



ATMAS-II

MANUAL

DRAFT, NOVEMBER 13, 2018.

ATMAS-II Manual

A Macro-assembler, editor and machine language monitor for the ATARI 400/600XL/800/800XL and ATARI 130XE with a diskdrive and at least 48kB RAM


The **ATMAS-II** program, disk and manual are copyright by

Ing. W. Hofacker GmbH and Dipl.-Ing. Peter Finzel


All rights reserved, 1985

Scanned, OCR'd, typeset in L^AT_EX 2_ε and translated from German to English
by Ivo van Poorten, 2011

LOAD INSTRUCTIONS









- Remove any cartridge from the cartridge slot of your ATARI Computer.
- Turn on the diskdrive and monitor.
- Once the red LED of the disk drive is off, insert the disc with the inscription at the top; the oval cut enters the drive first. Close the door of the diskdrive.
- Then turn your ATARI computer. On XL models you have to hold the  key while swithing on the computer.
- **ATMAS-II** is now loading. After a short time the opening credits appear and then the editor. The loading process is complete.



If the message '**ATMAS-II benötigt 48K-Speicher**' (**ATMAS-II** needs 48K Memory) appears, there are several options:

- a There's still a cartridge in the cartridge slot.
- b The  key is not pressed (XL models only).
- c Your computer does not have enough memory. **ATMAS-II** needs at least 48kByte.

DEMO PROGRAM HOWTO

If you want to view a short demo program of the **ATMAS-II** system before you read the manual, you can achieve this by following the instructions below:

 RD:DEMO   (Load the demo program)
 (Assemble the program)
 (Return to the editor)
 U  (Start the program)
 (Stop the program)

Note:  means you have to press the ESC-key.  means you have to hold the CTRL-key while you press the Y-key.

Contents

Contents	5
1 The editor	9
1.1 Using the editor	9
1.1.1 The status bar	9
1.1.2 Text window	10
1.1.3 Command line	10
1.2 Text input	10
1.3 Edit commands in text mode	10
1.3.1 Cursor movement	11
1.3.2 Deleting characters	11
1.3.3 Deleting lines	11
1.3.4 Cursor jumps	11
1.3.5 Tabulator	11
1.3.6 Copy-Register (C-Register)	12
1.3.7 Special commands for the editor	12
1.3.8 Calling the assembler and the monitor	12
1.4 Commands on the command line	13
1.4.1 Using the command line	13
1.4.2 Edit functions on the command line	13
1.4.3 More editing commands	13
1.4.4 Special functions	14
1.4.5 Commands for in-/output of text	14
1.4.6 Listings	15
1.4.7 Notes on advanced editor use	15
2 The macro assembler	17
2.1 Using the macro assembler	17
2.2 Input format of the assembler	17
2.2.1 Addressing modes	18
2.2.2 Labels	18
2.2.3 Konstante	19
2.2.4 Negative Konstante	19
2.2.5 Interner Adresszähler	19
2.2.6 Ausdruck	20
2.3 Assembler-Direktiven	20
2.3.1 ORG	20
2.3.2 EQU, EPZ (Equates)	21

2.3.3	DFB (Define Byte)	21
2.3.4	DFW (Define Word)	21
2.3.5	ASC (ASCII-String)	22
2.3.6	OUT (Output Listing)	23
2.4	Makrofähigkeit	23
2.4.1	Makro-Definition	23
2.4.2	Makroaufruf, Makroexpansion	24
2.4.3	Lokale Labels	24
2.4.4	Verschachtelte Makros	25
2.4.5	Makros contra Unterprogramme	25
3	Der Maschinensprache-Monitor	27
3.1	Allgemeine Benutzungshinweise	27
3.2	Befehlsvorat des Monitors	27
3.2.1	M - Memory-Dump	28
3.2.2	D - Disassemble	28
3.2.3	C - Change Memory	29
3.2.4	F - Fill Memory	29
3.2.5	B - Blocktransfer	29
3.2.6	G - Goto Address	30
3.2.7	S - Binary Save	30
3.2.8	L - Binary Load	32
3.2.9	E - Editor	32
3.2.10	I - Disketteninhaltsverzeichnis	32
4	Examples	33
4.1	Demo program, color scrolling	33
4.2	Macro libraries	35
A	ATMAS-II Memory Map	43
B	Error codes	45
B.1	Assembler errors	45
B.2	Editor errors	46
B.3	Monitor error codes	46
C	ATMAS-II REFERENCE CARD	47

Introduction

ATMAS-II is a powerful development system for assembly programs on your Atari computer. **ATMAS-II** consists of three integrated subsystems: an editor, a macro assembler and a machine language monitor. These components together allow a convenient and fast low level assembly program development.

All three parts of this programming environment are tightly integrated and only a keystroke away from each other. A user-friendly, screen-based editor with features such as a copy register, repeatable command sequences and space-saving real tabs, facilitate the entry of large programs. The macro assembler is able to achieve the shortest possible assembling time by using modern hashing algorithms such that longer programs are assembled in only a few seconds. Finally, there's a built-in monitor for testing programs.

Although someone who has never programmed in assembly will quickly feel familiar with **ATMAS-II**, it also offers many features for advanced users, such as the powerful editor functions and the macro programming commands.


It should be noted that this manual is not a textbook for learning 6502-machine language. I refer to other books on the subject, particularly to a book appearing in the fall of '85 by the author of this manual that is tailored specifically to the Atari computer. In addition, the examples in chapter 4 of this manual show a little insight into the assembly language programming of the Atari computer.

CHAPTER 1

The editor

The **ATMAS-II** editor is a screen oriented editor that works without any line numbers. It essentially behaves like a word processor. This eliminates the tedious entry of **LIST** commands which is known from several other editors (e.g. in **BASIC**). You can easily move through the listing by using the cursor keys, which then scrolls from top to bottom or from bottom to top.

1.1 Using the editor

Once **ATMAS-II** has been loaded from the floppy disk, you go directly into the editor, which is also sort of the Control Center of the **ATMAS-II** system. From here you can call the macro assembler and the machine language monitor. The  key will always bring you back to the editor.

After booting, you will see the following screen:

```

+-----+
I P:00000 T:17408 C:1024 OK I <---Status bar
+-----+
Cursor ->I C                                I
I                                           I
I      Textwindow                          I
I                                           I
I                                           I
+-----+
I .....I <---Command line
+-----+
```

1.1.1 The status bar

The status bar will continuously inform you about the cursor position, free memory, the copy register and the state of the editor.

'P:nnnnn' tells you the distance (in characters) of the cursor from the beginning of the text to the current position, 'T:xxxxx' provide you with information

on free memory, (i.e. you can still enter 'xxxxx' characters). 'C:yyyy' offers information on the free space of the copy register (the so-called C-register) which consists of 1024 characters when it is empty. The last two characters in the status bar display a status message from the editor. If everything is fine, you'll find 'OK' displayed there. Any error or status messages (e.g. if the copy register is open) is shown here. A list of possible messages can be found in Appendix B.

1.1.2 Text window

The next 21 lines are the text window, where you always see the current text, which you can edit with the editing functions discussed in the following sections. In the truest sense of the word, this screen must be understood as a window into the text, which can be moved up and down over the text with e.g. the cursor keys. All editing is immediately visible in the text window.

1.1.3 Command line

The last line of the screen is reserved for entering commands. Besides a number of direct commands that are given directly in the text window, there is also another command mode of the editor, which is activated by pressing the **[ESC]** key. Here you enter commands that require additional information, such as a word to look for. Commands for loading and saving of text can be entered here, too.

1.2 Text input

You can immediately start entering text, without any line numbers. Each character is stored at the current cursor position and the cursor moves right one position. You can always start a new line by pressing **[RETURN]**. If you enter more than 38 characters in a row, the cursor disappears from the right side of the screen similar to pressing the **[RETURN]** key and appears at the beginning of the next line again. This behavior seems unusual at first glance, but you will soon find that in assembly language programs this is quite an advantage. The essential part of an assembler program is always in the first 30 characters, because the assembler instruction is there. When you want to edit long lines (e.g. comments), there is a special function available (**[CTRL-V]**, see section XXXXX on special features), giving you a maximum of 76 characters on two lines of the screen. You will notice, however, that the normal 38-character screen is much better for editing assembly programs, because all assembly instructions, contrary to the two-line mode, will not be interrupted by blank lines between them.

As usual, all the keys have the auto-repeat function. Lower case can be achieved with the **[CAPS]** key.

1.3 Edit commands in text mode

As mentioned above, the ATMAS-II editor offers two different levels of command input. The simpler of the two are the so-called direct commands, also

called commands in text mode. These commands are executed either by special keys, for example `BACKSPACE`, or control functions. The latter simply means that the `CTRL` key (For 600/800XL: `CONTROL`) has to be hold down while you press another key, similar to how you move the cursor in ATARI BASIC. If you read the following: `CTRL-X`, it means that you are holding down `CTRL` while pressing the `X` key.

1.3.1 Cursor movement

In text mode, you have the following cursor functions:

To move the cursor right, press `CTRL-→`.

A cursor movement to the left is done with `CTRL-←`. When you get to the right end of a line, the cursor is moved to the beginning of the next line. Go beyond the left end, and the cursor is moved to the end of the previous line.

To move the cursor up, press `CTRL-↑`. The cursor is, if it is not already there, moved to the left margin. If that is this case, it is the second key press that moves one line up.

The cursor can be moved to the line below to the beginning of the next line by pressing `CTRL-↓`. With all cursor movements, you can not go beyond the beginning and end of the text. Unlike other editors, you will notice that the cursor is not on the text, but is inserted. This has the advantage that you always know exactly where the text will be inserted.

1.3.2 Deleting characters

To delete a character left of the cursor, press `BACKSPACE`. When the cursor is at the beginning of the line, the return character is deleted, i.e. the line where the cursor was is appended to the previous one. If this was a blank line and consisted solely of a return character, the blank line is deleted.

To delete a character right of the cursor, press `CTRL-BACKSPACE`. Just like the above case, the next line can be appended to the current line. If this is a blank line, it is deleted.

1.3.3 Deleting lines

If you want to delete a line, you can press `CTRL-X`. All characters from the current cursor position till the end of the line will be deleted. If the cursor is at the beginning of a line, the whole line will be deleted.

1.3.4 Cursor jumps

`CTRL-E` brings you to the beginning of the text.

`CTRL-D` brings you to the end of the text.

1.3.5 Tabulator

The ATMAS-Editor has a real Tabulator that moves the cursor between 1 and 9 spaces, but internally only occupies 1 character. To reach the next tabulated point, simply press `TAB`.

If you later move your cursor around a tab location, you will notice that the cursor jumps and inserts before a tab do not move the text after it. The tab is stored as a single character, hence you can use it to create readable listings without wasting memory space.

1.3.6 Copy-Register (C-Register)

The C-Register is a secondary text buffer which has a size of 1024 characters. You can use it to move or copy parts of your text. First, position the cursor at the end of the block you want to transfer and open the C-Register by pressing **CTRL-R**.

At the left of the status bar, this mode is denoted with 'CR'. If you move the cursor backwards, the text you walk-over will be copied to the C-Register buffer. You can follow its progress. The number of free characters in the C-Register ('C:1024') decreases with every cursor movement.

When you're finished, you can close the C-Register by pressing **CTRL-F**.

Now you can paste the contents of the Copy-Register wherever you want by pressing **CTRL-J**. The contents of the Copy-Register are preserved, so you can make multiple copies of the same text. To clear the Copy-Register, press **CTRL-K**. The C-Register is also cleared if you run the assembler or any I/O takes place.

Moving, instead of copying, a block of text can be done by filling the C-Register buffer using **BACKSPACE** or **CTRL-X** instead of **CTRL-↑** or **CTRL-←**. The deleted text now resides in the C-Register and can be inserted at another location.

You can also combine several pieces of text by reopening and closing the C-Register without clearing it in between with **CTRL-K**.

When the C-Register is full ('C:0000'), the error message 'C?' will appear at the left side.

1.3.7 Special commands for the editor

You can switch to two-line mode by pressing **CTRL-V**. In this mode, lines that are longer than 38 characters are displayed on two lines. The maximum line length is increased to 76 character, but the number of lines displayed is decreased to 11. Pressing **CTRL-V** again brings you back to single-line mode.

Control characters, especially tabs and end-of-line characters, can be made visible by pressing **CTRL-T**. All control characters are shown in inverse video. Pressing **CTRL-T** again brings back the normal mode.

CTRL-G repeats the command present on the command line. See section 1.4 for more information.

1.3.8 Calling the assembler and the monitor

The **ATMAS-II** macro assembler is called by pressing **CTRL-Y**. Immediately, **ATMAS-II** begins assembling the source code that resides in the text buffer. When it is finished, you can return to the editor by pressing a key.

The machine code monitor can be called by pressing **CTRL-P**. 'MONITOR.' is shown and it awaits your input (see section 3). Returning to the editor can be accomplished by inputting 'E'.

1.4 Commands on the command line

In addition to the direct commands in text mode, the ATMAS-II editor has another powerful editing mode! The command line. You will find primarily commands here that need more information than the actual call of the command, such as a character string to be searched for. You will also find some commands that you know from the text mode here, but these can be provided with repetition factors and chained. A sort of 'command macros'.

1.4.1 Using the command line

You enter the command mode by pressing the `[ESC]` key, which is represented as a dollar sign. Now you can enter individual commands while using the `[ESC]` key as a delimiter. The command line is executed when you press `[ESC]` twice. The execution is acknowledged with a double cross behind the chain of commands, you then move back to text mode. If the command line has proven to be faulty, you will be told so in the upper left corner by an error symbol.

All commands (except the I/O commands R and W) can be performed repeatedly by preceding it with a repetition factor between 2 and 255. In addition, a single repetition of a complete command line can be done by pressing `[CTRL-G]` from the text mode.

Errors in the command line itself can be corrected with `[BACKSPACE]`. `[CTRL-X]` will delete the command line and return to text mode.

1.4.2 Edit functions on the command line



- B Cursor one position backwards
- F Cursor one position forwards
- D delete one character left from the cursor
- T delete from cursor position until the end of the line

With these commands, for example, rapid cursor movement in the text is possible. Open the command line with `[ESC]` and type '200F' followed by pressing `[ESC]` twice. Each repetition of the command line with `[CTRL-G]` will bring you 200 characters further in the text buffer.

1.4.3 More editing commands

- H**<HexByte> Insert any ATASCII-code in the text, e.g. for printer control. EXAMPLE: `$H0F$` switches an EPSON printer to condensed
- S**<String> Search string from current position to the end of the text. If it's not found, 'S?' will appear in the status bar. EXAMPLE: `$SLABEL$` searches for LABEL.
- !**<String> Inserts string <String> at the current cursor position. Can be used as a search and replace function (see below) in conjunction with the S command.




1.4.4 Special functions


- J Repeat the complete command line. Can, for example, be used for global search and replace.
- K Clears the entire text buffer. CAUTION: All entered text will be deleted!
- U USER Command. Call a machine code program located at \$A800 with the JSR instruction. CAUTION: be sure there's an actual program at that location.
- ⓪<n> Set the tab-width to <n> characters. If none is specified, it is set to 8. EXAMPLE: \$⓪6\$: set tab-width to 6 characters.
- M Return to DOS. You can return to **ATMAS-II** with 'B - * **ATMAS-II** *' from the DOS menu.
- Two more commands that affect the C register:
- E Clear the C-Register (like )
- G Insert C-Register contents in the text (like )


1.4.5 Commands for in-/output of text

If you want to save a text present in the text buffer to disk, you can do this with the 'W'-command. You only need to specify the desired name of the file preceded by a device name.

EXAMPLE: If you want to save the text as **TEST.SRC** on drive 1, you enter:

 **WD1:TEST**  

The extension **.SRC** is appended automatically. You can also specify a different extension manually, like **\$WD1:TEST.ATM\$** .




CAUTION: The text will be saved starting at the *current cursor position*. Hence, if you want to save the full text, you need to press  first.

In a similar way, you can load a text file from disk to memory. For this, the command 'R' is used, which is used like 'W'.

EXAMPLE: To load the demo file **DEMO.SRC** of the **ATMAS-II** diskette (in drive 1):

 **RD1:DEMO**  

The extension **.SRC** will again be appended automatically. The file is then loaded starting at the current cursor position. This gives two possibilities:

1. When a new file is to be loaded, then delete the previous text buffer in advance with  **K**  .
2. You can also insert individual files in the text buffer (similar to the copy register) by bringing the cursor to the desired position and then execute the load command.

If an error occurs during an I/O command, this is displayed in the status line with the text 'RW'.


1.4.6 Listings

Printouts of the source code can be made with the 'L' command. There are several options:

- L Listing scrolls across the screen
- L0 Output to RS232 Interface # 1 of the 850 interface modules (R1:)
- L1 Output is via the normal Printer-handler 'P:', that is usually on the Atari 850 interface module, too.
- L7 Listing is output via the Centronics interface on the joystick ports 3 and 4. This option is available only for the Atari 400/800 computer.

1.4.7 Notes on advanced editor use

The possibility of command chaining at the command line allows very convenient editing features:

`$SLDA$3D$ILDX$J$` 

This chain of commands will exchange all LDA commands from the cursor position to LDX commands. How does it work? First, an LDA instruction is sought (\$SLDA). After that, the cursor can then be thought to be positioned behind the string searched for. '\$3D' now deletes three characters to the left of the cursor (remember the repetition factor!), while '\$ILDX' inserts the string 'LDX'.

'texttt \$J' in turn causes the entire command line to be executed until the text is finished, or a 'S?' error (String not found) occurs.

The macro assembler

2.1 Using the macro assembler

From within the editor the ATMAS-II macro assembler is invoked with `CTRL-Y`. It then immediately begins assembling the source code, which must reside in the text buffer of the editor.

The assembly is divided into three passes. The second pass indicates its activity by a rapidly changing character in the upper left corner of the screen. The third pass is a listing of the assembled program on the printer or the screen output (see OUT-directive).

The process of assembling is continued until either the source is over, a Ctrl-Z character is detected (assembler stop character) or an error has been detected. In the latter case, an error message (see Appendix B) is displayed on the screen. Any key will take you back to the editor, exactly at the place that caused the error. You can immediately improve the error and restart the assembler with `CTRL-Y`.

This comfortable 'hand-in-hand' work between editor and assembler is a key point that makes ATMAS-II so powerful as a development system for machine language programs, and yet it makes the operation easy.

2.2 Input format of the assembler

The ATMAS-II macro assembler knows all the opcodes of the 6502 CPU as far as they are described in the '*MOS Technology 6502 Programming Guide*'. Similarly, the notation of the addressing modes follows what is proposed in that book (see below).

A line of code can have the following formats:

- a) Blank line: Consists only of a `RETURN` character.
- b) Comment line: Starts with a ' texttt *' in the first column.

Example:

```
* Copyright (c) M. Huber
```

- c) Command line with label: Starts with a label followed by a delimiter (space or tab), then a 6502 opcode or assembler directive, and finally, after a further delimiter, it can be followed by a comment.

Example:

```
LOOP    LDA #$10
DOSVEC  EQU $000C          DOS-VECTOR
```

- d) Command Line: Must always start with a delimiter. To get clean Listings it is recommended to have these begin with a tab. This is followed by a 6502 opcode or assembler directive. As in the previous case, a comment can be here after another delimiter.

Example:

```
        STA COLOR          CHANGE COLOR
        DFB 100,120,140
```

2.2.1 Addressing modes

Im folgenden eine kurze Zusammenfassung der Schreibweisen der einzelnen Adressierungsarten, jeweils mit einem Beispiel versehen:

Implied Akku	<Opcode>	ASL
Immediate	<Opcode> # <Ausdruck>	LDA #\$FF
Absolut, Zeropage	<Opcode> <Ausdruck>	STA \$600
Relativ	<Opcode> <Ausdruck>	BNE ++4
Absolut X-indiziert	<Opcode> <Ausdruck>,X	CMP \$3000,X
Absolut Y-indiziert	<Opcode> <Ausdruck>,Y	LDA TABLE,Y
Indirekt-indiziert	<Opcode> (<Ausdruck>),Y	EOR (\$F0),Y
Indiziert-indirekt	<Opcode> (<Ausdruck>,X)	STA (\$F0,X)

Wie schon erwähnt, ist vor jedem <Opcode> ein Label möglich, nach jedem Assemblerbefehl kann ein Kommentar stehen, der mit einem Trennzeichen abgesetzt ist.

AUSNAHME: Nach implied-Akku Befehlen muß das Kommentarfeld durch einen Strichpunkt abgetrennt sein.

2.2.2 Labels

Labels bestehen aus Buchstaben und Zahlen, wobei das erste Zeichen immer ein Buchstabe sein muß. Die maximale Länge beträgt 8 Zeichen, wobei alle Zeichen signifikant sind.

Gültige Labels	Ungültige Labels
SCHLEIF	1LOOP
SCHLEIF1	SCHLEIFEN (9 Buchstaben!)
S12345	

2.2.3 Konstante

ATMAS-II kennt vier Arten von Konstanten: Dezimal-, Hexadezimal-, Binär- und Zeichen-Konstante.

a) Dezimale Konstante

Bestehen einfach aus der Zahl selbst (im Bereich 0 bis 65535).

Beispiele: 10, 100, 3000, 65535

b) Hexadezimale Konstante

Bestehen aus einem Dollar-Zeichen ('\$') und gültigen Hexadezimalziffern (0-9,A-F).

Beispiele: \$10, \$FF, \$AB00, \$FFFF

c) Binäre Konstante

Bestehen aus einem Prozent-Symbol mit nachfolgenden gültigen Binärziffern (0,1).

Beispiele: %1, %1001, %11110000

d) Zeichen-Konstante

Werden aus einem Apostroph mit nachfolgenden ASCII (bzw. ATASCII)-Zeichen gebildet. Die Zeichen werden in 7-Bit-Werte umgewandelt (Bit 7 immer Null).

Beispiele: 'X', '#', 'e', '"

2.2.4 Negative Konstante

Alle Konstanten können durch ein vorangestelltes Minus-Zeichen als negative Zahlen interpretiert werden. Die Darstellung erfolgt im Zweierkomplement ($-1 = \$FFFF$).

Beispiele: -1, -\$100, -%101101, -'B'

2.2.5 Interner Adresszähler

ATMAS-II führt während der Assemblierung einen internen Adresszähler mit, der jeweils auf den Upcode, der gerade übersetzten Anweisung zeigt (bzw. auf das nächste freie Byte nach dem zuletzt berechneten Ausdruck in DFB/DFW Direktiven). Dieser interne Adresszähler kann jederzeit durch das '*'-Symbol abgerufen werden. Dies kann in vielfältigen Weisen benutzt werden:

a Speicherplatz reservieren: `ORG **$100` reserviert 256 Bytes

b relative Sprünge ohne Label: `BNE **4` überspringt die nächsten 4 Bytes

c Adressversetzte Labels für Software-Stacking: `RETADR EQU *-1` momentane Adresse minus 1

2.2.6 Ausdruck

Ein Ausdruck kann eine Konstante, ein Label, ein Makroparameter (siehe 2.4), der momentane Adress-Zähler ('*') oder eine arithmetische Verknüpfung derselben sein. Zugelassene Verknüpfungen sind: +, -, *, /, der Rechenbereich beträgt 0-65535. Einige Beispiele dazu:

LDA #FF	Konstante
LDA #LABEL1	Label
STA LABEL+\$A0	Verknüpfung
LDA *+\$10	momentaner Adr.-Zähler+Offset

Zusätzlich sind auch Klammern zur Klärung der Priorität möglich:

```
LDA #($A0+5)/10
LDA #LAB1-(Lab1/256)*256
```

2.3 Assembler-Direktiven

2.3.1 ORG

Zweck: bestimmt den Inhalt des assemblerinternen Adress-Zählers.

Syntax: [*<label>*] ORG *<AL>*[,*<AP>*] [*<kom>*]

Beispiel: ORG \$A800

ORG bestimmt die Anfangsadresse des Objektprogrammes, im obigen Beispiel ist der für diese Zwecke reservierte Speicherbereich ab \$A800 gewählt. In manchen Fällen ist es aber nicht möglich, das Objektprogramm direkt in den gewünschten Speicherbereich zu legen, wenn dieser 2.8. von ATMAS-II belegt wird (s. Memory-Map im Anhang!). Hierzu können Sie den zweiten Parameter der ORG-Direktive verwenden, die sogenannte 'physikalische Adresse' *<AP>*.

```
ORG $3000,$A800
```

Dieser ORG-Befehl würde ein Programm so assemblieren, daß es an der Adresse \$3000 (logische Adresse, *<AL>*) lauffähig ist, während der Assemblierung aber im freien Speicherbereich ab der Adresse \$A800 abgelegt wird. Damit das Programm ausgeführt werden kann, muß es an die richtige Stelle verschoben werden, das kann 2.8. mit dem SAVE-Befehl des Monitors geschehen, der ein Binärfile so abspeichern kann, daß es beim erneuten Einlesen an den richtigen Speicherplatz kommt. In einfacheren Fällen (2.8. wenn das Programm in dem Adressbereich abgelegt werden soll, in dem ATMAS-II seine Symboltabelle hat) hilft auch der Blocktransfer-Befehl des Monitors.

2.3.2 EQU, EPZ (Equates)

Zweck: Zuordnung eines Wertes zu einem Label.

Syntax: `<label> EQU <ausdruck> [<kom>]`
`<label> EPZ <ausdruck> [<kom>]`

Beispiel: `GRUEN EQU $A0`
`CIOV EQU $E456`
`SAVHSC EPZ $58`
`PLAYER EQU PHBASE+1024`

Die EQU-Direktive wird benutzt, um einen Label mit einem bestimmten Wert zu definieren, es kann sich dabei sowohl um ein Datum oder eine Adresse handeln. Der erste Label des Beispiels wird z.B. sicher als Datum für einen immediate-Befehl verwendet werden (LDA #GRUEN), während das zweite Beispiel eine ROM-Einsprungadresse bezeichnet (JSR CIOV).

EPZ (Equate Page Zero) hat prinzipiell die gleiche Funktion, kann aber verwendet werden, wenn zum Ausdruck gebracht werden soll, daß der Label einen Zeropage-Speicherplatz darstellt. Es besteht aber kein Zwang zur Verwendung von EPZ, da ATMAS-II Zeropage-Adressierungsarten selbsttätig erkennt. EPZ dient hierbei nur der Verdeutlichung, Sie können in allen Fällen auch EQU verwenden.

2.3.3 DFB (Define Byte)

Zweck: definiert den Inhalt einzelner Bytes im Objectcode

Syntax: `[<label>] DFB]fooausdruck[,<ausdruck>][<kom>]`

Beispiel: `TABELL DFB 1,2,4,8,16,32,64,128`
`DFB 'A, 'B, 'C`
`DFB 'X+128`

Die nach DFB folgenden Ausdrücke werden als 8-Bit werte in den Objektcode abgelegt. Die erste Zeile des Beispiels würde folgende Sequenz erzeugen:

01 02 04 08 10 20 40 80

ATASCII werte werden als 7-Bit Nerte abgelegt, das höchstwertige Bit ist immer Null.

2.3.4 DFW (Define Word)

Zweck: definiert den Inhalt eines 16-Bit Wortes im Objektcode.

Syntax: `[<label>] DFW <ausdruck>[,<ausdruck>...][<kom>]`

Beispiel: `ADRTAB DFW $A900,$AA03,$AB06`
`DFW ADRTAB-1,ADRTAB+2`

Die nach DFW foldenden Ausdrücke werden als 16-Bit Werte im Objektcode abgelegt. Dabei wird die Reihenfolge der 6502-Adressen benutzt, d.h. zuerst das niederwertige (LSB), dann das höherwertige Byte (MSB). Die erste Zeile des Beispiels würde demnach folgendes erzeugen:

```
00 A9 03 AA 06 AB
```

2.3.5 ASC (ASCII-String)

Zweck: fügt einen String in ATASCII- oder Bildschirmcodierung in den Objektcode ein.


Syntax: [*<label>*] ASC *<STZ>**<string>**<STZ>*[*<STZ>**<string>**<STZ>*][*<kom>*]

Beispiel: TEXT ASC "HALLO"
 MELD ASC /DISK-FEHLER/
 ASC /MAKRO/, \ASSEMBLER\
 ASC Z Normaler Bildschirmcode Z
 ASC S Inverser Bildschirmcode S

Strings nach einer ASC-Direktive werden Byte für Byte in den Objektcode übertragen. Abhängig vom Trennzeichen (*<STZ>*) können verschiedene Funktionen angewählt werden:

- " Text im ATASCII-Code eintragen
- / ATASCII-Code, aber Bit 7=1 (Invers)
- \ ATASCII-Code, beim letzten Zeichen wird Bit 7 gesetzt (Ende-Kennung)
- % interner Bildschirmcode wird generiert
- \$ Bildschirmcode mit Bit 7=1 (Invers)

Jeder String muß von zwei gültigen String-Trennzeichen umschlossen werden. Anfangs- und Endtrennzeichen müssen gleich sein.

Statt dem Anführungszeichen " () können Sie auch ein beliebiges anderes nicht alphanumerisches Zeichen benutzen, die oben genannten natürlich ausgeschlossen. Denkbar wären hier z. B. !, #, oder &, Bedingung ist lediglich, daß der String mit dem gleichen Trennzeichen abgeschlossen wird. Mit diesem Trick ist es möglich, daß Sie ein Anführungszeichen in den Text bekommen.

In Verbindung mit dem Displaylistkonzept der Atari- Computer gestattet Ihnen der ASC-Befehl die komfortable Programmierung von Titeln und Überschriften, da direkter Bildschirmcode erzeugt werden kann. Das Beispielprogramm ASCDEMO.SRC auf der ATMAS-II Diskette zeigt Ihnen, wie's gemacht wird.

2.3.6 OUT (Output Listing)

Zweck: Ausgabe eines Assembler-Listings

Syntax: [*<label>*] OUT [L][N][M][P0][P1][P2][*<kom>*]

Beispiel: OUT L
OUT LP1
OUT LNMP1

Mit OUT können Sie bestimmen, ob Sie ein Protokoll des Assembliervorganges haben möchten und das zugehörige Ausgabegerät festlegen. Folgende Parameter werden erkannt:

L Listing wird erzeugt
N Symboltabelle wird ausgegeben
M Makros werden NICHT expandiert (ausgedruckt!)
P0 RS232 Schnittstelle
P1 Atari-Drucker, Centonics-Schnittstelle (850)
P2 Joystick-Interface Port 3 und 4 (nur 400/800)

Nach jeweils 66 Zeilen wird ein Seitenvorschub generiert.

2.4 Mahrfähigkeit

ATMAS-II gestattet Ihnen die Verwendung von Makrobefehlen. Darunter versteht man eine Folge von Assemblerbefehlen, denen im Rahmen einer Makrodefinition ein Name zugeordnet wurde. Bei der Verwendung dieses Makro-Namens im Quelltext werden die gesamten, ihm zugeordneten Assemblerbefehle erzeugt, man sagt, der Makro-Befehl wird expandiert. Mit Hilfe von Makros können Sie sich eine Art von zusätzlichen Assemblerbefehlen schaffen, die in Wirklichkeit aus einer Sequenz von einzelnen Maschinenbefehlen bestehen.

2.4.1 Makro-Definition

Bevor Sie ein Makro verwenden können, müssen Sie es zuerst definieren. Zu diesem Zweck dienen die zwei Assembler-Direktiven **MACRO** und **MEND**.

Zum besseren Verständnis zuerst ein Beispiel, das den Basic-Befehl POKE imitiert:

```
POKE    MACRO ADDRESS,DATA
        LDA #DATA
        STA ADDRESS
MEND
```

Die Definition wird von der **MACRO**-Direktive mit voranstehendem Namen des Makros (hier: POKE) eingeleitet. Dieser Name wird später zum Aufruf

verwendet. Nach der **MACRO**-Direktive folgen die sogenannten formalen Parameter, das sind keine Labels im eigentlichen Sinn, sondern nur Platzhalter für die beim Aufruf angegebenen tatsächlich einzusetzenden Parameter. Dieser Vorgang der Parametersubstitution wird bei der Besprechung des Makro-Aufrufes noch genauer erläutert.

Jetzt folgen die Assemblerbefehle, die als Operanten sowohl gewöhnliche Labels als auch die in der **MACRO**-Direktive angegebenen formalen Parameter benutzen können. Abgeschlossen wird die Makro-Definition durch die **MEND** (Makro-Ende)-Direktive

```
Syntax:  <Makroname> MACRO [<param>][,<param>...]
          ...
          MEND
```

2.4.2 Makroaufruf, Makroexpansion

Das in 2.4.1 definierte Makro kann im Quelltext mit

```
POKE 752,1
```

aufgerufen werden. Bei der Assemblierung findet dann folgendes statt: **AT-MAS-II** erkennt, daß es sich bei **POKE** um ein bereits definiertes Makro handelt und fügt die Maschinenbefehle der Definition in den Objektcode ein. Dabei werden die formalen Parameter der Definition mit den tatsächlichen Werten des Aufrufes ersetzt. Im Beispiel wird folglich **ADDRESS** durch **752**, und **DATA** durch **1** ersetzt. Nach der Makroexpansion ergibt sich folgendes Maschinenprogramm:

```
LDA #1
STA 752
```

Die allgemeine Syntax für den Makroaufruf lautet:

```
[<label>] <Makroname> [<param>][,<PAR>...]
```

Es müssen ebensoviele Parameter (*<param>*) übergeben werden, wie formale Parameter in der Definition angegeben wurden. Als Parameter können Konstante, Ausdrücke und auch Strings verwendet werden. Ein Beispiel zur Verwendung von Strings als Parameter finden Sie im Teil 4.2 (Makro-Bibliotheken).


2.4.3 Lokale Labels

Als nächstes Beispiel betrachten Sie bitte das Listing des unten angegebenen Makros. Es handelt sich dabei um ein Programm zum Löschen eines Speicherbereiches von max. 256 Bytes (einer Page). Da das Programm mit einer Schleife arbeitet, enthält es folgerichtig auch einen Label. Hürden Sie dieses Makro zweimal innerhalb eines Programmes aufrufen, so wäre das Label **LOOP** doppelt verwendet, wodurch der Assembler den Fehler 'SAME LABEL TWICE' meldet.


```

LOESCH  MACRO  ADRESS,LAENGE
        LDY  #LANGE
        LDX  #0
        LDA  #0
LOOP    STA  ADRESS,X
        INX
        DEY
        BNE  LOOP
        MEND

```

Auch hier bietet Ihnen **ATMAS-II** Unterstützung an: Ein Label, das mit dem '@' Symbol () endet, wird als lokales Label des Makros angesehen. Im Beispiel: statt **LOOP** muß der Label **LOOP@** verwendet werden (auch im **BNE**-Befehl!). Intern ersetzt **ATMAS-II** das '@' Symbol durch eine vierstellige Zahl, die bei jedem Makroaufruf um eins erhöht wird. Dadurch erzeugt auch zweimalige Aufruf von **LOESCH** verschiedene Labels (nämlich **LOOP0001** und **LOOP0002**).

2.4.4 Verschachtelte Makros

Eine Makrodefinition selbst kann wiederum einen Makro- aufruf einschließlich der Übergabe von Parametern enthalten. Als Beispiel kann ein Double-Poke Befehl dienen, der das bereits besprochene **POKE**-Makro benutzt:

```

DPOKE  MACRO  ADRESS,WORT
        POKE  ADRESS,WORT
        POKE  ADRESS+1,WORT/256
        MEND

```

Diese Verschachtelungstechnik ist nur durch den Hardwarestack begrenzt.

2.4.5 Makros contra Unterprogramme

Sicherlich ist Ihnen bei der Beschreibung der Makros die nahe Verwandtschaft zu Unterprogrammen aufgefallen. Es gibt jedoch eine Reihe von wichtigen Unterschieden, die Sie sich verdeutlichen sollten:

Makros sind universeller in der Benutzung, da Sie über den Mechanismus der Parameterübergabe verfügen. Sie eignen sich daher gut zum Aufbau von Makro- (Programm) Bibliotheken. Solche Makrosammlungen lassen sich leicht mit dem flexiblen **ATMAS-II** Editor in den Quelltext einbinden. Makros gestatten wesentlich übersichtlichere Assemblerprogramme, man darf allerdings nicht übersehen, daß alle in der Definition des Makros angegebenen Maschinenbefehle bei jedem Aufruf des Makros in das Programm eingesetzt werden. Das bedeutet, wenn Sie das **LOESCH**-Makro fünfmal im Programm verwenden, daß es ebenso oft in Ihr Programm eingesetzt wird, und natürlich auch dementsprechend Speicherplatz verbraucht.

Die Verwendung von Unterprogrammen ist wesentlich optimaler in Bezug auf Speicherplatz, wobei es allerdings schwieriger ist, die übergabe der Parameter so elegant wie im Makro zu gestalten. Ein weiterer, in manchen Fällen entscheidender Gesichtspunkt ist die Geschwindigkeit: Während Makros hier

die bessere Lösung darstellen, dauert es bei Unterprogrammen etwas länger, da JSR und RTS-Befehle auch Zeit benötigen.

CHAPTER 3

Der Maschinensprache-Monitor

Vom Editor gelangen Sie durch die Eingabe von `CTRL-P` in den Maschinensprache-Monitor. Der Bildschirm wird gelöscht und das "MONITOR."-Prompt erscheint in der linken oberen Bildschirmecke. Mit dem ATMAS-II Monitor können Sie Maschinenprogramme starten, auf Diskette ablegen, den Speicherinhalt prüfen, verändern und disassemblieren und dabei ein Protokoll am Drucker mitführen. Da der Monitor über eine dialogorientierte Eingabe verfügt, brauchen Sie sich keinerlei komplizierte Befehlssyntax merken.

3.1 Allgemeine Benutzungshinweise

Alle Eingaben innerhalb des Monitors erfolgen in hexadezimaler Schreibweise (ohne vorangestelltes Dollarzeichen!). Sollten Sie sich bei der Eingabe vertippen, so kann diese jederzeit mit 'X' abgebrochen werden. Wenn Fragen im Dialog auftreten, werden diese mit 'Y' (Yes) positiv beantwortet, jeder andere Tastendruck (auch `RETURN`) beantwortet die Frage verneinend. Auch hier ist ein Abbruch mit 'X' möglich. Filenamen müssen immer im Standard-Atari Format eingegeben werden, d.h. zuerst einen Gerätenamen (D:, D1:, D2:...) dann der Filename mit max. 8 Zeichen, der nach einem Dezimalpunkt noch eine max. 3 Zeichen lange Erweiterung haben darf.

Beispiele: D:TEST.OBJ, D2:CODE.COM, D1:FILE


3.2 Befehlsvorat des Monitors


Die Befehle des Maschinensprache-Monitors bestehen aus einfachen Buchstaben, die Parameter, soweit nötig, werden im Dialog abgefragt.

3.2.1 M - Memory-Dump

Hit dem M-Befehl können Sie einen Speicherbereich in hexaderimaler Schreibweise auf Bildschirm oder Drucker ausgehen, weiterhin können Sie eine zusätzliche Darstellung den Speicherinhaltes in ASCII-Zeichen erhalten.

Beispiel:

Ihre Eingabe	Bildschirm
M	DUMP
D000	FROM:D000
D080	TO:D080
Y	ASCII?Y
	PRINT?


Nun wird der Speicherbereich von \$D000-\$D080 sowohl in hexadezimaler Schreibweise als auch in ASCII ausgegeben. Hätten Sie bei der Frage ASCII? einfach  gedrückt, so würde nur die hexadezimale Schreibweise ausgegeben. Antworten Sie auf die Frage PRINT? mit 'Y', dann müssen Sie sich für einen Ausgabekanal entscheiden:


- (1) wählt die serielle Schnittstelle R1: des ATARI-Interfaces als Ausgabekanal
- (2) Ausgabekanal ist die Centronics-Schnittstelle des ATARI-Interfaces (normaler 'P':-Printer-Handler).
- (3) Ausgabe über Joystickinterface Port 3 & 4 (400/800)

3.2.2 D - Disassemble

Mit dem D-Befehl können Sie den Inhalt eines Speicherbereiches als 6502-Befehle rückübersetzen (disassemblieren) lassen. Wie bei M können Sie die Ausgabe auf den Drucker umlenken.

Beispiel:

Ihre Eingabe	Bildschirm
D	DISASSEMBLER
D000	START?D000
	PRINT?

Jetzt wird der Speicherinhalt ab der Adresse \$D000 in disassemblierter Form ausgegeben, wobei immer nach einer Füllung des Bildschirmes unterbrochen wird. Durch Drücken einer beliebigen Taste (außer 'X') wird der nächste Bildschirm ausgegeben. 'X' beendet die Disassemblierung. Bei Ausgabe auf Drucker findet keine Unterbrechung des Listings statt, der Ausdruck kann mit  abgebrochen werden.

3.2.3 C - Change Memory

Das C-Kommando erlaubt Ihnen die Veränderung von Speicherinhalten. Die Eingabe erfolgt in hexadezimaler Schreibweise.

Beispiel:

Ihre Eingabe	Bildschirm
C	CHANGE
A900	ADDRESS?A900
FF	A900 => FF
00	A901 => 00
X	A902 => MONITOR.

Sie können gezielt einzelne Bytes oder auch zusammenhängende Speicherblöcke in hexadezimaler Form eingeben. Durch die Eingabe von 'X' kommen Sie wieder in den Befehls-Eingabemodus des Monitors zurück.

3.2.4 F - Fill Memory

Mit dem F-Befehl können Sie einen Speicherbereich mit einem gewünschten Wert vorbesetzen.

Beispiel:

Ihre Eingabe	Bildschirm
F	FILL
A900	FROM:A900
AFFF	TO:AFFF
FF	WITH:FF

Wirkung: Der Speicherbereich \$A900 bis \$AFFF wird mit dem Wert \$FF gefüllt.

3.2.5 B - Blocktransfer

Das B-Kommando erlaubt die Verschiebung ganzer Speicherblöcke. Dies kann nützlich sein, wenn Sie bei der Assemblierung mit **ATMAS-II** eine unterschiedliche logische und physikalische Adresse gewählt haben. Der B-Befehl benötigt Anfangs- und Endadresse des zu verschiebenden Bereiches, sowie die Anfangsadresse des Zielbereiches.

Beispiel:

Ihre Eingabe	Bildschirm
B	BLOCKTRANSFER
A800	FROM:A800
A900	TO:A900
AC00	INTO:AC00

Wirkung: Der Speicherblock von \$A800 bis \$A900 wird in den Speicherbereich von \$AC00 bis \$AD00 kopiert.

3.2.6 G - Goto Address

Mit dem G-Befehl können Sie ein Maschinenprogramm starten, dessen Einsprungsadresse Sie eingeben müssen.




Beispiel:

Ihre Eingabe	Bildschirm
G	GOTO
A800	GOTO A800


ACHTUNG: Sie müssen selbst dafür Sorge tragen, daß ein ausführbares Maschinenprogramm an der angegebenen Stelle steht!

Die Kontrolle wird an den Monitor zurückgegeben, wenn das Maschinenprogramm entweder mit einem RTS (Return from Subroutine) oder BRK (Break-Befehl, Hex-Byte \$00) endet. Im letzteren Fall bekommen Sie die Registerinhalte und die Prozessor-Flags angezeigt, eine hervorragende Möglichkeit, um ein Programm nach Fehlern zu durchsuchen.

3.2.7 S - Binary Save

Um ein vom Assembler erzeugtes Maschinenprogramm auf Diskette abzuspeichern, können Sie entweder ins DOS gehen (über Editor,  M  ) oder den Monitor-Befehl S benutzen. Der Monitor speichert Maschinenprogramme so ab, daß Sie vom DOS-II (oder dazu kompatiblen DOS-Versionen) wieder geladen werden können.

Beispiel:

Ihre Eingabe	Bildschirm
S	SAVE
A800	FROM:A800
AF00	TO:AF00
	INTO:
D:TEST.OBJ	FILENAME(D:FN.EXT)? D:TEST.OBJ

Dieses Beispiel bewirkt, daß der Speicherbereich von \$A800 bis \$AF00 als File TEST.OBJ im Laufwerk 1 abgespeichert wird. Der Filename muß immer im Standard- Atari Format eingegeben werden.

Der **SAVE**-Befehl des **ATMAS-II** Monitors hat noch einige Zusätze aufzuweisen, die über den Standard hinaus- gehen:

- a) Adressversetztes Abspeichern: Sie können ein Programm so abspeichern, daß es beim erneuten Einlesen in einen anderen Speicherbereich geladen wird. Das ist sehr nützlich, wenn Sie Programme mit getrennter logischer und physikalischer Adresse (s. **ORG**-Direktive) assembliert haben.

Sie müssen dazu bei den Fragen **FROM** bzw. **TO** den physikalen Adressbereich angeben (wo das Programm momentan abgelegt wurde) und bei der Frage **INTO** die logische Adresse (an die das Programm geladen werden soll) angeben.


Beispiel:

Ihre Eingabe	Bildschirm
S	SAVE
A800	FROM:A800
A780	TO:A980
4000	INTO:4000
D:TEST.OBJ	FILENAME(D:FN.EXT)?D:TEST.OBJ

Dieses Beispiel würde ein Programm, das von \$A800 bis \$A980 im Speicher steht, so auf die Diskette schreiben, daß es beim erneuten Einlesen im Bereich von \$4000 bis \$4180 liegt. Hatten Sie das Programm mit **ORG \$4000,\$A800** assembliert, so ist es jetzt ein lauffähiges Maschinenprogramm. **ACHTUNG:** Sie sollten dieses File nicht mehr im Monitor einlesen, da sonst **ATMAS-II** überschrieben würde!

- b) Append-Option: Wenn Sie als letztes Zeichen des Filenames ein Größer-Zeichen ('>') eingeben, so wird das File an ein eventuell bereits bestehendes mit gleichem Filenamen angehängt. Sie können damit Compound-Files (aus mehreren Blöcken zusammengesetzte Files) oder Files mit **INIT** und **RUN**-Adresse erzeugen.

Beispiel:

Ihre Eingabe	Bidschirm
S	SAVE
AA00	FROM:AA00
AB00	TO:AB00
	INTO:
D:TEST-OBJ>	FILENAME (D:FN.EXT)?D:TEST.OBJ>

Das File **TEST.OBJ** (von vorhin) wird in diesem Beispiel um den Speicherblock von \$AA00 bis \$AB00 verlängert. Mit derselben Methode können Sie auch **RUN** und **INIT**-Adressen an ein File anfügen: Sie tragen die **RUN**-Adresse

(LSB, MSB) in die Adressen \$02E0, \$02E1 (INIT: \$02E2, \$02E3) ein und hängen den jeweiligen 'Speicher-block' (der nur aus zwei Bytes besteht FROM: 02E0 TO: 02E1) an das File an.

3.2.8 L - Binary Load

Analog zum Save-Befehl kann hier ein Binär-File von der Diskette geladen werden. Es kann sich dabei um ein vom Save-Befehl erzeugtes oder um ein DOS erzeugtes Binär-File handeln, auch zusammengesetzte Compound-Files werden geladen. RUN und INIT Sprünge werden nicht ausgeführt.

Beispiel:

Ihre Eingabe	Bildschirm
L	LOAD
D:TEST.OBJ	FILENAME(D:FN.EXT)? D:TEST.OBJ
	FROM: 5000 TO: 5190
	FROM: AA00 TO: AB00

Der Load-Befehl gibt Ihnen gleich an, in welchen Speicherbereich das Binär-File geladen wurde. Im Beispiel wurde ein File geladen, welches ähnlich zu dem im Save-Beispiel erzeugten ist.

3.2.9 E - Editor

Mit dem E-Kommando gelangen Sie zurück in den Editor, der Cursor befindet sich noch an der Stelle, wo Sie den Editor Verlassen haben.

3.2.10 I - Disketteninhaltsverzeichnis

Der I-Befehl zeigt Ihnen alle Filenamen des Laufwerks 1 auf dem Bildschirm an.

CHAPTER 4

Examples

4.1 Demo program, color scrolling

The following example should clarify the interaction of the individual components of `ATMAS-II`. It would be advantageous if chapter 1 (The editor) of this manual has been read and you are slightly familiar with the editor. Chapter 2 and 3 would not necessarily be required, but they will certainly contribute to better understanding.

Although the following demo program is included on the `ATMAS-II` disk (`DEMO.SRC`) you should still type it in manually in order to better familiarize yourself with the editor. When problems occur, you can use the included file as a fallback.

```
*****
* ATMAS-II Demo: Color Scrolling PF85
*
* Stop by pressing START
*****

COLPF2 EQU $D018      Color register
HSYNC  EQU $D40A      Horizontal sync
VCOUNI EQU $D40B      Rasterline
RTCLK  EQU $14         VBI Clock
CUNSDL EQU $D01F      function keys

*
* Program in USER-space
* (at $A800)
*

      ORG $A800

      LDA #8           Prepare polling the
```

```

          STA CONSOLD      START-key
SCRCOL   CLC
          LDA VCOUNT       Screen counter
          ADC RTCLK        plus raster line
          STA HSYNC        synchronise
          STA COLPF2       to color register
          LDA CONSOL       function keys
          AND #1           START-key?
          BNE SCRCOL       no, continue-->
          RTS

```

As a reminder, here are some tips on typing in the program: comment lines start with a star in the first column. Also have lines that are provided with a label starting at the first column. For all other lines, it is advantageous to use a tab at the beginning of the line.

Example:

The first 'EQU' line should be typed like this:

```
COLPF2 TAB EQU $D01B RETURN
```

After you have typed-in the whole program, you can begin assembling by pressing CTRL-Y. You see the copyright of the assembler and if the assembler has detected no error, it will display the end address of the generated object code.

If errors occurred, these are presented to you in plain text on the screen. You can change them using the error table in Appendix B to track the down. We only discuss typo's here.

First, you press any key to return to the editor. The cursor is now at the offending line. A small tip: Sometimes it helps by pressing CTRL-T to see if perhaps a tab sits at the wrong location (i.e. a line that has only one tab or has only spaces will be reported as a **SYNTAX ERROR**).

If the assembler does not report an error, you can go out to start the program you just assembled. To do this, you press any key to get back to the editor and then type CTRL-P to call to the monitor.

In the monitor you can view the program generated by the assembler. To do this, type 'D' (Disassemble) followed by 'texttt A800' and RETURN. You should now see a disassembled listing of the program on the screen. Now press 'X' to cancel the disassembler's listing.

To start the program type 'G' (GOTO) and the start address of the program, in our case 'A800'. If you did everything correctly, you should now see 128 colors running on the screen. The color spectacle can be terminated by pressing START or RESET. When you press RESET, you will automatically return to the editor. By pressing START, the program is properly terminated and control goes back to the monitor. 'E' finally reactivates the editor.

4.2 Macro libraries

On the ATMAS-II disk, you can find two files – IOLIB.SRC and GRAFLIB.SRC – whose source code is included on the following pages. Both are so-called macro libraries, which means that, except for a small demo program, they only include macro definitions. Both contain inline comments, so an in-depth description is not necessary.

GRAFLIB.SRC contains macros that behave similarly to their BASIC counterparts (GRAPHICS, COLOR, PLOT and DRAWTO). You can view the demo by loading and assembling GRAFLIB.SRC and by using the monitor to 'G'o to address \$A800.

IOLIB.SRC contains macros similar to the OPEN, CLOSE, PRINT and INPUT BASIC commands and two additional macros (BGET and BPUT) for loading and saving binary data. This library also contains an interesting demo, which can be run similarly as that of GRAFLIB.

The macro libraries contain examples of nesting macros and passing string parameters. Both files can be seen as examples of what can be done with ATMAS-II .

Obviously, it is allowed to extend and improve both libraries as you wish.

```
*****
*           GRAFLIB.SRC
*
*           Macro Library
*
*           GRAPHICS
*
*           For ATMAS-II
*
*                               PETER FINZEL
*****
*
* IOCB Structure
*
ICCOM      EQU $342
ICSTA      EQU $343
ICBAL      EQU $344
ICBAH      EQU $345
ICBLL      EQU $348
ICBLH      EQU $349
ICAX1      EQU $34A
ICAX2      EQU $34B

CIOV       EQU $E456

* CIO Commands

COPEN      EQU 3
CCLSE      EQU 12
CGTXT      EQU 5
CPTXT      EQU 9
CGBIN      EQU 7
```

```

CPBIN      EQU 11
CDRAW      EQU $11

* ATARI Graphics Variables

ATACHR      EBU $2FB
ROWCRS      EBU $54
COLCRS      EBU $55
CURSOR-
POSITION

*
* GRAPHICS-Command
*
* Call: GRAPHICS <mode>
*
* <mode>  0 bis 15 (XLs)
*          0 bis 11 (400/800)
*
GRAPHICS MACRO MODE
            JMP GR1@

DEV@      ASC 'S:'

GR1@      LDX #$60
            LDA #CCLSE          FIRST CLOSE CHANNEL 6
            STA ICCOM,X
            JSR CIOV
            LDA #MODE           SET NEW MODE
            STA ICAX2,X
            AND #$F0
            EOR #$10
            ORA #$0C
            STA ICAX1,X
            LDA #COPEN
            STA ICCOM,X
            LDA #DEV@
            STA ICBAL,X
            LDA #DEV@/256
            STA ICBAH,X
            JSR CIOV
            MEND

*
* Select drawing color
*
* Call: COLOR <color>
*
* <color> ranges from 0-255,
*          depending on graphics mode.
*          Must be constant
*

```

```

COLOR    MACRO COL
          LDA #COL
          STA ATACHR
          MEND

*
* Move cursor to position
*
* Call: POSITION <x>,<y>
*
* <x>,<y> depends on graphics mode
*         both must be constant
*
*
POSITION MACRO X,Y
          LDA #X
          STA COLCRS
          LDA #X/256
          STA COLCRS+1
          LDA #Y
          STA ROWCRS
          MEND

* Draw point
*
* Call: PLOT <x>,<y>
*
* <x>,<y> depends on graphics mode
*         must be constant
*
PLOT      MACRO X,Y
          POSITION X,Y
          LDX #$60          CHANNEL 6
          LDA #CPBIN
          STA ICCOH,X
          LDA #0
          STA ICBLH,X
          STA ICBLH,X
          LDA ATACHR
          JSR CIOV
          MEND

*
* Draw line
*
* Call: DRAWTO <x>,<y>
*
* <x>,<y> depends on graphics mode
*         must be constant
DRAWTO    MACRO X,Y
          POSTTION X,Y

```

```

        LDX #$60          CHANNEL 6
        LDA #CDRAH
        STA ICCOM,X
        LDA #CCLSE
        STA ICAX1,X
        LDA #0
        STA ICAX2,I
        JSR CIOV
        MEND

*****
* Demo-Program for Graphics Library
*
* draw diamond shape in GRAPHICS 7
*****

*
* assemble to memory reserved
* for object-code
*
        ORG $A800

        GRAPHICS 7+16
        COLOR 1
        PLOT 79,0
        DRAWTO 159,47
        DRAWTO 79,95
        DRAWTO 0,47
        DRAWTO 79,0

ENDLESS JMP ENDLESS
*
* Stop by pressing <RESET>
*

*****
*          IOLIB.SRC
*
*          MACRO LIBRARY
*
*          Input/Output
*
*          for ATMAS-II
*
*          by PETER FINZEL
*****

* IOCB Constants

CIOV    EQU $E456

```

```

ICCOM EQU $342
ICSTA EQU $343
ICBAL EQU $344
ICBAH EQU $345
ICBLH EQU $348
ICBLH EQU $349
ICAX1 EQU $34A
ICAX2 EQU $34B

```

* CIO Commands

```

COPEN EQU 3
CCLSE EQU 12
CGTXT EQU 5
CPTXT EQU 9
CBBIN EQU 7
CPBIN EQU 11

```

```

EOL EQU $9B

```

*

* MACRO FOR CALCULATING THE CHANNEL NUMBER

* (only for internal use, is

* an example of nested

* macro calls)

*

CHANNUM MACRO CHANNEL

```

    LDA #CHANNEL      IOCB-Offset
    ASL                *output channel number
    ASL                *(times 16)
    ASL
    ASL
    TAX                * Result in X-Register
    MEND

```

*

* Name : OPEN

* Goal : Open file

* Call : OPEN <Num>,<Aux1>,<Aux2>,<Filename>

* Example : OPEN 1,4,0,"D:TEST.DBJ"

*

OPEN MACRO CHANNEL,AUX1,AUX2,FILENAME

JMP OP1@

FNAME ASC FILENAME

DFB EOL

OP1@ CHANNUM CHANNEL

LDA #AUX1

STA ICAX1,X

LDA #AUX2

```

        STA ICAX2,X
        LOA #COPEN
        STA ICCOM,X
        LDA #FNAME
        STA ICBAL,X
        LDA #FNAME/256
        STA ICBAH,X
        JSR CIOV
        MEND

*
* Name      : CLOSE
* Goal      : Close file
* Call      : CLOSE <Num>
* Example   : CLOSE 1
*
CLOSE    MACRO CHANNEL
          CHANNUM CHANNEL
          LDA #CCLSE
          STA ICCOM,X
          JSR CIOV
          MEND

*
* Name      : PRINT
* Goal      : Print a string defined with 'ASC',
*            must end with EOL
* Call      : PRINT <Channel>,<Label>
* Example   : PRINT 0,TEXT1
*
PRINT    MACRO CHANNEL,LABEL
          CHANNUM CHANNEL
          LDA #CPTXT
          SIA ICCOM,X
          LDA #LABEL
          STA ICBAL,X
          LDA #LABEL/256
          STA ICBAH,X
          LDA #127      maximum length
          STA ICBLL,X
          LDA #0
          STA ICBLH,X
          JSR CIOV
          MEND

*
* Name      : PRINTS
* Goal      : print string directly
*            to screen
* Call      : PRINT <String>
* Example   : PRINTS "HALLO"

```



```

*
PRINTS  MACRO STRING
        JMP PR2@
PRI1@   ASC STRING
        DFB EOL
PR2@    PRINT 0,PRI1          macro above!
        MEND

*
* Name      : INPUT
* Goal      : Read string
* Call      : INPUT <Channel>,<Label>
* Example   : INPUT 0,TEXT1
*
INPUT    MACRO CHANNEL,LABEL
        CHANNUM CHANNEL
        LDA #CGTXT
        STA ICCOM,X
        LDA #LABEL
        STA ICBAL,X
        LDA #LABEL/256
        STA ICBAN,X
        LDA #127             maximum length
        STA ICBLN,X
        LDA #0
        STA ICBLH,X
        JSR CIOV
        MEND

*
* Name      : BGET
* Goal      : Read data block
*            of length L at address A
* Call      : BGET <Num>,<L>,<A>
* Example   : BGET 1,$B000,$100
*
BGET     MACRO CHANNEL,LENGTH,BUFFER
        CHANNUM CHANNEL
        LDA #CGBIN
        STA ICCOM,X
        LDA #LENGTH
        STA ICBLN,X
        LDA #LENGTH/256
        STA ICBLN,X
        LDA #BUFFER
        STA ICBAL,X
        LDA #BUFFER/256
        STA ICBAN,X
        JSR CIOV
        MEND

* Name      : BPUT

```

```

* Goal      : Save data block
*             of length L at address A
* Call      : BPUT <Num>,<L>,<A>
* Example   : BPUT 1,$B000,$100
*

```

```

BPUT      MACRO CHANNEL,LENGTH,BUFFER
          CHANNUM CHANNEL
          LDA #CPBIN
          STA ICCOM,X
          LDA #LENGTH
          STA ICBLL,X
          LDA #LENGTH/256
          STA ICBLH,X
          LDA #BUFFER
          STA ICBAL,X
          LDA #BUFFER/255
          STA ICBAH,X
          JSR CIOV
          MEND

```

```

*****
* Demo Program I/O-Library
* display directory of drive 1.
*****

```

```

          ORB $A800
          PRINTS "Directory Drive 1:"
          OPEN 1,6,0,"D1:*.*)"

```

```

NEXT      INPUT 1,BUFFER           Read directory line
          BMI END                 End of File?
          PRINT 0,BUFFER          and print
          JMP NEXT                next line

```

```

END        CLOSE 1                Done!
          RTS

```

```

BUFFER     ORG **20                Reserve space

```

```

* START THE DEMO:
* Assemble with <CTRL>-Y,
* activate the monitor with <CTRL>-P
* and start with 'G'oto A800.

```

ATMAS-II Memory Map

0000 - 007F:	Used by Operating System
0080 - 0085:	*** free for personal use ***
0086 - 00DF:	Used by ATMAS-II Editor and Monitor
00E0 - 00FD:	Used by ATMAS-II Assembler, but is free to use after the assembler has finished
00FE - 047F:	Hardware stack, O.S. Vectors, IOCB's etc...
0480 - 05FF:	*** free for personal use ***
0600 - 06FF:	*** free for personal use (Page 6) ***
0700 - <LOMEM>:	DOS, <LOMEM> DOSII:=\$1F00 DOSXL:=\$2700
<L0> - 27FF:	*** free for personal use ***
2800 - 4AFF:	ATMAS-II
4800 - 4BFF:	Line buffer
4C00 - 5FFF:	Symbol table
6000 - 63FF:	Copy-register
6400 - A7FF:	Text buffer (source code)
A800 - <MEMTOP>:	*** free for personal use ***
<ME> - BFFF:	Display list and screen memory

There are several ranges of free memory you can use for your object code:

- A) \$600-\$6FF: the infamous Page 6, especially used for writing small routines to be called by `USR()` from BASIC.
- B) \$1F00-\$27FF: this range starts at the end of DOS, which can be determined by reading \$2E7-\$2E8, up to the start address of ATMAS-II itself. The end of DOS can vary for different DOS versions, so you have to be careful here. For standard DOS-II, this range starts at \$1F00.
- C) \$A800-\$BC3F: The standard range for object code, reaches up to the display list and screen memory.

The memory used by the symbol table (\$4C00-\$5FFF) can be used for variables, font definitions or Player-Missile Graphics, as the symbol table will not

be used after the assembler has run and will be rebuilt the next time the assembler is run.

Error codes

B.1 Assembler errors

SYNTAX ERROR	e.g. a line-number is specified or a line consists only of whitespace (tabs and spaces)
NAME UNKNOWN	Invalid opcode, e.g. SRA instead of STA
UNDEFINED EXPRESSION	Invalid operand, e.g. invalid expression or undefined label.
ADDRESSING ERROR	Addressing mode mismatch, i.e. STA #\$6FF
IMPOSSIBLE BRANCH	Branches (BNE , BCS ...) can only reach addressess between *-128 and *+127 . Use JMP instead.
DIVISION BY ZERO	e.g. LDA #100/0
NUMBER-ERROR	Invalid number, e.g. LDA #%30 or LDA #A0
WRONG DELIMITER	Invalid delimiter used for ASC command.
NO ASCII	Invalid ASCII character constant, e.g. LDA #' RETURN
LINE TO LONG	Maximum line length is 127 characters
MACRO ERROR	Invalid macro definition or macro call (invalid parameters)
ORG ERROR	Invalid ORG -directive
TOO MANY LABELS	Symbol table is full

OPCODE DIFFERENT	Pass 3 does not match pass 2. This can happen if pass 2 overwrites the text buffer because of an invalid ORG .
------------------	---

B.2 Editor errors

RW	Error during disk I/O
CO	Command line too long
E?	Error, invalid command line
H?	Invalid hexadecimal value
I?	Textbuffer is full (T:00000)
L?	Device error during output of listing
S?	String not found (search-function)
T?	Invalid tab-width (must be 1-9)
C?	Copy-register is full (C:0000)
#?	Invalid repeat-factor (must be 2-255),
OK	Situation Normal, All... is well. Copy-register is closed
CR	Copy-register is open

B.3 Monitor error codes

ADR ERROR	Wrong address during loading or saving of a program. If this occurs during a LOAD -command, it's probably not a binary file, but, for example, a text file.
ERRORCODE	80-FF: Operating System Error, see the DOS-Manual. Error codes are hexadecimal.

APPENDIX

C



ATMAS-II REFERENCE CARD

<div> <div>CTRL-E</div> <div>CTRL-D</div> <div>CTRL-R</div> <div>CTRL-F</div> <div>CTRL-J</div> <div>CTRL-K</div> <div>CTRL-V</div> <div>CTRL-T</div> <div>CTRL-G</div> <div>CTRL-Y</div> <div>CTRL-P</div> </div>	Editor commands in textmode		<div> <div>ORG</div> <div>EQU</div> <div>EPZ</div> <div>DFB</div> <div>DFW</div> <div>ASC</div> <div>OUT</div> <div>MACRO</div> <div>MEND</div> <div>M</div> <div>D</div> <div>C</div> <div>F</div> <div>B</div> <div>G</div> <div>S</div> <div>L</div> <div>E</div> <div>I</div> </div>	Assembler directives	
	Cursor to start of text			Specify start address	
	Cursor to end of text			Define constant	
	Open copy-register			Define zero-page constant (not obligatory)	
	Close copy-register			Insert byte into object code	
	Insert copy-register			Insert word into object code	
	Clear copy-register			Insert ASCII or screencodes into object code	
	Switch between 1- and 2-line mode			Control listing format	
	Toggle control-character display			Start macro-definition	
	Repeat command line			End macro-definition	
	Call ATMAS-II macroassembler			Monitor Commands	
	Call Monitor			Memory dump	
Editor commands at the command line				Disassemble memory range	
B	Cursor one position backwards			Change memory range	
F	Cursor one position forwards			Fill memory range with constant value	
D	Delete character left from cursor			Block transfer (copy)	
T	Delete untill end-of-line			Goto, i.e. jump to specified address	
H<Hexbyte>	Insert specified ASCII-Code			Binary Save	
S<String>	Search for <String>			Binary Load	
I<String>	Insert <String>			Return to editor	
J	Repeat command			Inventory, list directory contents of D1:	
K	Clear textbuffer				
U	User command, start code at \$A800				
@<n>	Set tabwidth to <n> characters				
M	Return to DOS				
E	Clear copy-register				
G	Insert copy-register				
W<D:FN>	Save text as <D:FN>				
R<D:FN>	Read text from <D:FN> (insert at cursor position)				
L<0 1 2>	List text to device (1:=Drucker)				
			ATMAS-II Reference Card		