has one command: A - Assemble.

The prompt OPTIONS› is then displayed, and simply typing ‹RET› to this prompt will assemble normally without listing. See section below on Listing & Options.

The assembler "moonlights" as a linker, and will (by default) take any linker code placed in the linker input file, and link that, as well as assembling the source to produce an executable file - see Linker Section. Normally it is not necessary to use a linker, and this section, and the linker input buffer may be ignored.

ASSEMBLY SYNTAX

All 68000 instructions are available with standard Motorola Mnemonics. Although a summary of these is given in appendix B, the user is advised to obtain a book on 68000 programming if he or she is not already familiar with it. Instructions may be entered in free format, i.e. there is no need to use TABs to format the code neatly - the assembler will automatically do this on listing.

A line of code can be considered to consist of 4 fields:

Label field	e.g. LOOP:	Must end in a colon.
Operator field	e.g. MOVE.B	Instructions are all
Operand field	e.g. D0,(A5	standard Motorola format.
Comment field	e.g. ;Setup	Must begin with a semicolon.

This may be entered in free format, with a space only required between the operator and operand fields. For example, the above line can be entered:

```
LOOP:MOVE.B D0,(A5);Setup
```

Any or all the fields may be omitted, although an operand without an operator is meaningless, and an error message will result.

LABELS/SYMBOLS

A label is just a special kind of symbol, and the two will be described here together. A symbol is a name which has an associated value. Unlike variables in high level languages, the number is constant. Symbols can be defined in two ways: (a) by placing it with a colon afterwards, as a label, it takes on the value of the current location counter, and can be used to refer to that location; or (b) by using EQU or = pseudo ops, see below.

Symbols consist of any number of alphanumeric letters or numbers, upper or lower case. No distinction is drawn between upper and lower case, e.g. Help is the same as HELP. The first character of the symbol must be a letter. Long symbols may be truncated on listing. Reserved words may not be used for symbols.

COMMENTS

Comments are ignored by the assembler. They are preceded by a semicolon (or star), and terminated by the end of line.

A comment can be preceded by a star ("*") in the first column, or by a semicolon (";") at any point in the listing. A star in any other location does not introduce a comment (rather, it indicates multiplication, or the current location counter).

When a listing is requested, the assembler formats comments which lie on a line of their own differently to those after an instruction.

NUMERIC EXPRESSIONS

Numbers may be entered in one of four bases, or as ASCII characters. In addition Labels and symbols may be freely used, and arithmetic operators Add, Subtract, Multiply, Divide, And, Or and Xor (+-*/&!~) may be used to combine these. Prefix Minus and Not (-~) are also available.

Comparasons may be performed with the operators equal_to, less_than and greater_than (=<>), which return a value of 0 if false and 1 if true. Square brackets [] may be used to indicate ordering of arithmetic, otherwise the operations are performed strictly in the order in which they are entered.

The four bases allowed are: Decimal, Hexadecimal, Octal, and Binary. Default are decimal numbers, Hex require a $ prefix, Octal an @ prefix, and Binary require a % prefix.

ASCII constants need to be placed in a pair of single or double quotes, e.g. 'A', 'AB', 'ABC', or 'ABCD'. Where more than one character is placed inside the quote marks, the first character fills the more significant byte, etc, and the last character is allocated the least significant byte. i.e. the characters are packed at the bottom of the word or longword. "ABC" = $00414243. Thus a MOVE.B £'A',D0 and MOVE.L £'A',D0 will both place an 'A' in the low byte of D0.

The location counter is represented by *, and may be used freely to represent its current value. NB: * represents the current location counter, not the value at the start of instruction! Beware of DBRA D0,* !! However, *(PC) will give the anticipated result, as will DC.W HERE-*, THERE-* which is the same as DC.W HERE-* followed by DC.W THERE-*

When the prompt OPTIONS> appears, none or more options can be specified.

Options V, E, and P

Listing can be sent to the screen by entering "V" for video. A formatted listing will be produced. A "P" (or "E") option will send the formatted listing to the Printer - a listing name will be prompted for. For neatness, it is best to assemble without listing until errors have been removed. If no device is specified, no listing is produced.

Option "H" will direct the listing to hold between pages, which is useful both on screen, and with manually sheet fed printers. Continue by hitting any key.

Option "O" will optimize branches. Only branches with no .S or .L are optimised, if you deliberately use .L the assembler assumes you have a reason! Note that this modifies the source, it adds ".S". Note that option "O" takes a long time by SEKA standards, it is NOT recommended and can cause strange effects sometimes.

All short branches that are out of range are automatically upgraded by SEKA to .L branches, and a warning message is produced. This occurs whether or not option "O" is specified.

Option "L" will produce linkable code, see the Linker section later in this manual.

A symbol table is produced as standard, when listing is requested. Entries are of the form NAME...VALUE, with a "+" sign after the value signifying a relocatable or external name. If the name is a Macro name, -MACRO- is displayed instead of a value.

Errors raised in the assembly (except warnings generated by changing .S branch to .L) are treated as fatal if no listing has been requested, otherwise all the errors are listed. When the assembler stops on error, the current line is set to the one which contains the error, to aid rapid correction.

It is recommended to assemble without listing until all assembly errors have been removed, unless a listing with errors is deliberately required. Most users find that the Stop-on-error-with-quick-edit-and-reassemble approach is easy and simple to use.

More than one option can be specified on the OPTIONS line:

```
e.g  SEKA>A
     OPTIONS>VH  will list to screen, holding
                 between pages.
```

PSEUDO OPERATORS

Pseudo operators (pseudo ops for short) are assembler directives. They do not usually generate code, but instead affect the operation of the assembler.

Three pseudo ops do generate code: DC.B, DC.W and DC.L will generate Byte(s), Word(s) or Longword(s) containing the value of the arguments. BLK will leave space for data tables, etc. It takes one or two parameters, the first is the space in bytes (or words, or longwords if .W or .L suffixes are attatched), the second the desired fill value.

Code generation begins at an origin. This can be relative code, at which case the code begins at relative 0 - the CODE and DATA pseudo op; or absolute code, in which case the code has a prespecified start address - specified by the ORG directive. Assembling straight into memory will occur automatically with relative code, but after an ORG a LOAD directive must be included in the source code. This specifies the address in memory to load the code - and is usually the same as the ORG address.

The assembler will stop assembling the source file when it meets an END directive, or at the end of file.

TABLE OF PSEUDO OPERATORS

DC	Defines a byte, word, or longword in the object code. More than one number may be placed after the directive, separated by commas. In addition, messages may be placed after the DC.B directive, in single or double quotes. Default size=.B
BLK	Defines a block of memory. The first parameter specifies the size of the block, and the second the value to fill it with. If the second is omitted, it is undefined. Default size=.B
ORG	Takes a single parameter - the address to start assembling at. Switches on Absolute code mode.
LOAD	Takes a single parameter - the address to start loading into memory from. Only works in Absolute code mode.
CODE	Switches on Relative mode, Code segment.
DATA	Relative mode, Data (uninitiallized) segment. (note that some assemblers call this BSS).
EVEN	Forces the address even. If odd,defines a byte.
ODD	Forces the address odd. If even, defines a byte.

END — Ends the assembly process.

EQU — Both EQU and = can be used to seta symbol to a value. The symbol name is entered, followed by a colon, as normal, and then by EQU. Instead of taking on the value of the Location Counter, the symbol takes on the value of the expression after EQU.

= — Identical to EQU, only no colon is needed after the symbol name.

LIST — Turns listing on. Note listings are only produced if a E, P, or V option is given to OPTIONS>. It may also be used with a parameter to selectively enable Listing of Macro Calls (LIST C) Macro definitions (LIST D) Macro Expansions (LIST E) and Code extensions (LIST X).

NLIST — Turns listing off. May also be used with a parameter to switch off listing of the items specified in LIST.

PAGE — Forces a new listing page. The PAGE directive lists on the first line of the new page.

IF	Conditional assembly - takes a numeric expression, which if it evaluatesto 0 (False) does not assemble the following lines, and if non zero (True) assembles the following lines. See ELSE and ENDIF. May be nested to 8 levels.
IFB	If Blank usually takes a macro argument, and is true if the argument is blank, (i.e. null). e.g. IFB ?2
ELSE	Toggles the conditional assembly condition, i.e. if assembling, ceases to assemble and vice versa.
ENDIF	Terminates the conditional assembly block.
MACRO	A symbol given before the macro op becomes the name of the macro, and it may be called by typing that name, followed by a list of arguments, separated by commas. The code of the macro (its definition) follow on subsequent lines, and assembly is turned off.
ENDM	Ends a macro definition, and restores assembly.

?n	used at this point. Valid only in macro definitions. n=1 thru 9
?0	Generates three digits, unique for every macro call. May be used to create local labels (e.g. X?0). Valid only in macro definitions.
GLOBL	Takes a list of symbols which are to be treated as globals - see linker section. Must be the first operator in the file. May only be used with the "L" option.
PWID	Sets the printer page width for listings. default 80.
PLEN	Sets the printer page length. The default is 66 lines.
PINIT	Sets a code sequence to initialize the printer, useful to set 132 columns, USA font (with £), etc. e.g. for Epson to set 132 columns... PINIT 15 Note the above 3 are ignored if printer listing not requested.
ILLEGAL	Generates an illegal instruction,useful for causing exit to the SEKA debugger (or other debug system). identical to DC.W $4AFC.

LINE A)	Generate calls to emulator traps, e.g. LINE A $123 is LINE_F) identical to DC.W $A123.
ALIGN	Will align the object to an n byte boundary;ALIGN 2 is identical to EVEN, ALIGN 4 longword aligns. Only powers of 2 give sensible results. (i.e. 2, 4, 8, 16, etc).

CHAPTER 4

SYMBOLIC DEBUGGER

The symbolic debugger is a built in Machine code monitor with extensive use of the assembler's facilities, such as symbol table access, arithmetic operations, and input in any base. In addition, the debugger offers a disassembler, a line assembler, trace, multiple breakpoints as well as examine/modify registers & memory, fill, copy, search etc.

```
X  - Xamine all registers
Xr - Xamine/Modify register r
Gn - Goto address n
Jn - Jump to subroutine at n
Qsn- Query (examine) memory at n
Nn - mNemonics of memory at n
An - Assemble immediate at n
Msn- Modify/Examine memory at n
Sn - Single step, n steps
Fs - Fill memory (bytes)
C  - Copy memory
?  - Display value of expression Y - VDU
```

Various default values apply if the numeric parameter is omitted from the commands above. In the case of Single Step, the default is 1. Goto, and Jump to Subr, default to the current PC. The memory addresses in Q, N, M, and O, all default to the last used address of one of the four (the current object location). In addition, a size can be specified for F, Q, and M. This will default to Byte, but .W or .L can be specified for Word or Longword operations.

X- Xamine registers

The 8 data, 8 address, PC, SR, USP and SSP registers are displayed. The flags indicated by bits in the status register are explicitly displayed and the current instruction (i.e. the one which would be executed next) is disassembled and displayed.

The format of the display is:

```
D0=00000000 00000000 00000000 00000000
D4=00000000 00000000 0000000 00000000
A0=00000000 00000000 00000000 00000000
A4=00000000 00000000 0000000 00000000
SSP=00000000 USP=00000000
SR=A31F tsxnzvc PC=000000 BR   $000008
```

Across the top line are the 8 data registers, D0, D1, D2, D3, D4, D5, D6, and D7.

The next line displays the Address registers, A0, A1, A2 A3, A4, A5, A6 and the stack pointer - A7 - which is the User or Supervisor SP depending on the 'S' bit in the Status Register. The third line displays the two stack pointers, the status register word, the flags set in the status register, the program counter, and the instruction at which it points.

Xr - Xamine/Modify register

The register indicated by r is displayed, and the contents can be altered. To alter the number, just type in the new value, followed by <RET>, and the new value will be displayed. If no change is required, just type <RET>. All registers are 32 bits except for SR, which is 16. SP and A7 refer to USP or SSP, depending on the S bit of the status register. Valid registers are:

D0 .. D7	Data registers
A0 .. A7	Address registers
SP	Stack pointer (=A7)
USP	UserStack pointer
SSP	Supervisor Stack pointer
SR	Status Register (16 bit)
PC	Program Counter

In addition the following pointers can be examined and modified:

SOF the start of source pointer;
SOL start of link pointer
EOF the end of source pointer;
EOL end of link pointer

Gn - Goto address n
The address n is placed in the program counter, If n is not given, the PC remains unchanged. Breakpoints are prompted for, up to 8 in all. To set a breakpoint, just type in the address after the prompt. Typing <RET> after a prompt sets no breakpoint, and begins to execute the code from the address in PC.

Jn - Jump to Subroutine
See G above, only a return address is placed on the stack (If SR=User mode, USP. If SR=Supervisor mode, SSP). The stacks are reset, thus any information previously on the stack is lost.Note that on return, the PC points to an ILLEGAL instruction, somewhere in the heart of SEKA. This is correct, it's used as a breakpoint. Since the RTS instruction looses the value of PC before return, there is no way to tell if you exited via a normal exit or not.

Qsn - Query memory
128 bytes, 64 words, or 32 longwords of memory are displayed in Hex starting at n, or defaulting to the current object location. The current object location is advanced to the address one after the last byte. The Ascii characters represented by this are displayed on the right of the hex display.

Nn - mNemonics
16 lines of code are disassembled to the screen, starting at n, or defaulting to the current object location. The current object location is advanced to one past the last address displayed. See section on DISASSEMBLER, below.

An - Assemble immediate
The address n is displayed, and the user is in immediate assembly mode. An instruction may be entered, and it will be assembled directly into memory, and the next address will be displayed. The mode may be exited by typing <ESC>. If an error is made in the mnemonic, an error message will be displayed, and the mode exited. Note: It is not possible to omit n for this command, but "A*" may be typed, to indicate to assemble from the current location.

Msn - Modify/Examine memory
The address n, defaulting to the current object location is displayed, and the contents of the byte, word or longword is also displayed. To modify this location, simply type in the desired value, and the displayed memory will update itself. To advance to the next location, type just <RET>. To exit to SEKA command mode, type <ESC>.

Sn - Single step
The S command will single step the program, using the Trace facility on the 68000. If n is specified, n steps will be performed before reporting back to the debugger. The PC must be set up beforehand, either as a result of a previous S or G command, or as a result of an explicit XPC.

STEP will not trace through TRAP calls or LINE-A LINE-F calls since these will usually be calls to the operating system. This is done automatically by STEP by placing a breakpoint after the call.

Fs - Fill
Three parameters will be prompted for, BEGIN, END, and DATA. The data byte, word, or longword will be filled in between BEGIN up to, but not including END.

C - Copy
Three parameters will be prompted for. BEGIN, END, and DESTINATION. The memory will be copied from the area BEGIN up to but not including END, to the area starting at DEST. The copy is an 'Intellegent' copy, and will not overwrite the data it is copying even if the destination area overlaps the source area.

? - Display value
The expression or value after the ? will be evaluated, and the result displayed in decimal and Hexadecimal. This is useful for everything from examining the value of a symbol, to performing calculations. It can also be used to convert from one base to another.

Y - VDU
This enters a simple VDU mode, where the screen and keyboard communicate with a device on the serial line, using default baud rate & characteristics set on power up or other initialization program. Exit by typing <Ctrl-C>. Note: <Ctrl-C>, <Ctrl-S>, and <Ctrl-Q> are interpreted in their normal context, and not sent down the serial line.

DISASSEMBLER

A standard feature of **SEKA** is a simple line disassembler, fully integrated with the main assembler package. The Mnemonics used are standard Motorola - just like the assembler. The disassembler is invoked by the command **N** (for mNemonics) followed by an address. 16 lines of code are displayed.

The disassembler can disassemble all 68000 instructions, but is not defined for invalid instructions. It should always be remembered that some areas of memory are data areas, and do not contain valid code. Disassembling these can give incorrect instructions - you have been warned!

Although its primary function is as a debugging aid, it can be used to disassemble blocks of code. This can be a slow process, but the line disassembler is preferable to none. It should be remembered that some instructions can display differently to the instruction typed into the assembler. All numbers and addresses are disassembled in hex, and although .B .W and .L suffixes are sometimes added in places where they could be omitted, in unambiguous places they are often not displayed. Instructions where the assembler has used a Quick form, etc, are displayed as such. In addition, instructions such as EXG can display the registers in either order - the function is identical.

CHAPTER 5

LOADING AND SAVING

The filing commands are:

R read source file	W write source file
	WO write object file
RL read linker file	WL write linker file
RI read image	WI write image
V view directory	
KF kill file	> copy output to file

V - View Directory

will display the directory in the form:

```
nnnnn Blocks Used mmmmm Blocks free
ssss fffff      ssss fffff      ssss fffff
ssss fffff      ssss fffff      ssss fffff
ssss fffff      ssss fffff      ssss fffff
```

where nnnnn and mmmmm are space used and free on the entire disk, and ssss is the size (in bytes) of file fffff. The command may be suffixed by a directory name, otherwise it will display the current directory.

When a directory is specified with View, it attaches to the directory (i.e. makes it the current one), and then displays it.

KF - Kill file

KF will prompt for a file name and delete that disk file if it exists.

R/W source
Prompt for filename (default extension .S). Read will read the file into the current location in the source buffer - for overwrite simply kill the source first (KS). Write will write the whole source buffer to the file.

<Ctrl-W>
From screen editor, it is possible to output the current paste buffer to disk by typing <Ctrl-W>. A filename will be prompted for. Default extension is .S, since it is a source file. To read it back in, use the R command.

WO object
Prompt for filename (no default extension). WO will save the object on disk, with a header and relocation information: this is the usual way of saving an executable file.

RL/WL link
Prompt for filename (Default extension .LNK). See linker section. WL will write a linkable file FROM OBJ BUFFER. This can be used after an Assemble Option>L to save the linkable code produced. RL will read a disk file and append it to the link buffer. Use KL to kill the link area if necessary. Many files may be read into the link buffer and linked together. See also CL.

RI/WI image
For flexibility, these commands will allow a data or other file to be read in or written out. Both prompt for filename and start/stop addresses. On read: put STOP as -1 for entire file.

> copy output to disk

Prompts for FILENAME. Will subsequently copy anything which appears on the display/printer to the specified file on disk. Finish the operation with either another ">" command, or by exiting SEKA.

Note on filenames If no name is specified when FILENAME is prompted for, the operation is aborted and "** Not Done" is displayed.

CHAPTER 6

LINKER

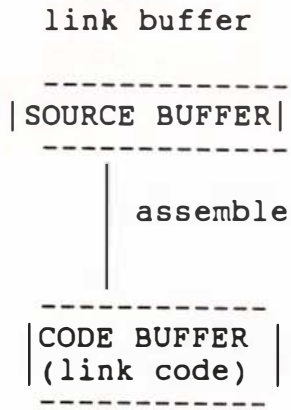
Convel assemblers produce linkable code which then needs to be linked by a linker to produce executable code, even if no additional modules need to be linked in.

SEKA is different. The assembler can produce either executable code, or linker code, and for small programs there is no need to use a linker. Since the assembler will accept either standard 68000 source code or linker code, it can act as a linker, linking modules of linkable code (which have been loaded into the LINK buffer), and producing executable code.

A novel "extra" SEKA provides is the ability to link modules of linkable code AND assemble a source AT THE SAME TIME, so enabling one module to be worked upon - and small changes made - and the results of tests to be seen very rapidly.

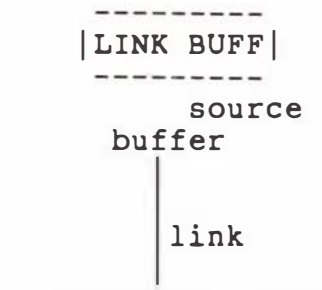
The 'L' option on the assembler command controls the OUTPUT: default is executable, with L option gives Linkable output.By default both link input and source input files are taken by the Assembler/linker, but when L is specified, the Link input file is not input (since link input => link output is not allowed).

Assembling with linkable output
Use the A command, with L option:



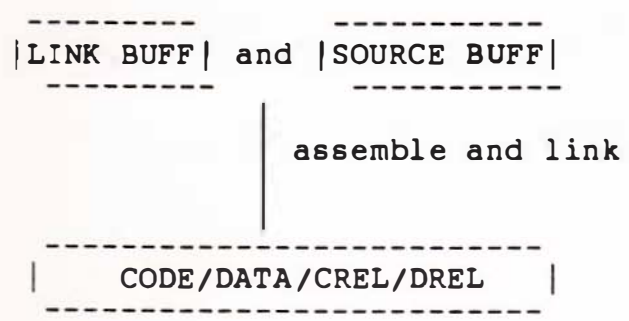
Note that the Link buffer remains unused and unaltered.

Linking
Kill the SOURCE. Use the A command without L option:



Note that the source buffer must be zero size, unless it is desired to assemble as well as linking.

Assemble & Link
Use the A command without L option:



LINKER COMMANDS

CL - Copy Obj to Link input buffer
Use this after an assemble-option-L to place the linkable code in the link input buffer, ready for a link. Prompts "Sure?" because it DELETES THE SOURCE buffer!! Beware!

CL, like RL, appends to the Link input buffer, (as to overwriting, see KL).

KL - Kill Linker Buffer
Kills linker buffer. Prompts "Sure?".

RL and WL

See Filing Section. WL can be used (instead of CL, or possibly in addition to), to save the linkable file assembled into the OBJ buffer. RL can be used to read in linkable files - these are read in to the LINK INPUT BUFFER. Note: RL Prompts "Sure?" because it DELETES THE SOURCE!

RL, like CL, appends to the Link input buffer, (as opposed to overwriting, see KL).

Relocation Modes - a clarification

All arithmetic can contain externals or relocatables, providing the result is absolute, relocatable or external, and not a composite of these (like REL+REL or EXTERN-REL).

```
e.g. HERE: DC.L THERE-HERE
     THERE:
```

is valid, and the result is absolute (a pure number).

Similarly BRA HERE does an implied HERE-* (* is the current location counter, initially Code Relocatable) and is OK if HERE is a label to the same REL section of the program.

BUT: BRA EXTERN will attempt to calculate EXTERN-*. (EXT-REL), and will fail with an error.

Use JMP EXTERN

In practise don't worry - until you get a relocation mode error.

CHAPTER 7

EXAMPLE PROGRAM

This section leads the novice user through a simple program, which will help to understand the simple and easy to use features of SEKA.

Output characters to screen

Type the following underlined sections into SEKA. The non-underlined sections represent output given as a result of the commands typed (note that addresses may differ on different versions of the machine or operating system):

Note that the file is shown being entered with the I command on the line editor. Most users find it easiest to type <ESC> and use the Screen editor, but it is easier to show the I command in the examples here.

```
WORKSPACE        KB>150
SEKA>h
Work 02EB48 054348 153600
Src  02EB4E 02EB4E      0
SEKA>i
1 ;Program to print some characters on screen
2
3 start:move.l execbase,a6;Open DOS library
4 lea dosname,a1
5 jsr OpenLib(a6)
6 move.l d0,a6;Base of AmigaDOS
7 jsr Output(a6);get output stream
8 move.l d0,d1
9 move.l £buf,d2
10 moveq £bufsiz,d3
11 jsr Write(a6);Write text to screen
12 clr d1
13 end:jsr Exit(a6)
14
15 align 4;Align to longword
16
17 dosname:dc 'dos.library',0
18
19 align 4
20
21 buf:dc 'Hello World',LF,0
22 bufsiz = *-buf
23
24 end
25 <ESC>
SEKA>h
Work 02EB48 054348     153600
Src  02EB4E 02ECDE        400
SEKA>
```

Now the main part of the program has been typed in, but the equates for the libraries reside in a file on disk, called AmigaDOS.i. Read this into the file, at, say line 2.

```
SEKA>t2
    2
SEKA>r
FILENAME>amigados.i
SEKA>
```

now lets assemble the source... no options for just assembling.

```
SEKA>a
OPTIONS>
** Undefined Symbol
    46 buf:dc 'Hello World',LF,0
SEKA>
```

It stopped on an error. Something is undefined. Come to think of it, we never did define LF did we? Lets do so.

The cursor is currently positioned on the line with the error. Lets put the definition on the line before. (the I command inserts before)...

```
SEKA>i
   46 LF=10
   47 <ESC>
SEKA>a
OPTIONS>
No Errors
SEKA>h
Work 02EB48 054348      153600
Src  02EB4E 02EE56         776
RelC 02F19C 02F1AC          16
RelD 02F1B0 02F1B8           8
Code 02F1C8 02F210          72
SEKA>
```

don't be upset by the sudden increase in complexity of the Howbig command. What it's trying to tell you is that there is now some object code in the CODE buffer (plus information about how to relocate it, should you save it to disk, and reload it in another location - this is stored in RELC and RELD).

Well, since it's there, lets look at it... we can do this by using a label - remember START: ? First, use the N command to get mNemonics.

```
SEKA>n start
02F1C8 MOVE.L $000004.L,A6
02F1CE LEA    $02F1F4.L,A1
02F1D4 JSR    $FE68(A6)
```

alternatively, we can look at the machine code itself, using Q (Query Memory). This has Byte, Word, or Longword options. Lets use the Word option.

```
SEKA>q.w start
02F1C8 2C79 0000 0004 43F9 0002 F1F4 4A4E FE68 ,y....Cy..qtN.~h
02F1D8 2C40 4EAE FFC4 2200 243C 0002 F200 760D ,@N..D".$<..r.v.
02F1E8 4EAE FFDO 4241 4EAE FF70 FFFF 646F 732E N..PBAN..p..dos.
02F1F8 6C69 6272 6172 7900 4865 6C6C 6F20 576F library.Hello Wo
02F208 726C 640A 0000 0000 0000 0000 0000 0000 rld.............
02F218 0000 0000 0000 0000 0000 0000 0000 0000 ................
02F228 0000 0000 0000 0000 0000 0000 0000 0000 ................
02F238 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Well, at least I can make out "dos.library" and "Hello World" in there. Probably a good sign.

Another way to see what the code looks like is to get an assembly listing, e.g.

SEKA>A

OPTION>V (or option P for printer)

anyway, back to the matter in hand... we have an untested program. Lets save it before we try and run it! Place a blank, formatted disk in the main disk drive.

```
SEKA>vdf0:
        0 blocks used 1758 blocks free
** No files
SEKA>w
FILENAME>hi
SEKA>v
        2 blocks used 1756 blocks free
     777 hi.S
SEKA>
```

Now it's securely stored, immune to crashes (well, if you take the disk out of the drive, it would be!), we can try to run it.

Remember it starts at START, and ends at END (were it normally EXIT's back to the operating system).

```
SEKA>g start
BREAKPT>end
BREAKPT>
```

```
D0=00000000 00000000 0002F280 00000000  04=00000000 00000000 00000000 00000000
A0=00000000 00023820 00000000 00000000  A7=00000000 00000000 000920E 0002412E
SSP=000250CE USP=0002412E SR=0004 z PC=02F29E JSR  $FF7D(A6)
```

```
SEKA>
```

What happened to the message? well, it was sent to the default output device, which is still the system window. The SEKA window does not alter this. So flip back to the system window...

Good. It seems to work.

Lets save the object program, so it can be run from the system.

```
SEKA>wo
FILENAME>hi
SEKA>!
Exit to system, sure? y
```

APPENDIX A

AmigaDos Introduction

The Amiga operating system is quite complex if you want to use it to its full extent, but the outermost DOS level is quite simple, and explained briefly here.

All calls to the DOS are by JSR calls to an address made up of a base address (the address of the library) and an offset. All offsets are negative. Since the base address varies, we have to call another routine to find where it is. It will never vary during the running of your program, but might from one run to the next.

Only two libraries need to be used: "EXEC" the most primative library, which contains calls to open all the others, and "dos.library", the DOS library.

The EXEC library

The base of the EXEC library is to be found in location 4 (which we will call ExecBase for neatness). This is the only pre-ordained value in the entire system. Once that value has been picked up, and by convention placed in address register 6, A6, (this is essential, since some of the libraries assume that it is there and call other routines!), then the call OpenLib(A6) can be made to open other libraries.

OpenLib (offset -408) requires a library name string in A1, and returns the new library base in D0.

```
e.g.
move.L ExecBase,A6
lea    dosname,A1   ;Library name
jsr    OpenLib(A6)  ;call Exec's OpenLib routine
move.L D0,dosbase   ;to get the base of AmigaDOS
                     library
```

note: the labels ExecBase and OpenLib are to be found in the AmigaDOS.i equates file on the disk.

The AmigaDOS library

This library is like the DOS on a normal small computer system. It allows input and output to files or other devices, but does not have facilities to access graphics, or sound. These must be accessed via the rest of the operating system - the other libraries. See your Amiga dealer for more information.

All calls to AmigaDos take the same form as the EXEC call, OpenLib. There is a table of the offsets on the disk in the file "AmigaDOS.i". These offsets must be applied to the AmigaDOS base, not the EXEC base (see example below).

The table below sets out - very briefly - the function of the routines, and lists the parameters they require, and the values they return.

Parameters are passed to the routines in Data registers, the first parameter in D1, the second in D2, etc. All returned values are returned in D0.

The table is set out below as:

```
Routine( D1, D2, D3, ... ) : result_in_d0
```

mnemonic names are given to the parameters and the result for descriptive value, notes on these names are to be found below.

AmigaDOS functions

`Open( name,mode ) : Handle`	Opens a file.
`Close( handle )`	Closes an open file.
`Read( handle,buffer,maxsize ) : size`	Reads a block
`Write( handle,buffer,size ) :size`	Writes a block
`Input() : handle`	Returns the default input
`Output() : handle`	Returns the default output

Seek(handle,position,mode) : oldpos	Positions file pointer
Delete(name) : success	Deletes file or directory
Rename(oldname,newname) : success	Renames a file
Lock(name,mode) : lock	Obtains a lock on a file
UnLock(lock)	Releases the lock
DupLock(lock) : lock	Returns a duplicate lock
Examine(lock,infobuff) : success	Finds first file in lock
ExNext(lock,infobuff) : success	Examines the next file
CurrDir(lock) : oldlock	Sets the current directory
Exit(returncode)	Exits to the system

```
WaitForCh( handle,timeout )
: success                          Waits for a
                                   character
```

note: there are other AmigaDOS calls, but these are the most useful ones. Note all I/O must go through READ and WRITE, even interactive channels to the keyboard and screen.

Notes on AmigaDOS calls

handle	A file handle is returned from Open, and used by other file routines. A return of 0 indicated Open failed.
lock	A lock is a bit like a handle. Since it is cheaper to obtain a lock on a file, than it is to open one, use LOCK if you just want to check it exists.
name	A pointer to a null terminated file name.

mode	for open ... use 1005 for oldfiles, 1006 for newfiles. for lock ... use -2 for shareable locks. for seek ... use -1 for Position relative to file start 0 for Position relative to current 1 for Position relative to file end
position	A (possible negative) character position in file. It may be relative to the file start, end, or current position, see above entry for mode.
infobuff	The Examine/ExNext commands return lots of useful information. The block must be longword aligned.
success	Zero for fail. nonzero for success.
buffer	Read and Write require a pointer to a buffer containing the data, even if it is just a single character.

size	The size of the buffer. Read returns the actual size read, and write returns the actual size written (which only differs if the disk is full, or other error).
timeout	A time in microseconds.

APPENDIX B

Instruction Set Summary

ABC		MOVEM	.WL
ADD ADDA ADDI	.BWL	MOVEP	.WL
ADDQ	.BWL	MULS	
ADDX	.BWL	MULU	
AND ANDI	.BWL	NBCD	
ASL	.BWL	NEG	.BWL
ASR	.BWL	NEGX	.BWL
Bcc		NOP	
BRA BR		NOT	.BWL
BSR		OR ORI	.BWL
BCHG		PEA	
BCLR		RESET	
BSET		ROR	.BWL
BTST		ROL	.BWL
CHK		RORX	.BWL
CMP CMPA CMPI	.BWL	ROLX	.BWL
CMPM	.BWL	RTE	
DBcc		RTR	
DBRA DBR		RTS	
DIV		SBCD	
DIVU		Scc	
EOR EORI	.BWL	STOP	
EXG		SUB SUBA SUBI	.BWL
EXT	.WL	SUBQ	.BWL
JMP		SUBX	.BWL
JSR		SWAP	
LEA		TAS	
LINK		TRAP	
LSL	.BWL	TRAPV	
LSR	.BWL	TST	.BWL
MOVE	.BWL	UNLK	

Mnemonics on same line are synonyms.

cc

T	VC
F	VS
HI	PL
LS	MI
CC HS	GE
CS LO	LT
NE	GT
EQ	LE

Addressing modes

Dn	d.W
An	d
(An)	d(PC)
(An)+	d(PC,xn)
-(An)	£d
d(An)	SR
d(An,xn)	CCR
	USP

Please refer to a suitable reference book for further details, Motorola 68000 Users Manual (Motorola) or Leventhal & Kane "68000 Microprocessor" (Osborne/McGrawHill)

APPENDIX Z

Summary of all Commands

```
!  - Exit to system Chapter 1
>  - Set output file Filing
?n - Display value. Debugger
A  - Assemble file. Assembler
An - Assemble immediate. Debugger
B  - Bottom of file. Editor
C  - Copy memory. Debugger
CL - Copy linkable code. Linker
Dn - Down n lines. Editor
E  - Edit current line. Editor
Fs - Fill memory bytes. Debugger
Gn - Goto address n. Debugger
H  - Howbig is file. Editor
I  - Insert at current line. Editor
Jn - Jump to subroutine. Debugger
KS - Kill source buffer. Editor
KL - Kill linker buffer. Linker
KF - Kill File. Filing
Lt - Locate text. Editor
Msn- Modify/Examine memory. Debugger
Nn - mNemonics. Debugger
O  - Old source. Editor
Pn - Print n lines. Editor
Qsn- Query (display) memory. Debugger
R  - Read file. Filing
Sn - Single step, n times. Debugger
Tn - Target to line n. Editor
Un - Up n lines. Editor
Vd - View directory. Filing
W  - Write file to disk. Filing
```

```
X  - Xamine registers Debugger
Xr - Xamine/Modify registers. Debugger
Y  - VDU. Debugger
Zn - Zap n lines.Editor


 s = size (.B .W .L). Default=.B

n = number (e.g. $100, LABEL+2).
    Default=1 or previous

d = directory (e.g. DF1:C)Default=Default
    directory

t = text to be located. Default=previous
    t

r = register name (e.g. D0)
```

Note: All ranges given by START STOP are exclusive of the STOP address,

e.g. START>5 STOP>8 means 5, 6 and 7

Screen editor

<Ctrl-B>	Begin block Editor
<Ctrl-C>	Cut Block Editor
<Ctrl-P>	Paste block Editor
<Ctrl-W>	Write block to disk Filing
Cursor Keys	Move
<DEL> and <BS>	Delete right and left.