COMPASS #1.2

Compjoetania

The Next Generation

Foreword

This is the manual for the “Finally Free Edition” of Compass.

After several years of people looking to obtain a version of Compass we decided to release version 1.2.09 as freeware. The software needs to be registered with us, so this version is registered to the fictitious user “Finally Free”.

Originally the software came on a floppy with a printed manual.
This PDF tries to make up for the missing dead trees edition of the manual, with some small fixes and updates as deemed necessary.

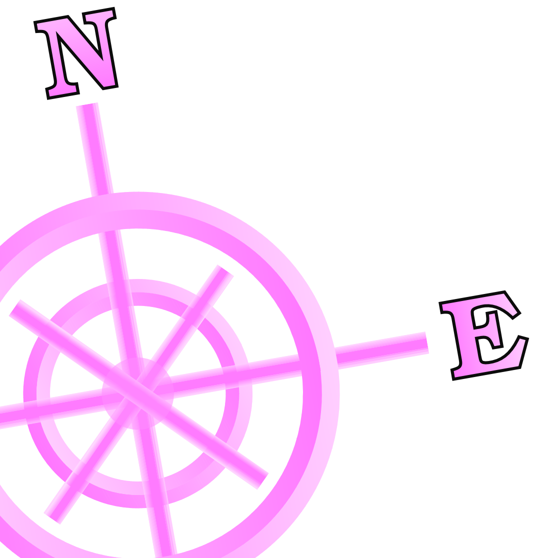
A plain MSX readable text version of the original manual can be found on disk in the filename C12MANUA.TXT

```
File: C12MANUA.TXT - 24/04/1999
Subject: Compass #1.2.08 manual
(c)1998 Compjoetania The Next Generation
Note: do not distribute this document, it is copyrighted [1]

Because Compass #2.0 is on its way, we've decided not to print a new manual
for Compass #1.2. [2]

MSX, MSX-DOS and R800 are trademarks of ASCII Corp.
Z80 is a trademark of ZILOG Corp.
MemMan is developed by the Msx Software Team

Despite all the care taken by the production of this text, Compjoetania TNG
can not be held responsible for any possible damage resulting from errors
contained in this document.
```

1 This is the original line, for the ‘Finally Free’ edition you are allowed to distribute this file together with the software.

2 Compass 2.0 has been in closed internal beta for over a decade by now, but at the time of writing there are no plans to release the software.

Table of Contents



Preface

Why another MSX assembler?

In the dark past many different assemblers were produced for the MSX system in addition to the already available CP/M assemblers.

Along with the advantages of these products, there were also a lot of disadvantages.

Some of them had an integrated environment but weren't able of handling large sources, others were very powerful but needed a separated editor causing a waste of time when testing and debugging. Most of these programs didn't take advantage of the new developments in the MSX scene. Just to name a few of them: Turbo R (the R800 instruction set), MemMan, DOS 2.xx (subdirectories).

There just wasn't an assembler which combined enough advantages to satisfy almost everyone.

Compass is our attempt to create an easy-to-use, menu driven and extensive package specifically designed for the MSX-system. The name "Compass" is a contraction of the words "COMPjoetania ASSembler". We would like to thank everybody who, one way or another, has contributed to the development of Compass.

We hope you enjoy Compass.

September 1998, Compjoetania The Next Generation

Original concept and program by Compjoetania (1995):

- Remo Jongeneelen
- David Heremans
- Eric van Beurden

Compass #1.2 by Compjoetania The Next Generation (1997...)

- Jon De Schrijder
- David Heremans
- Wouter Vermaelen

1 Preparations

It is important that you read this chapter before you proceed. It contains important information about the program as a whole and the contents of the package.

1.1 Service

If you obtained a physical copy and there are -despite our efforts- any errors (disk I/O errors etc.), you can return the malfunctioning disk. We will replace this disk with a better one without any further costs. This replacement service ends 2 years after the original purchase was made.

You can contact us by following means:

Good old snail-mail:

Compjoetania TNG
Goorweg 24
2221 Booischot
Belgium
Europe

Or if you prefer an electronic mail format:

E-mail: msxctng@telenet.be

Or if you are impatient the MSX community might be a good resource. An international MSX scene is still active on the website of the MSX Resource Center which can be found at https://www.msx.org
The members of Compjoetania TNG visit this site regularly and a lot of the other regular folk on the site are MSX experts who are willing to help out with a lot of MSX related questions.

1.2 The contents of the package

Compass v1.2 was released on a double-sided DD disk with a full colour label.
The disk contained:

- The program Compass:
  COMPASS.COM
  COMPASS.DAT
- The older manual in TXT format
  C12MANUA.TXT
- Compass version history list
  COMPASSV.TXT
- Utilities to extract and compress PMA archives
  PMEXT.COM
  PMARC.COM
- PMA archives, containing useful information for programmers

For the Finally Free edition a zip file was released containing:

- A disk image with
  - the Finally Free Compass program
  - MSXDOS.SYS and COMMAND.COM[3]
  - the original v1.2 extra's
- This manual in PDF format
- The content of the PMA archives as simple files

We expanded the PMA archives for the convenience of modern OS users, since PMA extractors aren't as ubiquitous as zip extractors.

An original 1.0 and 1.2 disk

3 Kazuhiko Nishi, head of the MSX Association which currently hold the rights, has confirmed that all the MSX system ROMs (and this includes the MSX-DOS files) are free to use and redistribute; with the sole exception of MSX-BASIC, for which Microsoft retains the copyright.

1.3 Configuration

Compass runs on every MSX2, MSX2+ and MSX Turbo R computer with a memory mapper of at least 128kB Ram, though for convenient use, we recommend a memory capacity of at least 256kB. 'Fixed' ram modules (this is: not memory mapped ram) are not supported.
(And you'll need a double-sided disk drive for the original disk to work of course.)

Compass supports the following configurations:

- MSX-DOS 2.xx
  If you have DOS 2.xx you are able to browse through subdirectories and you can take advantage of the DOS 2.xx Ramdisk.
- MEMORY MAPPER
  Compass supports multiple memory mappers.
- MSX TURBO-R
  If you run Compass on an MSX Turbo R you are able to switch between the Z80 and R800 (ROM/DRAM) processor.
- HARD DISK
  You are able to install Compass on your hard disk, just place the COMPASS.COM and COMPASS.DAT files in the same directory.
- MemMan 2.42
  Compass uses -if already installed- the memory management routines provided by MemMan. This allows you to use TSR's while you are using Compass. If you don't use MemMan then Compass will use the routines provided by DOS 2.xx . If you don't use either of them, Compass will resort to its own memory routines. You can find MemMan 2.42 and the accompanying programs in a packed archive on the disk.

Compass uses screen 0 in 80 columns mode. This mode is slightly altered to display 26.5 lines and 4 colors. The assembler in Compass also uses some VRAM, but this amount will never exceed the address space from #00000 up to #07FFF.

1.4 Booting Compass

The version of Compass that you'll find on the disk is a COM file which has to be launched using MSX-DOS. The two files that make up MSX-DOS are:

- MSXDOS.SYS and COMMAND.COM for MSX-DOS 1.xx
- MSXDOS2.SYS and COMMAND2.COM for MSX-DOS 2.xx

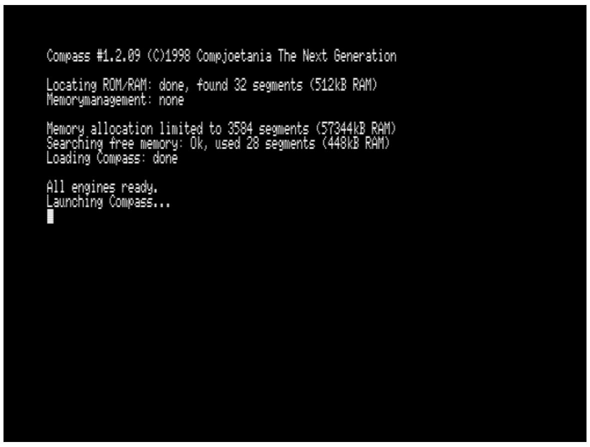
These files are copyrighted and therefore not distributed with the original package. You can copy the files COMPASS.COM and COMPASS.DAT from the Compass disk to a disk containing these MSX-DOS files, or just launch MSX-DOS and swap the disks.
Alternatively you can copy the Compass files to a directory on your hard disk.
When you have started MSX-DOS just type:

```
A>COMPASS [RETURN]
```

Now that you have started Compass you will see a nice graphical logo appear on your screen.
This logo is made in screen 5 causing the first page of screen 5 to be erased.

Launching Compass will take longer if you leave this logo enabled. In the main install menu there is an option which allows you to disable this logo!

After the logo, Compass will perform some hardware checks on your computer.
If your computer crashes here, then there is something wrong with your hardware. (for example: random memory errors when working on a 7MHz MSX2. These hardware problems are very difficult to trace.)
If everything went fine, you'll see the amount of mapped memory Compass has detected and what kind of memory management will be used.
Compass can run in four memory management conditions:



```
Compass #1.2.09 (C)1998 Compjoetania The Next Generation

Locating ROM/RAM: done, found 32 segments (512kB RAM)
Memorymanagement: none

Memory allocation limited to 3584 segments (57344kB RAM)
Searching free memory: Ok, used 28 segments (448kB RAM)
Loading Compass: done

All engines ready.
Launching Compass...
```

```
DOS1        no memory management, Compass will use its own routines
DOS1+MemMan MemMan will take care of Compass' memory management. Due to some
            unknown problems (probably MemMan) this setting will cause the
            computer to crash. So we do not recommend to use MemMan in DOS1
            environment.
DOS2        memory management performed by the DOS2 mapper support routines
DOS2+MemMan memory management performed by MemMan
```

If you use MemMan, we recommend to use the latest version of MemMan, version 2.42.

If it is the first time you start Compass, the program will search for available memory. Compass needs at least five free pages of mapper ram. And two of them should be situated in the Primary Mapper. Four of them may not be segments in the TPA because the Compass-program-data will be stored there.
If one of these conditions is not met, the installation of Compass will abort, otherwise the program will allocate a maximum of 1536kB (or 96 mapper pages) and make this memory unreachable for other programs. Of course if you don't use any memory manager, this memory is not unreachable for other programs and your Compass program and/or data may be overwritten by these programs. Compass will preferably allocate memory in other mappers than the Primary Mapper. This might help to prevent the previous problem, although this will make the program somewhat slower.

In some cases it can be useful not to use all available memory. You can limit the memory allocation of Compass by pressing the SHIFT key when you start Compass. Currently this limitation is set to 9 segments. In future versions of Compass this number will be changeable.

If you have performed a 'Main install' (see chapter 10) Compass will try to reinstall your saved memory configuration. If all used segments in this memory configuration are not free, or when the Primary Mapper slot has changed (for example: when you've placed an extra memory mapper into your MSX), Compass will try to allocate segments described as above.

Finally the Compass program is loaded from (hard)disk and is launched. On fast MSX-systems startup messages may disappear too fast. Hold down STOP during loading if you want to pause these messages. Press this key again to continue.

It is also possible to load an assembler file (ASM) from the command line.
Example:

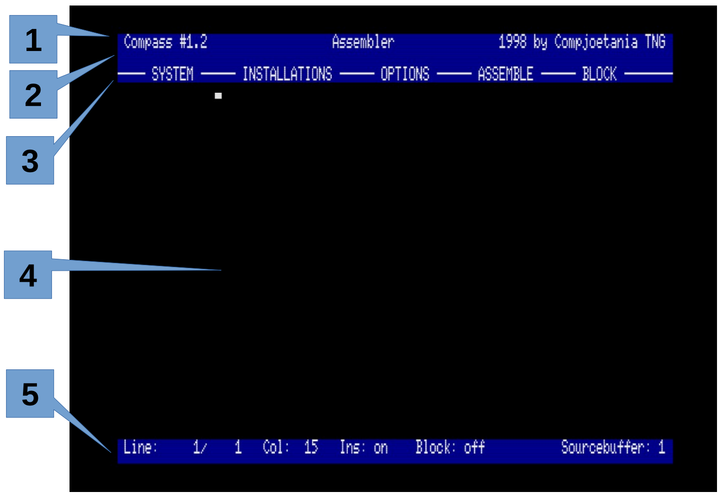
```
        A>COMPASS CHAIN.ASM
```

The file CHAIN.ASM will be loaded when Compass is launched.
If the file specified isn’t saved as an assembler file the loading will silently fail and will not display an error!

Notice:
Compass frequently uses Kbuf (#F41F: 318 bytes long). Make sure this part of your memory is free and that your programs and/or TSRs don't affect this memory. The printer buffer ‘PB.TSR’ for instance can't be used because of this.
Also some Compass code is stored in PLAY- queues for channel B and C.
(#F9F5-#FAF4) Make sure these queues are not overwritten by other programs or by executing a PLAY statement in BASIC for channel B or C.

2 General

2.1 General screen layout



```
Compass #1.2              Assembler          1998 by Compjoetania TNG
── SYSTEM ── INSTALLATIONS ── OPTIONS ── ASSEMBLE ── BLOCK ──

Line:    1/    1  Col:  15  Ins: on   Block: off        Sourcebuffer: 1
```

All parts of Compass use a common screen layout to have a general look and feel throughout the entire program.

1. Status bar:

The upper line of the screen always contains the status bar. On the left of this bar you will find the name Compass followed by the main version number. This number will change when a new main version is distributed.
(The complete version number can be obtained by displaying the COMPASS.COM or COMPASS.DAT file in some text-viewer.)

On the right side the announcement '1998 Compjoetania TNG'. Between these two messages you'll find the name of the currently selected program part of Compass.

2. The empty line:

It's just here for ease of survey. Don't expect any messages here.

3. Menu bar:

This menu bar is situated below the empty line. This bar will display the available menus of the currently active program part. It can contain a maximum of five menus. These can be reached using the function keys [F1] up to [F5].

4. Desktop:

This occupies most of the screen. Here the real action takes place.

5. Function bar:

Below the desktop you'll find a status bar. This bar is only present in the assembler and contains important information about the current source and settings. In the other parts of Compass this function bar is suppressed and the extra lines are used to extend the desktop.

2.2 Controls

Almost all functions in Compass can be selected by means of the keyboard, the mouse or the so-called shortcuts.
The menu bar contains the menus which can be activated using the function keys or second mouse button (if mouse support is activated). When a menu is activated a drop-down list will appear. You can select an other menu by clicking the appropriate function key or by using the cursor left and cursor right keys to gain access to the other menus. Use the cursor keys up and down to selected an item in the menu.
Your current choice will be indicated in a bar that uses the alternative colors. Use the space-bar to activate a choice.

pressing [ESC] will leave the menus and bring you back in the active program part.

For your convenience a standard order is used throughout all parts of Compass.
The function key [F1] will always activate the SYSTEM menu (see chapter 2.3) and the second menu (use [F2]) will always be the INSTALLATIONS menu. The contents of these menus will of course be slightly altered depending on the active part when the menu was summoned. The menus hidden under [F3] up to [F5] are different for each part and may even be absent.

Mouse control

A great effort has gone into enabling mouse support for Compass. A lot of options can be reached using the mouse without the need for keyboard interaction. Of course a lot of actions, like searching, need keyboard input, but a lot can be done without it.

If you're on the desktop you can use the right mouse button to reactivate the last used menu. Use the left button to activate an item. The mouse can now be used to walk through all options and menus.
Instead of pressing [ESC] you can use the right mouse button in most of the cases.
To move around with the cursor on the desktop, you should hold down the left mouse button.

Remark: mouse control is disabled by default. (see 'Main Install' (chapter 5.2.1))

Shortcuts

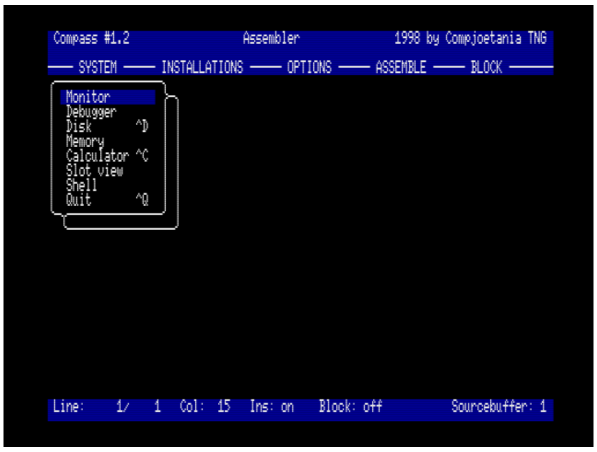
Shortcuts are key combinations which allow you to reach certain options in a very fast way without using the regular menu selection which can be time consuming if used a lot. In Compass a shortcut consists in most of the cases of the [CTRL] key combined with an other key. All the shortcuts are listed in Appendix A: Shortcuts.
Most shortcuts are mentioned after the option in the menu. For example: in the SYSTEM menu the disk option will be followed by ^D indicating that [CTRL] and [D] pressed together is the shortcut for the disk menu.

An other easy shortcut to use is the [STOP] key which switches you from one program part to another one. This in the order : assembler, monitor, debugger, assembler, monitor, ... etc.

2.3 System menu

This menu can always be reached by pressing [F1]. This menu differs very slightly from program part to program part. The options of this menu are listed below accompanied by the chapter number which deals with the topic.

Assembler	chapter 3
Monitor	chapter 4
Debugger	chapter 5
Disk	chapter 6
Memory	chapter 7
Calculator	chapter 8
Slot view	chapter 9
Shell	see below
Quit	see below



system menu when launched from the assembler

The options SHELL and QUIT are closely related.

2.3.1 Shell

The shortcut for SHELL is [SHIFT]+[ESC].

With SHELL you can leave the Compass program and go back to the shell from where you've last activated Compass. You can return to Compass by typing CMD COMPASS [RETURN] from the MSX-BASIC prompt or by pressing [SHIFT]+[ESC] (DOS and MSX-BASIC)

If you return to Compass, all settings are still preserved and if everything went well you can just continue were you left. Nevertheless certain precautions should to be taken. When using DOS2 and/or MemMan the memory blocks used by Compass are reserved by these respective memory managers. So if all other programs, started during this 'shell', use these memory managers., there is no problem.
If you use a program that does its own memory management, then you can easily destroy the data used by Compass. If this happens your source can be destroyed or Compass can be become unstable. A simple trick that most of the time works, is placing all the important memory blocks (sources) in the highly numbered blocks and the less important (label buffers) in the lower memory blocks.

2.3.2 Quit

The shortcut for QUIT is [CTRL]+[Q].

QUIT will terminate the Compass program completely. All used memory will be freed and you will return to the last activated shell.
Make sure you save your work before quitting! Otherwise there is no way to recover it.

3 Assembler

After you have chosen the option ASSEMBLER from the SYSTEM menu you will end up in the editor. The assembler consists of a very extensive editor and the actual assembler.

Programming machine language directly in processor comprehensible byte code is a very tedious job. When one instruction is inserted, all absolute addresses in the code have to be recalculated and to do this yourself isn't very enjoyable.
Besides, these instructions are made up of numbers, which aren't as easy to remember as carefully chosen abbreviations. The combination of these (hexadecimal) numbers are called opcodes. For each opcode an easy to remember word/sentence was standardized. These words are called mnemonics. Each mnemonic represents an opcode and vice versa. The editor is used to write a program with these mnemonics. The machine language written in mnemonics is called an assembler source.
When this source is assembled it is translated into machine code instructions.
Each time you change something in the source, the source should be reassembled before you can run the new program because all used addresses will probably need recalculation.

Labels

Most powerful feature of an assembler is the use of labels. These labels can be used for different purposes and they help to make the sources more comprehensible. They are mostly used to give constants a name and to help calculating addresses when assembling. The length of these labels in Compass can vary between 6 and 20 characters. The following characters are allowed in a label name:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789_-!?
```

There are a few restrictions. A label may not start with a digit. Neither reserved words may be used as a label. An overview of these reserved words can be found in Appendix C: Reserved words Programmers are used to place a colon behind a label when this label is declared, but there is no need to do this in Compass. If a label ends with two colons then this label is declared public (see chapter 14).

3.1 Assembler screen and controls

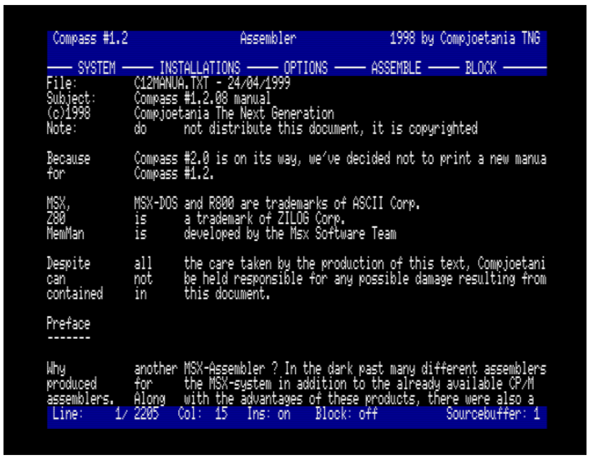
The assembler screen is made up of a status bar, a menu bar, the desktop and a function bar. The menu bar has five different menus, respectively the SYSTEM menu, the INSTALLATIONS menu, the OPTIONS menu, the ASSEMBLE menu and finally the BLOCK menu.
The desktop can be found below the menu bar, which is a text editor containing the sources.
The function bar can be found at the bottom of the screen, displaying the line numbers, INSert on/off, buffer used by BLOCK and the number of the current source buffer.

The editor supports the mouse and most of the functions can be activated using shortcuts. For a list of these shortcuts we refer to Appendix A: Shortcuts.
Of course the regular keys like [BS], [DEL], [TAB] etc. work like one expects them to do.
Pressing [GRAPH]+cursor keys can be used to change casing state of letters.

The editor is line oriented. The line is stored in memory by pressing [RET] and is displayed according to some simple guidelines.

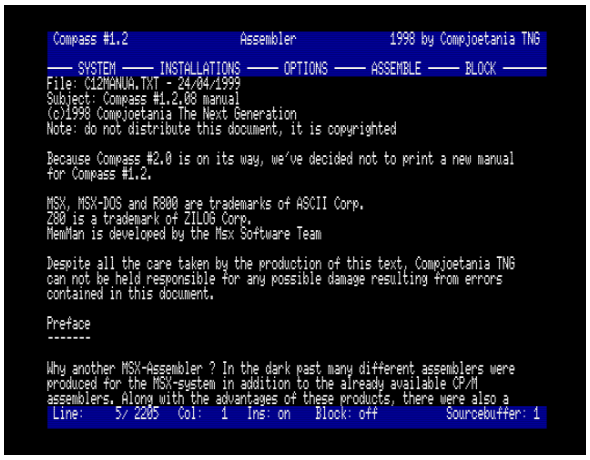
If AUTO ALIGN is turned on:
Labels are placed at the left margin, the instruction is placed at the first tab position and instruction parameters are placed at the second and third tab positions. All characters behind a semi-colon are ignored and are positioned behind the last used tab position. This is used to enter remarks about your program. The cursor is placed at the first tab position. on the next line. An empty line is inserted if the RET INSERT is on.

If AUTO ALIGN is turned off:
no repositioning of the entered text will take place, the cursor will be placed at the beginning of the next line. This is useful to edit/load normal text documents.



```
 Compass #1.2                Assembler            1998 by Compjoetania TNG
─── SYSTEM ─── INSTALLATIONS ─── OPTIONS ─── ASSEMBLE ─── BLOCK ────
File:        C12MANUA.TXT - 24/04/1999
Subject:     Compass #1.2.08 manual
(c)1998      Compjoetania The Next Generation
Note:        do      not distribute this document, it is copyrighted

Because      Compass #2.0 is on its way, we've decided not to print a new manua
for          Compass #1.2.

MSX,         MSX-DOS and R800 are trademarks of ASCII Corp.
Z80          is      a trademark of ZILOG Corp.
MemMan       is      developed by the Msx Software Team

Despite      all     the care taken by the production of this text, Compjoetani
can          not     be held responsible for any possible damage resulting from
contained    in      this document.

Preface
-------

Why          another MSX-Assembler ? In the dark past many different assemblers
produced     for     the MSX-system in addition to the already available CP/M
assemblers.  Along   with the advantages of these products, there were also a
 Line:     1/ 2205   Col:  15   Ins: on    Block: off           Sourcebuffer: 1
```

auto align on when loading a regular text file



```
 Compass #1.2                Assembler            1998 by Compjoetania TNG
─── SYSTEM ─── INSTALLATIONS ─── OPTIONS ─── ASSEMBLE ─── BLOCK ────
File: C12MANUA.TXT - 24/04/1999
Subject: Compass #1.2.08 manual
(c)1998 Compjoetania The Next Generation
Note: do not distribute this document, it is copyrighted

Because Compass #2.0 is on its way, we've decided not to print a new manual
for Compass #1.2.

MSX, MSX-DOS and R800 are trademarks of ASCII Corp.
Z80 is a trademark of ZILOG Corp.
MemMan is developed by the Msx Software Team

Despite all the care taken by the production of this text, Compjoetania TNG
can not be held responsible for any possible damage resulting from errors
contained in this document.

Preface
-------

Why another MSX-Assembler ? In the dark past many different assemblers were
produced for the MSX-system in addition to the already available CP/M
assemblers. Along with the advantages of these products, there were also a
 Line:     5/ 2205   Col:   1   Ins: on    Block: off           Sourcebuffer: 1
```

auto align off to load the file correctly

3.2 The assembler menus

The menu bar of the assembler consists of five menus

```
System         see chapter 2.3
Installations  see chapter 3.2.1
Options        see chapter 3.2.2
Assemble       see chapter 3.2.3
Block          see chapter 3.2.4
```

Most of these options can be reached by using the shortcuts, see Appendix A: Shortcuts

3.2.1 Installations menu

Some assembler settings can be modified in this menu, which you can reach by pressing [F2].

SOURCEBUFFER:

Use this option to switch between the different available source buffers.
Compass can use a maximum of four source buffers., each containing a maximum of 256 kB text. This allows you to work with four sources at the same time! Or you can use some source buffers. to hold a text with technical reference material etc.

The number and size of the source buffers. can be controlled using the MEMORY option in the SYSTEM menu. In the right lower corner of the screen the currently active buffer-number is displayed.

CPU:

If you're running Compass on a Turbo-R you can use this option to choose the desired CPU-mode: Z80-mode, R800 ROM mode and R800 DRAM mode. There isn't any difference in speed between the R800 ROM and R800 DRAM mode since Compass doesn't use bios calls. Therefore we suggest to choose for the R800 ROM mode since the DRAM mode uses 64kB of memory.

Note: on a MSX2 this will always display Z80 and no other selections are available of course.

LABEL LENGTH:

The length of the labels can vary from 6 to 20 characters. Use cursor keys to alter the value when this option is chosen. Use [ESC] to cancel. See also the assembler directive .LABEL (see Assembler directives chapter 3.3.3)

RET INSERT:

When this option is turned on, a new blank line will be inserted automatically each time [RET] is pressed. If the cursor is at the end of a line, an empty line is inserted after the current line. Otherwise the empty line is inserted before the current line. In both cases the cursor is placed at the new line.

AUTO ALIGN:

Turn this option off if you want to read and handle non source ASCII files.
For more information see chapter 3.2.1 .

UPPER CASE:

This makes the assembler case sensitive if turned off. This only affects label- and macro-names.
Notice: no effects are visible in the editor!
If this is turned off, labels like Print (some printing routine) and prInt (printer initialization) are completely different labels. Be aware of the fact that using this kind of labels can be very confusing later on, when you have to reread some old code or someone else has to maintain your code.

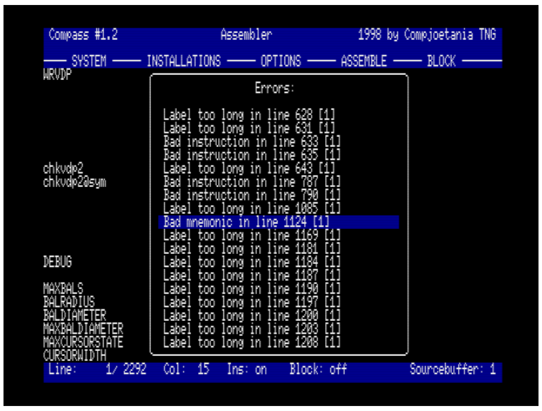
MAIN INSTALL:

This affects the general setting for Compass. For more info see chapter 5.2.1.

3.2.2 Options menu

In this menu, accessible using [F3] you'll find some options that are very useful when editing a source.

SHOW ERRORS:



```
Compass #1.2             Assembler          1998 by Compjoetania TNG
── SYSTEM ── INSTALLATIONS ── OPTIONS ── ASSEMBLE ── BLOCK ──
WRVDP
                          Errors:

                 Label too long in line 628 [1]
                 Label too long in line 631 [1]
                 Bad instruction in line 633 [1]
                 Bad instruction in line 635 [1]
chkvdp2          Label too long in line 643 [1]
chkvdp2@sym      Bad instruction in line 787 [1]
                 Bad instruction in line 790 [1]
                 Label too long in line 1085 [1]
                 Bad mnemonic in line 1124 [1]
                 Label too long in line 1169 [1]
                 Label too long in line 1181 [1]
DEBUG            Label too long in line 1184 [1]
                 Label too long in line 1187 [1]
MAXBALS          Label too long in line 1190 [1]
BALRADIUS        Label too long in line 1197 [1]
BALDIAMETER      Label too long in line 1200 [1]
MAXBALDIAMETER   Label too long in line 1203 [1]
MAXCURSORSTATE   Label too long in line 1208 [1]
CURSORWIDTH
Line:    1/ 2292   Col:  15   Ins: on   Block: off       Sourcebuffer: 1
```

If during assembling errors are encountered all these errors are placed in a single error list. This option displays the error list and enables you to select an error. When you press [SPACE] the editor will be placed on the line number containing the error.
If the line number doesn't exist any more a beep will be heard and the error list will stay activated.
The error list stays in memory until you assemble the source again, start the program or do some disk operations.
If you press [ESC] you go back to the editor, when needed you can return to the error list as often as you want. When inserting and removing lines in the editor, the line numbers in the error list will be adjusted automatically.
You can find an overview of all the possible errors and their meaning in Appendix B: Error messages .

SHOW LABELS:

When you assemble a source all the label names and their value will be placed in the label list. When displaying this list using the SHOW LABELS option, all labels will be displayed in alphabetical sequence. Use the cursors to scroll through the labels. If you press [SPACE] you will jump to to line number where the label is defined.

SEARCH/REPLACE:

Use this to search for certain words and if necessary to replace them by other words. [CTRL] + [Z] is the most easy way to reach this option. When activated you will be asked for the string to search for. This field will contain the value of the previous search.
When [ENTER] is pressed you can, if needed, type the replacement string. Leave this field empty if no replacement is needed. Searching forward or backward is possible. Case sensitivity can be altered. When the search is started there are two possibilities. If the string is found then the cursor will go to the line where the string is found. Otherwise the error message "String not found!" will appear. Use SEARCH NEXT to find the next occurrence of the string.

SEARCH NEXT:

If you didn't get an error message you can start a new search/replace for the same string. The shortcut is [CTRL] + [N].

JUMP TO LINE:

Allows you to jump to a given line. After selecting this option a window will appear, allowing you to enter the line number. When [RET] is pressed the cursor is immediately placed on this line. The cursor will be placed on the last line of the source if the line number doesn't exist.

DELETE SOURCE:

This removes the source in the current source buffer. This option will ask for confirmation. After you have confirmed the source will be gone and there will be no way to recover it!
So make sure you have saved the source if you think that you'll need it again!!

3.2.3 Assemble menu

You'll find everything you need to assemble your source in this menu, accessible using [F4].

ASSEMBLE:

Activate this to assemble the source in the current source buffer. Assembling in Compass is a so-called two-pass-assembling. First the entire source is evaluated to find the values of all the labels and in a second pass the source is assembled to memory.
Errors will be shown after the first, respectively second, pass. If an error is found in the first pass, the assembler will not start the second pass. Use the ERROR-menu to locate the errors in the source, see chapter 3.2.2 and Appendix B: Error messages section Assembler errors . If all went well the begin and end address will be shown.

ASSEMBLE TO DISK:

Where the first option assembles to memory, with the possibility to change the destination on the fly with the ORG directive, this option just writes the bytes generated during assembling sequentially to a file. You can choose where and with which name the source has to be assembled. The option save and load are replaced by assemble. Use this option to start the actual assembling. Assembling directly to disk enables you to write programs that are able to reallocate themselves on execution. For example:

```
start   equ     #8000
        org     #0100
        ld      hl,program ;move program
        ld      de,start
        ld      bc,programend-start
        ldir
        jp      start
program
        org     start
...     ;reallocateable program comes here
programend
```

ASSEMBLE TO TSR:

Assemble your source directly to a Terminate and Stay Resident program, to be used with MemMan.
In this case assembling will take place in three passes. The actual code will be written to disk during the third pass.
For more info on TSR's we suggest that you get a copy of the TSR development kit of MST. See chapter 13 for more info on the .TSRHOOKS directive, needed for assembling a TSR.

ASSEMBLE TO REL:

This will compile the code into a relocatable file format. This file format can be placed anywhere in memory or glued together with other programs using a linker.

This can be very useful if you have a greater project. You can split your source in different small modules. If you make a modification to one of these modules, only this small module has to be reassembled instead of the entire source.
For the moment it is still necessary to use an external linker program.

REGISTERS:

This option is used to alter the registers of the Z80/R800 before you use the GO option to execute a program. The contents of the registers when the program is finished are also stored so that you can use this option to examine to values after execution.

For more info on how to change the values, see the chapter about the disassembler; this window uses the same techniques as discussed there.
For a more thorough examination of the code as it is processed by the MSX, you would be better off with the debugger itself.

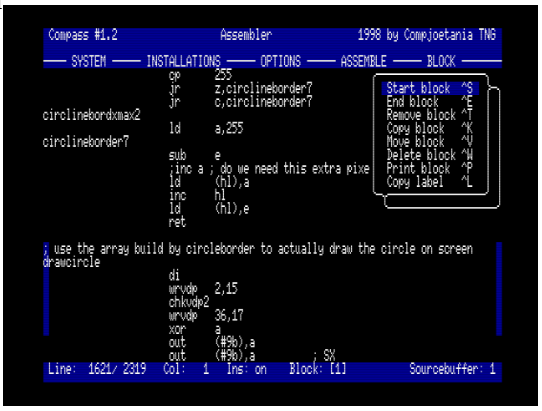
GO:

Use this to execute a program somewhere in memory. It uses the register/interrupt settings as set with the REGISTERS menu. When activated you will be asked for an address to 'call to'. Before the code is actually executed the screen is placed in regular 24 line screen 0 mode and cleared.
When the program returns Compass will reinitialize the VDP in 'Compass mode'. Be aware: your MSX may crash if the program to run is not stable! So make sure you have saved the source before trying this option!

3.2.4 Block menu

You'll find all available block functions of the editor in this menu, reachable by pressing F5. You can select only one block in one of the four source buffers at the same time. Whether a block is selected or not is displayed at the function bar: the number of the source buffer in which the block is selected is displayed here.

You can recognize a selected block by its coloured borders: the first and last position on these selected lines will have the colour of the menu bar.



```
Compass #1.2                  Assembler              1998 by Compjoetania TNG
── SYSTEM ── INSTALLATIONS ── OPTIONS ── ASSEMBLE ── BLOCK ──
                  cp      255
                  jr      z,circlineborder7
                  jr      c,circlineborder7
circlinebordxmax2
                  ld      a,255
circlineborder7
                  sub     e
                  ;inc a ; do we need this extra pixe
                  ld      (hl),a
                  inc     hl
                  ld      (hl),e
                  ret

; use the array build by circleborder to actually draw the circle on screen
drawcircle
                  di
                  wrvdp   2,15
                  chkvdp2
                  wrvdp   36,17
                  xor     a
                  out     (#9b),a
                  out     (#9b),a          ; SX
Line:  1621/ 2319   Col:   1   Ins: on    Block: [1]          Sourcebuffer: 1
```

A selected block and the block menu

START BLOCK

To select a desired block, you'll need to mark the begin and the end of the block. Use this option to mark the begin when you're on the desired line.

END BLOCK

Use this option to mark the end when you're on the desired line.

REMOVE BLOCK

Use this option to deselect a selected block. The text itself is not removed, only the begin and end-mark!

COPY BLOCK

You can copy the selected block with this option to the line you're on. The original block is not removed. Of course you can't copy a block to somewhere in the selected block. You can also copy between different source buffers with this option. You can copy the selected block as many times as your amount of memory will allow you.

MOVE BLOCK

You can move the selected block with this option to the line you're on. The original block is deleted. Of course you can't move a block to somewhere in the selected block. You can also move text between different source buffers with this option.

DELETE BLOCK

Use this option to delete the selected block.
Be careful, because a deleted block can't be restored!

PRINT BLOCK

The selected block will be sent to the printer. If your printer is not ready, an error message will appear.

COPY LABEL

To use this option you should have selected a block beginning with a label. If you activate this option the label will be copied to the cursor's current position.

3.3 Special commands

Compass' assembler contains a lot of extra commands for several purposes:

3.3.1 Exceptions

In addition to all standard Zilog/Mostek mnemonics some other mnemonics are allowed:

```
PUSH reg1,reg2,reg3,...  ;this is assembled as separate PUSH instructions
POP reg1,reg2,reg3,...   ;this is assembled as separate POP instructions

EX AF,AF   ;this is supported because of compatibility with other assemblers
EX AF,AF'  ;standard Zilog mnemonic

LD A,""      ;the same as LD A,0
LD HL,"KL"  ;the same as LD HL,#4B4C

LD B  ;the same as LD A,B This works for all mnemonics with register A

DJNZ $-2  ;dollar-sign is equal to the current address

LD A,%11 01 11 10  ;spaces are allowed between binary digits
```

Following is not allowed:

```
LD A,"""   ;this should be done with LD A,'"'
LD A,char  ;only character with an ASCII code below 128 are allowed
```

3.3.2 The R800 processor

A Turbo-R is equipped with a R800 processor in addition to the Z80. Almost all mnemonics on this R800 were renamed, but the opcodes have remained. Compass only recognizes the old Z80 mnemonics, but does recognize the new mnemonics of the new instructions. You're allowed to use these new instructions on MSX2, but they won't work on Z80.

There are four new 8-bit instructions for multiplication. HL contains the 16bit result:

```
MULUB A,B ;opcode: ED C1
MULUB A,C ;opcode: ED CD
MULUB A,D ;opcode: ED D5
MULUB A,E ;opcode: ED DD
```

There are two new 16-bit instructions for multiplication. DE-HL contains the 32bit result:

```
    MULUW HL,BC ;opcode: ED C3
    MULUW HL,SP ;opcode: ED F3
```

There is a new instruction to read a port to the Flag-register. Only S and Z are set according to the incoming data. H, N and P/V are reset.

```
    IN F,(C) ;opcode: ED 70
```

Using half index-registers on R800 is possible. These instructions were also available on Z80, but not officially. They are called IXh, IXl, IYh, IYl.

3.3.3 Assembler directives

These are special commands needed for correct assembling, also called pseudo-instructions.

`ORG`

`Syntax: ORG address`

With ORG you can select on which address the program must be compiled. That's why you have to put in at the beginning of a program. If no ORG directive is used, the program will be assembled at address #0100.

`EQU (equals)`

`Syntax: label EQU value`

Use this to define your constants.
Example: CHGET EQU #009F

`END (End of assembly)`

`Syntax: END`

Use this directive to indicate that assembling must be terminated here. There's no need to put this directives at the end of your source. It's only useful to stop assembling somewhere in the middle of source, for instance when not all labels are defined yet and you want to compile the first part of your source.

`DEFB / DB (Define Byte)`

`Syntax: DB value,value,text,...`

Use this directive to poke some 8bit values or text in your code.
Example: DB "This is great!",0

You can use numerical operations on the last character in a string:
Example: DB "This too!"+128 ;bit 7 of the last character (!) is set

`DEFM / DM`

Is exactly the same as DB. This directive was built in for compatibility.

`DEFC / DC`

`Syntax: DEFC string`

Is the same as DB, but bit 7 of the last character in the string is set.
Useful to recognize the end of a text field in memory.
Example: DC "Hallo"   ;bit 7 of the 'o' is set

Also numerical operations on the last character are allowed.
Example: DC "TNI"-2   ;same as "TNG" with bit 7 of G set.

```
DEFS / DS (Define Storage)
Syntax: DS number of bytes [,byte to fill with]
```

With this directive you can create empty memory areas in your program. Useful for buffers, etc. If the 'byte to fill with' is omitted, the area will be filled with zero.
Example: DS 100,"C"   ;area with 100 bytes "C"
        DS 4*20,13   ;area with 80 bytes #0D

```
DEFW / DW (Define Word)
Syntax: DW 16bitvalue,16bitvalue,...
```

Same as DB, but DW puts 16bit values in your code.
Example: DW #babe,label,20*5

```
TSRHOOKS
Syntax: TSRHOOKS
```

You'll need to put this directive in front of the hooks used by your TSR. (At the end of your TSR)
More info: see chapter 13.

```
.LABEL
Syntax: .LABEL length
```

You can tell the assembler how long the used labels can be in your source. So you won't need to change this setting in the INSTALLATIONS menu when you work with several labellenghts. Of course this directive has to be placed at the beginning of your source.

```
.UPPER
Syntax: .UPPER on/off
```

You can tell the assembler whether all text should be uppercased or not before assembling starts. So you won't need to change this setting in the INSTALLATIONS menu when you work with several types of casing.
Remark: your source itself will not change to uppercased characters.

```
BREAKP
Syntax: BREAKP
```

If the assembler encounters this directive, the current address will be parsed to the Debugger's breakpoint table. Easy to debug your own source.
Remark: Your previous breakpoint settings will be erased.

INCLUDE
```
Syntax: INCLUDE buffer[,filename]
```

This directive gives you the possibility to add external sources to your current source. This other source should be in source buffer 'buffer'. If you also enter a filename, the source will be loaded first in source buffer 'buffer'.

MACRO / ENDM / DEFL

These directives are necessary for working with macro's. See chapter 11.

IF / COND /ELSE / ENDIF / ENDC

These directives are necessary for working with conditional assembly. See chapter 12.

CSEG / DSEG / ASEG / PUBLIC / EXTRN / .PHASE / .DEPHASE

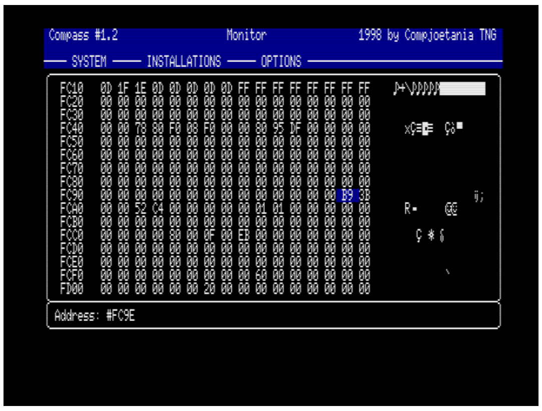
These directives are necessary for working with Relocateable files. See chapter 14.

4 Monitor

After you have chosen the option Monitor from the system menu you will be faced with a new screen layout indicating that you have entered the monitor.
This monitor is completely real-time, meaning that the information on screen is updated constantly to reflected possible changes in the memory as they occur.
Since this enables you to keep track of which bytes are changed in memory this is a perfect way to get an understanding/insight of what a program does or why a certain error occurs in your program.

4.1 Monitor screen and controls

The monitor screens is made up by a status bar, a menu bar. and the desktop. The menu bar has three different menus, respectively the SYSTEM menu, the INSTALLATIONS menu and finally the OPTIONS menu. Below the menu bar. the desktop can be found, which consist of, on the left side a hexadecimal dump and on the right side, an ASCII dump.



```
Compass #1.2                Monitor            1998 by Compjoetania TNG
FC10   0D 1F 1E 0D 0D 0D 0D 0D FF FF FF FF FF FF FF FF
FC20   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC30   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC40   00 00 78 80 F0 08 F0 00 00 80 95 DF 00 00 00 00
FC50   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC60   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC70   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC80   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FC90   00 00 00 00 00 00 00 00 00 00 00 00 00 00 B9 3B
FCA0   00 00 52 C4 00 00 00 00 00 01 01 00 00 00 00 00
FCB0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FCC0   00 00 00 00 80 00 0F 00 EB 00 00 00 00 00 00 00
FCD0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FCE0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FCF0   00 00 00 00 00 00 00 00 00 60 00 00 00 00 00 00
FD00   00 00 00 00 00 00 20 00 00 00 00 00 00 00 00 00
Address: #FC9E
```

Monitoring the JIFFY system variable

The monitor supports the mouse and most of the functions can be activated using shortcuts. For a list of the shortcuts we refer to appendix A. You can use the cursor keys to scroll trough the monitor. When you're using the monitor you can alternate between the ASCII and hexadecimal dump by pressing [TAB].

In the hexadecimal layout you can alter the value of the currently selected address by pressing one of the number keys [0] to [9] and the [A]-[F] keys. The pressed key will be displayed and the program will wait for the rest of the number to be entered. At this point you can press [ESC] if you didn't want to change the value. In this case the old value will be restored. If you have entered just one key so far and press the cursor keys then the value stored in the address will be changed with the just pressed figure and afterwards the cursor movement will be processed. If you entered a second digit the new value will be stored and the cursor will continue to the next address.

In the ASCII layout you can change the current character be pressing any key.
The value will be altered to the key value you've pressed and the cursor will automatically move to the next position. If you use the [BS] key then the current address will be set to zero and the cursor will be moved to the previous address.

By pressing the [SELECT] key you can rapidly switch between the monitor and debugger, making it possible to quickly adapt values or keep track of larger memory blocks when debugging. For the other available shortcuts, please have a look at appendix A.

4.2 The monitor menus

The menu balk in the monitor contains 3 different menus:

```
        System         see chapter 2.3
        Installations  see chapter 4.2.1
        Options        see chapter 4.2.2
```

4.2.1 Installations menu

A number of settings which are applicable to the monitor can be set in this menu, which you can reach by pressing [F2].

ADDRESS

Use this option to set the current address which you'd like to monitor. When selected a pop menu window appears so you can enter an address. This address can be explicit like #F3DE or 32894 but you can also use labels or calculate an address on the fly like label+4*7. You can leave this window by pressing [ESC] if you didn't want to change the address.

CPU

If you're running Compass on a Turbo-R you can use this option to switch between CPU mode: Z80, R800 ROM and R800 DRAM mode. Of course only Z80 mode is present on non-Turbo-R computers.

CHAINING

It is possible to link or 'chain' the monitor address to some of the addresses used in the debugger. In that case the monitor address and the specific debugger address will be kept equal.

MAIN INSTALL

Use this to change Compass' global settings. For more info read chapter 10.

4.2.2 Options Menu

In this menu, which you can reach by pressing [F3], are methods available to search and alter memory content.

SEARCH

This option always you to search the visible memory for sequences of bytes. A window appears so you can enter a start and ending address for the search. As third you are asked for the byte sequence to search for. Here a few examples how to enter some specific search criteria.

```
"TEST"  ; search  'T','E','S','T'
"ZOEK",0,label  ; search 'Z','O','E','K',0,label
```

When you press enter there are two possible effects:

- you return to the monitor and the address is changed to the first occurrence of the given byte sequence.
- you return to the monitor and the message *"String not found !"* is displayed indicating that the given sequence isn't found in the indicated memory block.

SEARCH NEXT

Search for the next occurrence of the previous searched-for byte-sequence.
Of course if you got the error *"String not found !"* than this option hasn't got any effect.

FILL

This option gives you the opportunity to fill a block of memory with a given value. Again a window will appear so that you can enter the beginning and ending of the region in memory. Next you'll have to enter the value that should be used to fill the memory.
If this value exceeds 255 then this memory will automatically be filled with 16 bit values. If you enter a number that's smaller or equal to 255 then there will be explicitly asked if this value should be treated as a 16 or 8 bit value. When all values are entered the filling will immediately take place.
You can press [ESC] at any time to leave this option.

COPY

This is, like the name suggest, for copying a block of memory to another address. First you'll be prompted to enter the beginning and the ending of the source. As third parameter you'll need to enter the destination address.
You can press [ESC] at any time to leave this option.

COMPARE

This makes it possible to compare two chunks of memory. Enter begin and end address of the first block; finally you are asked for the beginning of the second block, which shall be compared to the just defined first block.
If no differences are found you get back in the monitor without any message, otherwise the message *"Not equal !"* will be displayed and the monitor will be set to the address where the first difference is encountered.

POKE

Just like the BASIC command this option is used to change memory contents at a specified address. Although this can be done by direct entering new values at the cursor position, this option has definitely some advantages.
If the value entered exceeds 255 then this value will be automatically treated as a 16 bit value. If you entered a number that's smaller or equal to 255 then there will be explicitly asked if this value should be handled as a 16 or 8 bit value.
This option has also the advantage that you can write bytes to specific memory locations while some other memory part can be watched. Useful for writing to the DOS2 rompage selector address, SCC rompage selector addresses, ...

PEEK

Allows you to have a quick peek at an address without changing the monitor address. The contents of the address will be displayed both 8 and 16 bit.

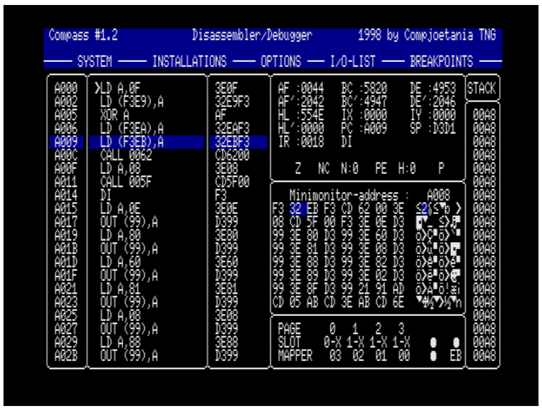
PRINT

Use this if you want to have a hardcopy of the memory dump. Again you enter begin and end of the desired chunk. Once this is done the block of memory will be printed using the same layout as appear on the screen.

5 Debugger

After you have chosen the option Debugger from the system menu you will end up in the debugger. This debugger can also be used as a disassembler.
This debugger allows you to have an easy and comfortable way of finding bugs in a program or just to monitor the program flow to get a better understanding of a program.

5.1 Debugger screen and controls



```
Compass #1.2          Disassembler/Debugger        1998 by Compjoetania TNG
 ── SYSTEM ── INSTALLATIONS ── OPTIONS ── I/O-LIST ── BREAKPOINTS ──
A000  >LD A,0F       3E0F     AF :0044  BC :5820  DE :4953   STACK
A002   LD (F3E9),A   32E9F3   AF':2042  BC':4947  DE':2046   00A8
A005   XOR A         AF       HL :554E  IX :0000  IY :0000   00A8
A006   LD (F3EA),A   32EAF3   HL':0000  PC :A009  SP :D3D1   00A8
A009   LD (F3EB),A   32EBF3   IR :0018  DI                   00A8
A00C   CALL 0062     CD6200                                  00A8
A00F   LD A,08       3E08       Z  NC  N:0  PE  H:0   P      00A8
A011   CALL 005F     CD5F00                                  00A8
A014   DI            F3         Minimonitor-address :  A008  00A8
A015   LD A,0E       3E0E     F3 32 EB F3 CD 62 00 3E        00A8
A017   OUT (99),A    D399     08 CD 5F 00 F3 3E 0E D3        00A8
A019   LD A,80       3E80     99 3E 80 D3 99 3E 60 D3        00A8
A01B   OUT (99),A    D399     99 3E 81 D3 99 3E 08 D3        00A8
A01D   LD A,60       3E60     99 3E 88 D3 99 3E 82 D3        00A8
A01F   OUT (99),A    D399     99 3E 89 D3 99 3E 02 D3        00A8
A021   LD A,81       3E81     99 3E 8F D3 99 21 91 AD        00A8
A023   OUT (99),A    D399     CD 05 AB CD 3E AB CD 6E        00A8
A025   LD A,08       3E08                                    00A8
A027   OUT (99),A    D399     PAGE     0   1   2   3         00A8
A029   LD A,88       3E88     SLOT    0-X 1-X 1-X 1-X        00A8
A02B   OUT (99),A    D399     MAPPER   03  02  01  00    EB  00A8
```

The monitor screens is made up by a status bar, a menu bar. and the desktop. The menu bar has four different menus, respectively the SYSTEM menu, the INSTALLATIONS menu, the OPTIONS menu and finally the STEP/TRACE menu. Below the menu bar. the desktop can be found, which consists of the debugger accompanied by a window displaying the CPU registers, a mini-monitor with hexadecimal and ASCII dump and a stack viewer.

Mini-monitor

In the lower region of the desktop there is a mini-monitor displaying a hex dump and an ASCII dump. This mini-monitor can be linked to the cursor or to the PC-register or it can be used to display a fixed address. This mini-monitor is real-time updated.

Stack viewer

The stack works according to the FILO principle (First In Last Out). This means that the value that is first placed on the stack will be the last to be lifted off the stack. The mnemonics PUSH and CALL put values onto the stack, the mnemonics POP and RET retrieve values from the stack.
The stack always uses 16-bit values. The stack viewer makes it possible to have an easy to access way to monitor stack changes.
If a program gets stuck after a ret it's almost always an error in the stack values (probably a PUSH or POP that forces a wrong RETurn value)
The stack-viewer is real-time updated.

Cursor and Program Counter Bar

This debugger uses independent cursor and PC-bar. The main advantage of this is that you can browse through the entire memory with the cursor without this having any influence on the program that's being debugged since the PC isn't changed by the browsing.
Of course the possibility exists to assign the value of the cursor to the PC or vice versa.

Safety

The debugger of Compass is made with a lot of detail for security; great effort has gone into the separation of the debugger and the program to be debugged. All instructions are executed in the program's environment. This means: all registers (including SP), interrupt state (DI or EI) and memory slot-selection are set up as displayed, followed by simulated execution of the instruction. In this way the program's stack and registers are separated from the debugger's stack and registers. Make sure the SP register points to a safe(=unused) location before tracing any stack-commands! Otherwise vital system information can be overwritten eventually resulting in a crash.

If you're going to safe-debug then it's good to know that ALL the different OUT instructions are diverted also, allowing to block certain ports. For example if you're debugging a program that directly handles the VDP and you'd allow this during debugging then the MSX could crash.

If you're debugging and a HALT instruction is encountered when the interrupts are disabled you'll receive a warning because under normal circumstances the MSX should completely block itself waiting for an interrupt that can never occur.

To separate Compass and the debugged program even further, the memory management of a normal MSX is also diverted to own routines. This means that instructions who read/write port #A8, #FF, #FE, #FD, #FC or subslotregisters #FFFF are handled by the memory management of Compass. (Except for INIR, INDR, INI and IND)

For the subslotregisters (address #FFFF) the following instructions are checked:

```
LD reg,(HL)
LD (HL),reg
LD (HL),value
LD A,(#FFFF)
LD (#FFFF),A
```

If a program uses a more exotic way of changing address #FFFF then Compass won't be able to automatically divert the instruction. So the (new) memory-selection will have to be done manually by the person who is debugging without execution of the instruction.
For example a LD IX,#FFFE followed by LD (IX+1),C will not be detected by the debugger. Executing the LD (IX+1),C instruction will write to the REAL subslotregister and may cause the computer to crash.

It is not possible to alter the slot selection for page 3 and thus it is not possible to trace instructions in page 3 in another slot than the primary ram slot currently selected. Nevertheless memory settings (port #A8 and #FFFF contents) for page 3 are well emulated. This will do for most debugging work.

Controls

The debugger supports the mouse and most of the functions can be activated using shortcuts. For a list of the shortcuts we refer to Appendix A: Shortcuts.
In the debugger all shortcuts that can be activated using a [CTRL] combination can also be activated without the [CTRL] key.
By pressing the [SELECT] key you can rapidly switch between the monitor and debugger, making it possible to quickly adapt values or keep track of larger memory blocks for easy debugging.

5.2 The debugger menus

The menu bar. of the debugger consists of five menus

```
        System         see chapter 2.3
        Installations  see chapter 5.2.1
        Options        see chapter 5.2.2
        IO-list        see chapter 5.2.3
        Step/trace     see chapter 5.2.4
```

5.2.1 Installations menu

Reachable by pressing [F2]

CPU

If you're running Compass on a Turbo-R you can use this option to switch between CPU mode.

MSX RST

This option indicates if you want the RST displayed on screen as they are intended by the standard MSX-BIOS. This option is very useful when writing your own BIOS. For example, if this option is turned on then RST #30 is shown as follows:

```
        RST 30 DB 00;578A       F7008A57
```

If you turn the option off, the screen will display

```
        RST 30  F7
        NOP     00
        ADC A,D 8A
        LD D,A  57
```

LINKING

This allows you to bind the scrolling of the mini-monitor to your actions. If this is turned off then the mini-monitor doesn't respond to actions of the cursor or debugger.
If this option says 'cursor' then the mini-monitor will change its address so that it displays the memory address that is indicated by the cursor.
When displaying 'PC reg.' the mini-monitor will respond to changes of the program counter and will display the according memory. If you manually force the mini-monitor to a certain address then this option is automatically turned off.

MAIN INSTALL

Use this to change Compass' global settings. For more info read chapter 10.

5.2.2 Options menu

Reachable by pressing [F3]

DISASSEMBLER ADDRESS

Use this to change the address of the debugger. [ESC] cancels this option when selected.

MONITOR ADDRESS

Use this to change the address of the mini-monitor. [ESC] cancels this option when selected.
Linking of the mini-monitor is turned off.

GO

Use this to start real execution of a program at a given address. [ESC] cancels this option when selected.
This starts the program and gives all control to the program, no checks are made when this option is invoked. Be very careful, when wrongly used your computer crashes! When debugging a program make sure you have saved your work before trying to execute it. If your program doesn't react as was planned or it changes memory banks etc. then you can lose all source modifications made after your last save!

REGISTERS

Use this if you want to alter the registers. A selection bar will appear in the register window allowing you to choose a register-couple to change.
Pressing space allows you to change the value of the registers. The value can be explicitly given, or can be a label or can be calculated on the fly.
The flags and the DI/EI indication can also be selected but using the space bar on them will only switch them to their alternate state. Register IR can't be modified since this doesn't make much sense.
[ESC] ends this option.

TEMPORARY

This option stores the address of the cursor in the call buffer and brings the cursor to the address given as parameter for this option. By storing the current value of the cursor in this buffer you can return here using the option RETURN (see below). Use [ESC] to cancel if you changed your mind.

RETURN

Gets the last value stored with TEMPORARY back from the call buffer and returns the cursor to this position. The call buffer for this option uses the FILO principle so nested calls can be made.
Combination of this last two options allows you to efficiently look at a call routine without having to actually execute it or remember the cursor position manually so you can return there later.

PRINT

If you want to print a disassembled chunk of memory this is the option you were looking for. The requested parameters are beginning and ending of the block to disassemble. During this inputting you can press [ESC] to cancel.

DIS=>SOURCEBUFFER

Use this little option to disassemble some code and place the disassembled code in the current source buffer.
Attention: THIS WILL DESTROY THE CURRENT CONTENT OF THE SOURCE BUFFER!

5.2.3 IO-list menu

In this menu, which you can reach by pressing [F4], are two options to alter the IO-list. The IO-list can contain at most 17 ports which will be ignored during stepping/tracing of OUT instructions. (also OTIR,...) If for example you put the four ports #98 up until #9B in it, all direct VDP commands will be blocked and will be ignored. This option's main purpose is to avoid locking and screen-garbage.
Remarks:
-port read instructions are ALWAYS performed in real. (except for port #A8,#FC,#FD,#FE,#FF)
-It has no influence if you insert port #A8,#FC,#FD,#FE or #FF into the IO-list.

INSERT IO

If the list isn't full you can enter a new port to be ignored in the list.
[ESC] cancels the operation.

DELETE IO

If the list isn't empty this option allows you to remove one of the ports in the list. When selected you can use the cursor to move up and down through the list. Using the [SPACE BAR] you indicate which item should be removed. By pressing [ESC] you leave this option.

Below these two options a dotted line separates the options from the current IO-list.

5.2.4 Step/trace menu

Reachable by pressing [F5]

SET BREAKPOINT

This option makes it possible to add breakpoints to the breakpoint-list of the debugger, so you can stop tracing at a given address. Once selected a window appears containing the current address of the cursor.
If the cursor address is already selected as a breakpoint then this address
will be ignored and the window will not contain any value.
If the breakpoint-list is already filled you'll receive an error message indicating so. You'll need to remove one or more breakpoints in that case if you need to insert other ones. The max. number of breakpoints is 15.

RESET BREAKPOINT

Use this to remove old breakpoints from the breakpoint-list. A window will appear asking for the address to be removed from the breakpoint-list. If your current position is on top of a breakpoint than this address will be automatically filled out in the window. If the address isn't a breakpoint you will be notified of this, otherwise the breakpoint is simply removed from the list.

EXECUTE TILL BP

This will automatically trace a program until a breakpoint is encountered. An error will be produced if there are no breakpoints defined. You can stop the execution by pressing [ESC]. The screen is updated after a fixed number of instructions is executed.

VIEW BREAKPOINTS

A list will appear containing all the defined breakpoints. You can select a breakpoint and press [ENTER] to delete it. This will bring up the window that is used for the RESET BREAKPOINT with the address of the selected breakpoint already filled out.

SPECIAL TRACE

This is a special variant of EXECUTE TILL BP. It will first place a breakpoint after the current instruction and then start the execute-till-breakpoint-routine.
Useful for stepping subroutines in a safe way because the whole subroutine is traced.

If you would do a real STEP on a call that modifies IO ports or change VDP/VRAM contents your MSX could crash.

5.3 Important keys

Most shortcuts for the debugger are explained in appendix A. Nevertheless there are five shortcuts that need your special attention.

[CTRL] + [S] or [S]
Pressing this shortcut will perform a so-called STEP-command. The instruction at the Program Counter is executed. Register contents are modified accordingly.

[CTRL] + [T] or [T]
This shortcut executes the TRACE command. This is almost the same as the STEP instruction, but now the CALL instruction is not directly executed; the return address is PUSHed onto the stack and the debugger jumps to the new address.

Executing a CALL as a single instruction is of course a lot faster but in such a case Compass isn't capable of checking the instructions in the subroutine.
If this subroutine directly changes the memory banks, VDP registers or VRAM contents then your MSX may crash!
Therefore it's better to trace each instruction unless you are sure that nothing dangerous will happen in the subroutine. Since going through an entire subroutine can take some time the option SPECIAL TRACE is very useful for this purpose.

[CTRL] + [H] or [H]
This is an easy way to place the PC register at the same address as the cursor.

[CTRL] + [J] or [J]
This is the inverse of [H]. The cursor will be placed at the same address as the PC-register is indicating.

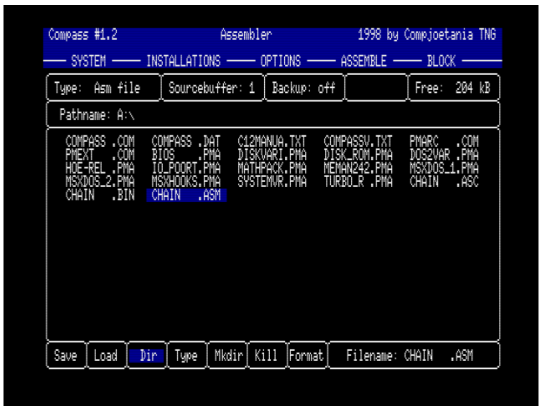
[CTRL] + [I] or [I]
This sets the SP register back to its initial value (when Compass was entered from the MSX-DOS command prompt minus 512 bytes) In most cases this is a safe stack space.

6 Disk

The disk menu can be reached by selecting the option DISK from the SYSTEM menu. Most people prefer to use the short-cut [CTRL] + [D]

6.1 Disk screen & controls

The disk menu replaces the entire desktop. At the top some important settings are displayed like the current source buffer (only if you came from the assembler), backup settings and the currently selected file-type. Below this a line is reserved to display the path and most of the desktop is used to display the directory contents. At the bottom the possible options are displayed respectively SAVE, LOAD, DIR, TYPE, MKDIR, KILL and FORMAT. At the right of these options the filename is displayed that will be used to perform these operations upon.



```
Compass #1.2             Assembler            1998 by Compjoetania TNG
── SYSTEM ── INSTALLATIONS ── OPTIONS ── ASSEMBLE ── BLOCK ──
Type:  Asm file | Sourcebuffer: 1 | Backup: off |  | Free:  204 kB
Pathname: A:\
 COMPASS .COM   COMPASS .DAT   C12MANUA.TXT   COMPASSV.TXT   PMARC   .COM
 PMEXT   .COM   BIOS    .PMA   DISKVARI.PMA   DISK_ROM.PMA   DOS2VAR .PMA
 HOE-REL .PMA   IO_POORT.PMA   MATHPACK.PMA   MEMAN242.PMA   MSXDOS_1.PMA
 MSXDOS_2.PMA   MSXHOOKS.PMA   SYSTEMVR.PMA   TURBO_R .PMA   CHAIN   .ASC
 CHAIN   .BIN   CHAIN   .ASM
Save | Load | Dir | Type | Mkdir | Kill | Format |  Filename: CHAIN   .ASM
```

To enter a filename just start typing the name. If you are entering a name you can press [ESC] to undo the modifications, the old name will then reappear.

Use the keys [CTRL] + [B] to change the backup settings. The possible values are 'off', 'asm' or 'asc'. The value 'asm' will have the effect that every time you want to overwrite an existing ASM-file this will be backupedd to using the same name but with the extension BAK. Previous backups of the same file will be lost. The value ‘asc’ will do the same for ASCII files.

Use the keys [CTRL] + [T] to change the current file type which you want to handle. See also chapter 6.2 and 6.3.2.

Use the keys [CTRL] + [1] until [4] to change the current source buffer. This option can be used when you entered the disk menu from the assembler. This is a very useful way to load or save multiple sources without the need to repeatedly switch to the assembler just to switch from one source buffer to another.
Each source does have its own name buffer, so there isn't any need to be concerned to accidentally overwrite a file because of forgetting to alter the name when you changed the source buffer.

Pressing [ESC] ends the disk menu and takes you back to the Compass part from which you summoned the disk menu.

6.2 File types

Compass gives you to the opportunity to operate on different kind of files.
these type are:

- Assembler files (only from within the assembler)
- ASCII files (only from within the assembler)
- Binary files
- Data files
- Block files (only from within the assembler)
- Sector (not really a file but nevertheless useful)

Notice the fact that if the disk menu is invoked from within the monitor or debugger you only have access to 3 of the 6 possibilities.

6.2.1 ASSEMBLER FILES

One of the nicest things in Compass is its ability to quickly load and save sources. This is possible by using this new type of file, the assembler file.
These files contain the total of all the necessary structures that Compass uses to store sources in memory.
Since this is a memory dump of the source and data buffers no conversion is needed when sources are saved or loaded this way. Of course since all this extra info is saved the files will usually be bigger then the same source saved in ASCII but the gain of time makes up for it.

6.2.2 ASCII FILES

Of course Compass would be quite useless for most people if the old trusted ASCII file couldn't be read (and saved of course). This option, in combination with the AUTO-ALIGN, gives you also the possibility to read and write normal text files in the editor.
It can be very useful to have a nice reference online in an other source buffer, like the BIOS overview text.

6.2.3 BINARY FILES

To store and retrieve chunks of memory Compass supports the BASIC bload format.
These files contain a seven bytes header (db #FE, dw start, dw end, dw begin_execution) followed by the data itself.
When saving you will be prompted to enter the addresses needed for the header.
When loading this address will determine the destination address of the block in memory.

6.2.4 DATA FILES

These are raw datafiles, with no header. Therefore when loading such a block of bytes you will be asked where the file must be placed in memory.
The ending address for loading is also asked, this will automatically contain the address where the last byte should be placed if the entire file is read. By changing this address you can stop reading before the end of the file is read.
When saving the begin and end address will be asked and the memory will simply be dumped in a file.

6.2.5 BLOCK FILES

These are just simple ASCII files. If you've chosen a block in the assembler/editor then this block can be saved as an ASCII file. When reading a block this piece of text will be inserted at the current line in the current buffer. This block will be selected when you return to the assembler.

6.2.6 SECTORS

Compass is capable of directly working with sectors (very nice if you want to alter the boot record of the disk for example). THIS IS OF COURSE VERY DANGEROUS! By typing a wrong sector number or by mistake overwriting a wrong disk you can permanently lose vital data on the disk.
Be very very cautious. This option is only valid if you want to change data on a floppy. For safety reasons this option will have no effect when trying to write sectors on your hard disk.
You will be asked for the source address, the first sector on disk and the number of sectors when saving. The same data will be asked when trying to read.

6.3 Disk menu

The disk menu contains the options SAVE, LOAD, DIR, TYPE, MKDIR, KILL, FORMAT.
The first two are already handled in the according file types.

6.3.1 DIR

This is the default option when you enter the disk menu, this should prevent accidentally destroying data.
If you choose this option you can enter a path in the path-line. When this is done and the pathname is valid then the contents of this directory is displayed. Use the cursor keys and space to choose a file in the list. If a name is surrounded by brackets then this is a directory name.
Choosing a directory will result in the displaying its contents and the modification of the pathname.
If a directory has too many entries to display you will be asked if you want to continue to the other names not on screen.
If you have chosen a name this name will be displayed in the name selection and the selection bar will return to the menu options.

6.3.2 TYPE

This has the same effect as pressing [CTRL]+[T].
The different file types are handled in chapter 6.2.
Remember the difference between the assembler and the monitor/debugger menu!

6.3.3 MKDIR

This makes a new directory in the current subdirectory. The current filename is used as name for the new directory. This works only with MSX-DOS2 of course.

6.3.4 KILL

This option will first ask for confirmation before actually destroying the file.

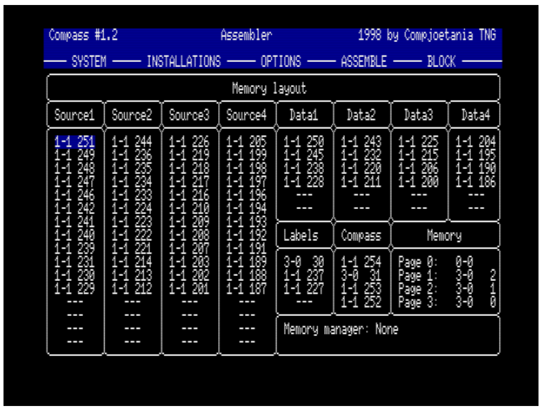
6.3.5 FORMAT

The type of formatting will be asked and the formatting will begin immediately when the choice is made. **NO FURTHER CONFIRMATION IS ASKED!** Be sure you have put the right disk in the disk drive. Also the label buffer will be erased for this action so after a format the label information built up during assembling is gone.

7 Memory

Some of the most powerful features of Compass is the ability to configure your memory. Especially when you make use of a memory manager like DOS2 or MemMan, because memory used by Compass will be protected by these managers.
If you don't use a memory manager, memory used by Compass can be corrupted by other programs.

If Compass is launched for the first time, Compass will automatically configure your memory.
Changes you may want to make can be saved by performing a SAVE INSTALLATIONS. (See chapter 10)



```
Compass #1.2              Assembler            1998 by Compjoetania TNG
── SYSTEM ── INSTALLATIONS ── OPTIONS ── ASSEMBLE ── BLOCK ──
                          Memory layout
Source1  Source2  Source3  Source4  Data1    Data2    Data3    Data4
1-1 251  1-1 244  1-1 226  1-1 205  1-1 250  1-1 243  1-1 225  1-1 204
1-1 249  1-1 236  1-1 219  1-1 199  1-1 245  1-1 232  1-1 215  1-1 195
1-1 248  1-1 235  1-1 218  1-1 198  1-1 238  1-1 220  1-1 206  1-1 190
1-1 247  1-1 234  1-1 217  1-1 197  1-1 228  1-1 211  1-1 200  1-1 186
1-1 246  1-1 233  1-1 216  1-1 196    ---      ---      ---      ---
1-1 242  1-1 224  1-1 210  1-1 194    ---      ---      ---      ---
1-1 241  1-1 223  1-1 209  1-1 193
1-1 240  1-1 222  1-1 208  1-1 192  Labels   Compass      Memory
1-1 239  1-1 221  1-1 207  1-1 191
1-1 231  1-1 214  1-1 203  1-1 189  3-0  30  1-1 254  Page 0:  0-0
1-1 230  1-1 213  1-1 202  1-1 188  1-1 237  3-0  31  Page 1:  3-0   2
1-1 229  1-1 212  1-1 201  1-1 187  1-1 227  1-1 253  Page 2:  3-0   1
  ---      ---      ---      ---      ---    1-1 252  Page 3:  3-0   0
  ---      ---      ---      ---    Memory manager: None
  ---      ---      ---      ---
  ---      ---      ---      ---
```

A MSX turbo-R with 4MB ram in a slotexpander

Each block is identified by its location in the computer.
Examples:

```
3-2 7  :this block is situated in primary slot 3, sub slot 2, mapper segment 7
 1  20 :this block is situated in Primary slot 1, mapper segment 20
---    :no memory selected for this purpose
```

Note the following:

```
3-2 7   <-- closest neighbour
 ---    <-- this block can be changed because it follows a selected block
 ---    <-- this block can't be changed because it is isolated
```

Press [SPACE] to change a block. Use cursors to choose a new block or to disable a block. Only unused memory blocks can be selected. Of course you can't disable the Compass-program-blocks and the first label-buffer-block.

Note the following:
The first label-buffer-block and the second Compass-program-block should be situated in the primary mapper.

On the right side of the Compass-program-blocks, you can select the work memory.
This work memory is:

- used as target memory for assembling
- shown in the Monitor
- shown in the Debugger

You can alter these settings for page 0, 1 and 2. Not for page 3 because the system variables kept in this page should be reachable at any time. Make sure you don't select a block that is in use by Compass. Compass will likely crash if it is (partially) overwritten.

remark: if you have lots of memory (>6MB) it is possible that not all memory can be paged in due to current limitations of Compass.

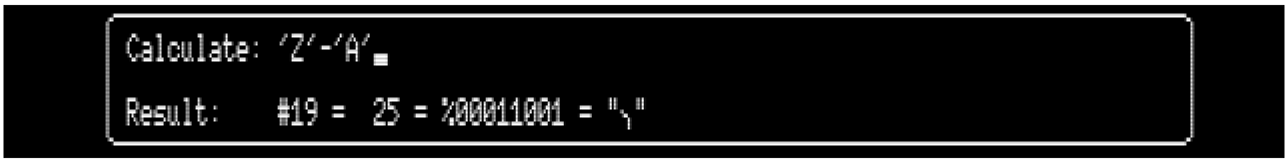
8 Calculator

When you use the SYSTEM menu or the shortcut [CTRL] + [C] you will see the calculator appear on your screen.
The calculator consists of a small frame, which can be moved up- and downwards using the cursors. This is useful to look at the information hidden 'behind' the calculator.
You can leave the calculator by pressing [ESC].

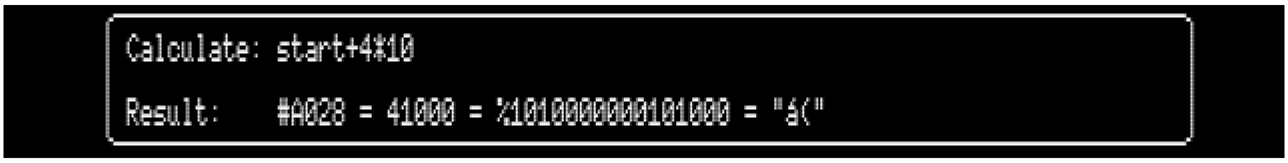
Compass is equipped with a very powerful calculator. The next enumeration should clearly state this.

- The calculator uses the mathematical priority of operations (raise to the n-th power, multiply, divide, add and subtract) and brackets.
  There is no limitation to the number of brackets used in an expression making for example ((6*(76-4)/23+(12-45))*7) a valid expression.
- The minus sign can be used as many times as one wishes.
  This makes ---5 equal to -5.
- The logical operators AND, OR and XOR and the MOD operator.
- You can use the ASCII values of single characters by enclosing them with single quotes.
- You can use the result of the previous made calculation in the current expression by using the $-sign.
- It is possible to place spaces between the digits of the same number, making % 1100 00 11 a correct number while keeping the readability high.
- The calculator can use the labels in the label-buffer.
  If your program uses labels as START, ENDPROG, DATA, ... then after compilation you can use an expression as : (DATA - START) / 1024 +ENDPROG for example.

When an expression is calculated the result is shown as an 8/16 bit hexadecimal number, an 8/16 bit binary number, as a regular decimal number and as one/two character(s). If you entered an error in the expression you will get a chance to correct it and you can try again. If all went fine the command line will be empty and you can enter the next expression.



```
Calculate: 'Z'-'A'

Result:    #19 =  25 = %00011001 = "╮"
```



```
Calculate: start+4*10

Result:    #A028 = 41000 = %1010000000101000 = "á("
```

9 Slot view

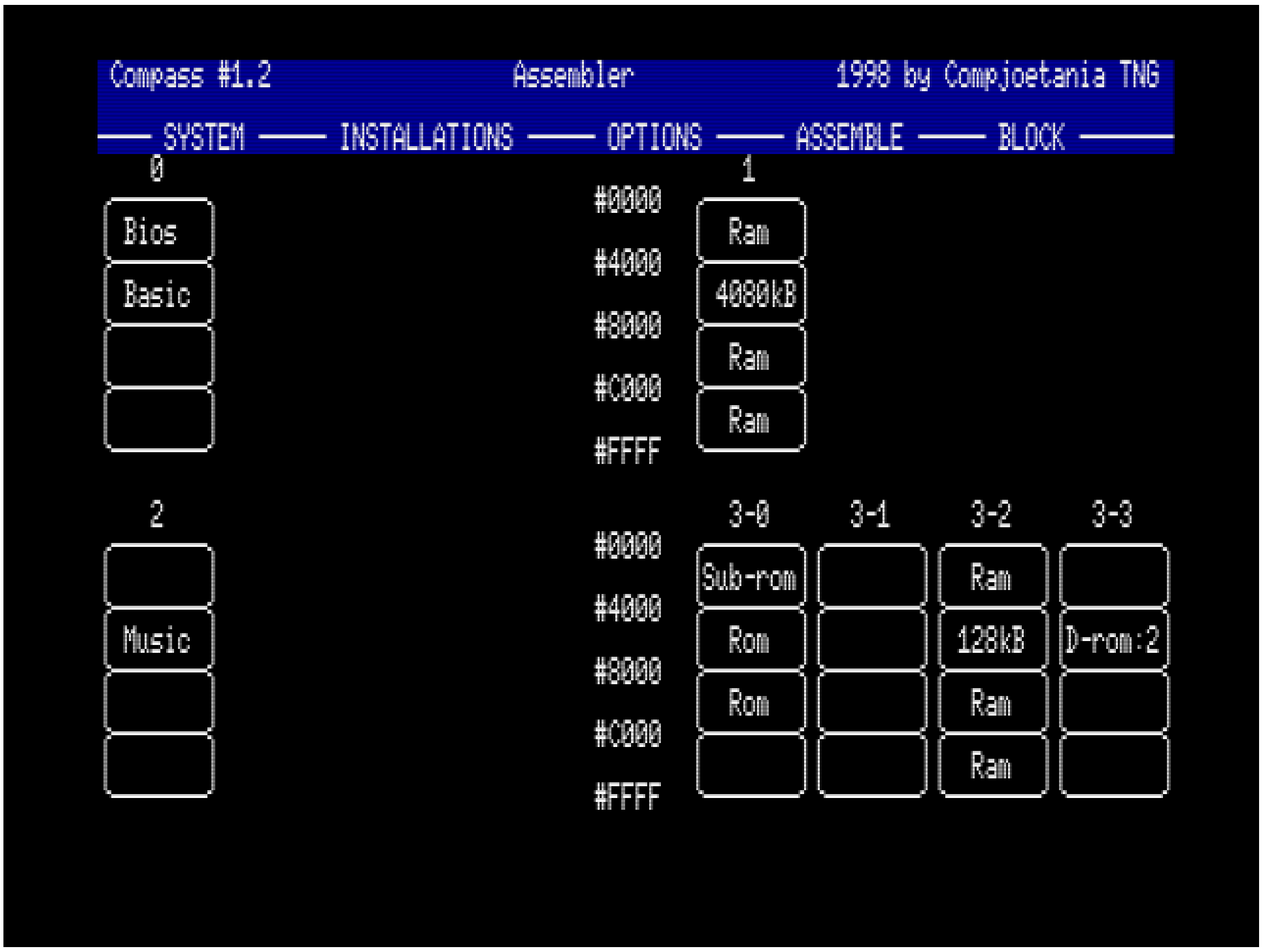
This option will show your hardware configuration.

Following things are displayed:

BIOS	This is the MSX-BIOS ROM (always in primary slot 0, page 0)
BASIC	This is the MSX-BASIC ROM (always in primary slot 0, page 1)
ROM	This page contains ROM
D-rom:x	This page contains a disk rom with x drives connected to it
MUSIC	This page contains a MSX-MUSIC ROM (FM-PAC)
AUDIO	This page contains a Philips Music Module ROM (MSX-AUDIO)
RAM	A memory mapper or fixed memory module is situated here Also the capacity is mentioned.

You can leave Slot view by pressing [ESC]

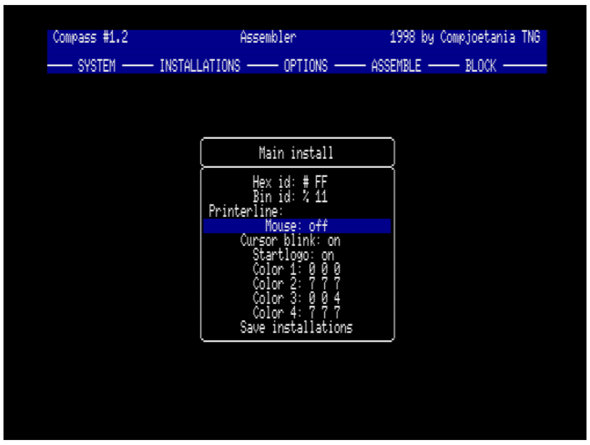
Remark: possible ROM pages or fixed RAM pages in page 3 are not shown.



A Philips NMS 8255 with extra memory mapper and fm-pac inserted

10 Main install

This menu is used to change and save the current settings of Compass.
Use [SPACE] to select the desired parameter.



```
Main install

Hex id: # FF
Bin id: % 11
Printerline:
Mouse: off
Cursor blink: on
Startlogo: on
Color 1: 0 0 0
Color 2: 7 7 7
Color 3: 0 0 4
Color 4: 7 7 7
Save installations
```

HEX-ID

Use this option to set your favourite hexadecimal ID.
Compass uses the following ID's by default:

in front: # and &H , example: #FD9F &HFD9F
behind: H , example: FD9FH

Use cursors to select the desired position and enter your ID bytes if you want to.

BIN-ID

Use this option to set your favourite binary ID.

in front: % and &B , example: %1010 &B1010
behind: B , example: 1010B

Use cursors to select the desired position and enter your ID bytes if you want to.

Note: the following characters are not allowed for ID use:

```
"'()*+,-./0123456789:;<=>
```

PRINTERLINE

This data is sent to the printer before the actual data. An ESC code should be entered as a ^.

MOUSE

Use this option to switch mouse control on or off. If no mouse is connected to the computer, we recommend to disable mouse control. Mouse control is disabled by default.

CURSOR BLINK

Use this option do dis/enable the blinking of the cursor. Useful for people who dislike blinking cursors...

START LOGO

Use this option to dis/enable the startup logo of Compass. This logo overwrites some VRAM. Booting Compass is also much faster without the logo.

COLOR 1-4

You can change Compass' colours with this option. This is done by changing the Red, Green and Blue values of the palette.

SAVE INSTALLATIONS

VERY USEFUL OPTION!
This option will save your current Compass configuration:

- Mouse, printer line,...
- Memory configuration
- Editor parameters for each source buffer: Label length, Return insert on/off, Auto align
- Backup status of disk menu

11 Macro's

Macro's are probably the most used assembler-directives ever. They give the possibility to place short pieces of often used code under a self defined command name, allowing you to use this home-made command instead of retyping all the commands every single time. The assembler will automatically expand these commands to the original code.

Besides the fact of saving time used for retyping al the commands one after another macro's also aid in maintaining the readability of your code. Of course you can pass parameters to your little routines which will be inserted in the macro during the assembling.
Compass also supports recursive macro's, meaning that one macro can use other (or it's own) macro inside it's definition.

Three assembler directives are preserved for usage with macro's.

MACRO
Syntax: MACRO [@param1, @param2, @param3]

This directive is used to open a macro definition. You can pass some parameters which name has to begin with an @, to differentiate them of regular labels. Once defined this way you can use this macro-labels inside of your macro definition, allowing you to use this macro for different purposes. Inside a macro you can place certain conditions on the parameters by use of conditional assembling.

You can't open a macro definition inside another macro definition.

ENDM
syntax: ENDM

Using this directive you end the current macro definition. The code entered between the MACRO and ENDM doesn't get assembled on the spot, instead every reference to the macro name later will be replaced by this code during assembling.

DEFL
syntax: label: DEFL value

This assembler directive does the same thing as the EQU directive, you use it to assign a value to a label. This directive can only be used inside a macro.

Within a macro you can use also the label-ending "@sym" or "@SYM". These four signs in a label will be translated to a four digit number. This number will automatically be raised every time a macro is expanded. When first invoked this will be 0000, the second time a macro is expanded this will be 0001 etc.
This has as advantage that a label can be used in macros that are more than once expanded without that this results in errors. If not this would result in a redefinition of the same label. Take a look at the examples for more clarity.

When a macro is invoked the passed parameters are placed literally in the macro itself. This means that the parameters itself are replaced in the source of the macro before the compilation of the instruction takes place. This allows you to pass registers, labels, texts etc. as parameters for the internal macro-code.

Example 1:

```
bc_de:  macro @part1,@part2
        ld bc,@part1
        ld de,@part2
        endm

        ...              ;program
        bc_de 567,#babe  ;macro is used in program
        ...              ;program
```

In this simple example the registers BC en DE will be filled with the parameters passed to macro bc_de. Let's assume you have two parameters the numbers 567 and #babe.
When assembling bc_de 567,#babe the assembler will generate the following code:

```
        ld bc,567
        ld de,#babe
```

Assume you want to load bc with the value at (#FBB1) then you pass as parameters

```
        bc_de (#FBB1),#babe
```

The compiler will replace the @part1 and @part2 and generate code for

```
        ld bc,(#FBB1)
        ld de,#babe
```

Example 2:

```
PRINT:  macro @p1
        push hl
        ld hl,m@sym
        call mout       ; a printing routine
        pop hl
        jr l@sym
m@sym:  db @p1,0
l@sym:
        endm
```

This generates a printing routine every time the macro PRINT is encountered.
For example:

```
        PRINT "Hello World :-)"
        PRINT "This is a test line"
```

By using the @sym construction it is possible to invoke the macro more than once. The first PRINT will be assembled with the @sym replaced by 0000. the next PRINT invocation the @sym will be replaced by 0001.

If we left out this @sym from let's say the label m@sym then when translating the code the first time (PRINT "Hello World") we would have generated a label 'm '. The second time (PRINT "This is a test line") the assembler would find again a definition for a label m.
Since the label m was already declared the first time this would give an error. By placing the @sym code after the label this is avoided. We now have two different labels (m0000 and m0001 to be exactly).

Example 3:

```
bdos:   macro @fun,@fcb
        if "@fcb">""
        ld de,@fcb
        endif
        ld c,@fun
        call #0005
        endm
```

This is a more advanced macro. The register pair DE is only loaded with the value @fcb if this parameter is declared by the invocation. If you should call this macro with only one parameter than the ld de,@fcb isn't assembled and register de will not be modified by this macro.

Example 4:

```
fact:   macro @result,@n
        if @n=1
@result defl 1
        else
        fact t@sym,@n-1
@result defl t@sym*@n
        endif
        endm
```

This is a more difficult macro which uses recursion. In this examples you can see how you can define and use internal labels (if they don't exists some where else) and assign a value to them. This macro should be called with a label as parameter and a number. for example:
fact label,5; this calculates the factorial of 5 and places the result in
label.

12 Conditional assembly

With conditional assembly you're able to select which part(s) of your source should be assembled. You are allowed to use 16 nested IF statements.

`IF / COND`
`Syntax: IF / COND parameter [condition parameter]`

Use this to open a condition: If the expression behind IF (or COND) is true (not equal to zero), the assembling will continue as normal. If it is false, all mnemonics between the IF statement and the next ENDIF statement are not assembled.

Examples:

```
IF 1               ;true (not equal to zero)
IF 3<=2            ;false
IF "Compass">"GPS" ;false (stringsize)
IF "ABC">"AAA"     ;true (ASCII alphabetical)
```

`ELSE`
`Syntax: ELSE`

Use this option to switch assembling on if it was turned off. Assembly is turned off if it was on.

`ENDIF / ENDC`
`Syntax: ENDIF / ENDC`

Use this to turn assembly on. Of course only when you've use IF.

Example 1:

```
label1  equu    1

        if      label1=1
        db      "Compass",0
        else
        db      "Test",0
        endif
```

Only "Compass" will be assembled.

Example 2:

```
              if     "Test">"test"
              if     1=2
              defs   50    ;this command is not assembled
              endif
              defs   100     ;this command is assembled
              endif
```

Example 3:

```
msxtype       equ    1  ; 1=MSX1,2=MSX2,3=MSX2+,4=TURBOR

              if     msxtype=1
maxvdpreg     equ    8
maxcolor      equ    15
              else
maxvdpreg     equ    24
              if     msxtype=2
maxcolor      equ    255
              else
maxcolor      equ    19200
              endif

              ; Here the values of the labels maxcolor and maxvdpreg
              ; depend upon the value of the msxtype label
              db     maxvdpreg
              dw     maxcolor
```

As you can see in the last examples: Given the layout in compass this kind of nested if structures gets unreadable fast.
Not to mention the possible problems with labels not getting defined or getting redefined in the last example if you are not carefully combing trough your code.

13 Terminate & Stay Resident programs

These programs, usually abbreviated to TSR's, are designed to work as a background task. Some can be activated by hotkeys, others intercept hooks to be activated. These programs can only be used when running MemMan 2.4.
This Memory Manager does all the necessary memory allocation and hook maintenance for all the different TSR's and other MemMan-enabled programs so that we don't have any trouble with programs who, involuntarily, destroy each others address space and therefore make your MSX crash. MemMan 2.4 can be found on the Compass-disk.

For more information about TSR's you should get the TSR development kit made by the MST.

Previously the only method to make a TSR was to compile the source to a Relocatable File (see chapter 14) which could only be done with gen80 from the DOS command line and further be linked with a specially by MST written linker to created the .TSR file. This is definitely a very time consuming and long-winded method. For this reason Compass has built-in TSR compiling. To make this possible an extra directive had to be built, specifically:

TSRHOOKS

This command has to be used before you define the hooks and accompanying labels at the end of the source (See the TSR framework in the TSR development kit). Afterwards you can compile your source to a TSR, this is a three-pass assembling. The compiled TSR can be loaded from DOS using the TL program. If the TSR doesn't function properly you can return to Compass using the Compass hotkey and adapt the source. This ensures an ideal working environment for TSR development.

Example:

```
        code
        .
        .
        .
        code

        TSRHOOKS
hooks:
        defw ...,...,...,...
        end
```

14 Relocatable files

In the prehistoric times of computer programming, CPU time was an awful expensive item. To make small changes in a 'large' program and recompile the entire source was something you'd better avoided. Beside that there was also the matter of multiple users using the same machine at the same time and the dangers that programs in the memory would trespassing each others memory space. For this reason the principle of small, quick to compile, modules raised its head.
The principal is thus to use different little sources, which by means of an assembler could be compiled and afterwards could be linked to each other and be placed anywhere in the main-memory. The big advantage was that now a small change in one or more small modules would cost less time to compile. The linking itself is relatively fast, so that the total time was decreased considerably. There was however a drawback to this method. When using labels and routines from an other module their (relative) address isn't known by the compiler since it isn't aware of the target address or the order in linking.
To solve this problem the Relocatable Files were designed. These type of files consist of a bitstream containing all unknown addresses in the form of offset to different counter registers. The linker program is now able to calculate the absolute addresses since he is familiar to the linking order of the project. For more technical info about the Relocatable files we advise you to read the articles once published by MCCM.

To make the development of relocatable files possible some extra assembler directives were added, making it possible to control the byte flow in the final program

CSEG / DSEG / ASEG
```
Usage: CSEG / DSEG / ASEG
```

Assembling to relocatable files makes it possible to use three different counters. These are known as the code-segment (CSEG), the data-segment (DSEG) and the absolute-segment (ASEG). These segments are joined during linking, therefore enabling clustering of data and code blocks. When you start assembling, the code-segment is automatically turned on. By using one of the 3 commands you can indicate in which block the next code has to be placed.

PUBLIC / EXTERN
```
Usage: PUBLIC / EXTERN label, label, … , label
```

By using the directive PUBLIC you declare labels in the current module known to the other modules, thus indicating that this label can be used by other modules. If a module want to use a label declared in an other module it has to indicate this by declaring that label with EXTERN. Instead of using the PUBLIC directive you can achieve the same effect by stating two double-points after the label name. So 'label::' is functional equivalent to 'PUBLIC label'.

.PHASE
```
Usage :. .PHASE address
```

Use this directive to create a temporary absolute segment. The program counter will be assigned the value 'address' and all the labels between this directive and the .DEPHASE directive will be placed upon this address or higher. This option is extremely useful if you want to replace a part of your program to an absolute address. Just place the desired block between the .PHASE / .DEPHASE directives, making the code to be compiled to the desired target address.

.DEPHASE

`Usage: .DEPHASE`

Using .DEPHASE restores the program counter to the old value and segment.

Appendix A: Shortcuts

If you are a frequent Compass user you will certainly appreciate the shortcuts, these are thousands times faster than going through the regular select-menu-select-option method.

General shortcuts

Japanese NO	Z80
Japanese YES	R800-ROM
CTRL + C	Calculator
CTRL + D	To disk menu
CTRL + Q	Quit the program
SHIFT + ESC	Exit to shell / resume Compass

Assembler shortcuts

CTRL + A	Assemble source to memory
CTRL + E	End of block
CTRL + G	Go to memory location (Run program)
CTRL + J	Jump to line in current source buffer
CTRL + K	Copy block to current line and buffer
CTRL + L	Copy the definition label from the block to current position
CTRL + N	Continue the search
CTRL + P	Print the currently selected block
CTRL + R	Show error list
CTRL + S	Set the beginning of the block
CTRL + T	Reset block
CTRL + V	Move block to current line and buffer
CTRL + W	Delete block
CTRL + Z	Search/replace text
CTRL + 1	Select source buffer 1
CTRL + 2	Select source buffer 2
CTRL + 3	Select source buffer 3
CTRL + 4	Select source buffer 4
CTRL + 5	Select PAL mode (50 Hz)
CTRL + 6	Select NTSC mode (60 Hz)
CTRL + F1	Store current position underneath the F1 key
CTRL + F2	Store current position underneath the F2 key
CTRL + F3	Store current position underneath the F3 key
CTRL + F4	Store current position underneath the F4 key
CTRL + F5	Store current position underneath the F5 key

SHIFT + F1	Get position stored with CTRL F1 and jump to that line
SHIFT + F2	Get position stored with CTRL F2 and jump to that line
SHIFT + F3	Get position stored with CTRL F3 and jump to that line
SHIFT + F4	Get position stored with CTRL F4 and jump to that line
SHIFT + F5	Get position stored with CTRL F5 and jump to that line
CTRL + INS	Insert a line
SHIFT + INS	Insert 10 lines
CTRL + DEL	Remove a line
SHIFT + DEL	Remove until end of line
HOME	Jump to beginning / ending of the current source buffer.
GRAPH + cursors	Change: Uppercase >< lowercase
CTRL + up	Scroll up one screen
CTRL + down	Scroll down one screen
CTRL + right	To the beginning of the next word
CTRL + left	To the end of the previous word
SHIFT + up	Scroll up one hundred lines
SHIFT + down	Scroll down one hundred lines
SHIFT + right	To the right of the last character of the line
SHIFT + left	To the first position of the line
STOP	Go to the Monitor

Monitor shortcuts

CTRL + A	Set current address
CTRL + E	'PEEK' an address
CTRL + F	Fill memory with given value
CTRL + K	Copy memory block
CTRL + N	Continue the search
CTRL + O	'POKE' a value into an address
CTRL + P	Print a block of memory
CTRL + X	Toggle Chaining mode
CTRL + Z	Search
CTRL + up	Scroll up one screen
CTRL + down	Scroll down one screen
STOP	Go to the debugger
SELECT	Toggle between debugger and monitor

Debugger shortcuts

Useful to adjust the disassembly start-address:

CTRL + up	Scrolls back exactly 1 byte
CTRL + down	Scrolls forward exactly 1 byte

For the following shortcuts in the debugger you don't need to use the CTRL key, just the key will do.

CTRL + A	Set address
CTRL + B	Set breakpoint
CTRL + E	Execute until breakpoint
CTRL + H	Puts the program counter onto the current cursor address
CTRL + I	Sets the SP register back to its initial value
CTRL + J	Puts the cursor address onto the current program counter
CTRL + L	Toggle Linking mode of the mini-monitor
CTRL + P	Change current slot selection
CTRL + R	Change Registers
CTRL + S	Step
CTRL + T	Trace
CTRL + U	Address, stored with TEMPORARY gets 'popped'
CTRL + V	View breakpoints
CTRL + W	Erase breakpoint
CTRL + Y	Special trace

STOP	Go to the Assembler
SELECT	Toggle between debugger and monitor

SHIFT + up	Jumps back approx. 200 bytes
SHIFT + down	Jumps forward approx. 200 bytes

Appendix B: Error messages

Here's a list of the messages you may encounter when using Compass

Editor messages

`Databuffer to small.…`

The program is unable to load or edit a source because your data buffer isn't big enough. You are able to change this in the memory-menu (chapter 7).

`Destination in block !`

You are trying to copy or move a block while the cursor (the target line) is in the block itself.

`No block selected !`

You attempted to execute a block-command when you don't have a block selected.

`Printer not ready !`

Your printer isn't online when you tried to print something.

`Sourcebuffer to small.…`

The program is unable to load or edit a source because your source buffer isn't big enough. You are able to change this in the memory-menu (chapter 7).

`Text not found !`

The search function couldn't locate (any other) occurrence of the wanted text in the current buffer.

Assembler errors

`Bad instruction`

Compass didn't recognize the given command.

`Bad mnemonic`

The entered registers, labels, values, etc. weren't recognized by Compass.

`Division by zero`

You tried to divide a (label representing a) number by zero.

`Illegal symbol in label`

You are using an unauthorised symbol in a label.

`Label not defined`

You are using a labelname which isn't defined although it should be at this point during assembling.

`Label too long`

You have used a label name which exceeds the length of the maximum label length which is set using the INSTALLATIONS menu or the assembler directive .LABEL.

`Labelbuffer too small.`

During assembling the label buffer encountered an overflow. You can change the size of this buffer in the memory menu (chapter 7).

`Macro already opened`

You issued a MACRO command while you're already inside a MACRO definition.

`Macro not definied`

You are using a MACRO command without naming it properly.

`Macro not opened`

You're trying to close a macro definition with the ENDM command but you haven't opened any macro definition.

`Macro used as label`

You are trying to use a defined macro name as a label, this is forbidden.

`Missing ")"`

You entered an expression and forgot one or more closing brackets which leaves the expression open.

`Number out of range`

The number you are using exceeds the maximum value allowed for this operation, ex. LD A,300

`Offset out of range`

The target address for the relative offset cannot be reached.

`Phase still active`

You're trying to use ORG / CSEG /DSEG / ASEG commands while you're still in a phase-mode.

`Public not found`

The entered public label names don't appear in the current module.

`Redefined label`

The labelname you try to define is already defined elsewhere.

`Relocateable instruction`

You are using a relocatable instruction and you aren't assembling to a relocateable file.

`Too many if's`

You have more nested IF's than the allowed maximum of sixteen.

Monitor errors

```
        Not equal !
```
The two memory areas you are comparing aren't equal.

```
        Printer not ready !
```
You tried to print a memory dump but the printer seems to be off line.

```
        String not found !
```
The entered search string couldn't be found in the selected memory chunk.

Debugger errors

```
        Breakpoint-memory filled !
```
The memory allocated for breakpoint definitions is full.

```
        Call-buffer already empty !
```
You tried to retrieve an address from the call-buffer which is already empty.

```
        Call-buffer already filled !
```
You tried to put an new address in the call-buffer but there aren't any entries left.

```
        Printer not ready !
```
You tried to print a memory dump but the printer seems to be off line.

```
        There are no breakpoints to delete !
```
The breakpoint list is already completely empty.

```
        There is no breakpoint at that address !
```
You are trying to delete a breakpoint but there isn't a breakpoint defined at that address.

```
        This address is already defined as a breakpoint !
```
The address you gave already contained a breakpoint.

```
        You haven't defined any breakpoints !
```
You try to execute-until-breakpoint, but there aren't any breakpoints set.

Disk errors

```
        Disk I/O error !
```
A part of the disk can't be read or written.

```
        Disk off-line !
```
There is no disk in the drive or the disk isn't formatted.

`File not found`

The given file and/or pathname can't be found.

`No assembler file`

You try to open a file as an assemblerfile but the file doesn't seem to be of the ASM type.

`No binary file`

You try to open a file as a binaryfile but the file doesn't seem to be of the BIN type.

`No diskdrive !`

You try to save a sector on a hard drive. This operation isn't allowed by Compass.

`Write error !`

During writing an error occurred. Try again using another disk (maybe your disk is full ?).

`Write protected !`

The disk on which you tried to save is write-protected.

Appendix C: Reserved words

Here's a list of the reserved words in the assembler.
These are used during compilation and aren't allowed to be used as labels, macro's or similar.

Instructions

ADC	CPI	EXX	JP	NEG	PUSH	RLD	SET
ADD	CPIR	HALT	JR	NOP	RES	RR	SLA
AND	CPL	IM	LD	OR	RET	RRA	SLL
BIT	DAA	IN	LDD	OTDR	RETI	RRC	SRA
CALL	DEC	INC	LDDR	OTIR	RETN	RRCA	SRL
CCF	DI	IND	LDI	OUT	RL	RRD	SUB
CP	DJNZ	INDR	LDIR	OUTD	RLA	RST	XOR
CPD	EI	INI	MULUB	OUTI	RLC	SBF	
CPDR	EX	INIR	MULUW	POP	RLCA	SCF	

Operands

A	H	AF	IXH	C	PE
B	I	BC	IXL	M	PO
C	L	DE	IY	NC	Z
D	R	HL	IYH	NZ	
E	SP	IX	IYL	P	

Assembler directives

ASEG	DEFW/DW	INCLUDE
BREAKP	END	.LABEL
COND	ENDC	MACRO
CSEG	ENDIF	ORG
DEFB/DB	ENDM	PHASE
DEFC/DC	EQU	PUBLIC
DEFM/DM	EXTRN	TSRHOOKS
DEFL	IF	UPPER