



**Skyles
Electric
Works**

MACROTEA

	0330	;
	0340	; STACK ADJUSTMENTS
	0350	;
0354-A900	0360SADJ	LDA #0
0356-48	0370	PHA
0357-48	0380	PHA
0358-48	0390	PHA
0359-48	0400	PHA
035A-4C85E6	0410	JMP \$E685
	0420	;
	0430	; LOOK AT KEYBOARD IMAGES AND SEE
	0440	; IF SHIFT AND PI ARE BOTH DOWN
	0450	;
035D-AD0402	0460CSTART	LDA \$0204 ; CHECK SHIFT
0360-C901	0470	CMP #01
0362-D00E	0480	BNE FINISH
0364-AD0302	0490	LDA \$0203 ; CHECK FOR PI
0367-C93B	0500	CMP #59
0369-D007	0510	BNE FINISH
036B-205403	0520	JSR SADJ
036E-EA	0530	NOP
036F-4C9040	0540	JMP RESTART ; GO DO IT!!
	0550	;
0372-205403	0560FINISH	JSR SADJ ; REQUIRED MUMBO
0375-EA	0570	NOP ; JUMBO TO GET BACK TO NORMAL
0376-4C7EE6	0580	JMP \$E67E ; OPERATION
0379-EA	0590	NOP
037A-EA	0600	NOP
	0610	.EN

LABEL FILE: [/ = EXTERNAL]

RESTART=4090
SADJ=0354

TURNON=033A
CSTART=035D

TURNOFF=0347
FINISH=0372

USER'S MANUAL

Copyright © 1980,1981,1982 by Skyles Electric Works
231-E So. Whisman Rd.
Mountain View, CA 94041
(415) 965 1735

This manual was originally prepared by Gregory Yob.
This manual was updated by Jim Berkey.

SUMMARIES

To Begin With

We are pleased that you have acquired the Skyles MACROTEA and are sure you will find the MACROTEA to be a handy and useful tool for your PET. Please spend some time looking at this manual to get a "feel" for the MACROTEA before commencing your first project.

ORGANIZATION OF THIS MANUAL

Most manuals are arranged in a literary order - starting with simple examples, and finishing with several appendices which contain the gory details. Though this is fun to read, it isn't very convenient to flip through a lot of pages to understand what !09 AT 257 means.

With this in mind, the most accessed parts are put first and the details are put second. When you use this manual, the summaries will usually be sufficient for your needs, and only now and then will you need to read all of the details.

Here is the rough outline:

SUMMARIES

- Error Numbers
- & So Forth (Defaults, etc.)
- Alphabetical commands lists for:
 - Editor
 - Assembler
 - Monitor
- 6502 Instructions Charts

DETAILS

- Editor
- Assembler
- Monitor
- Memory Usage

You must be warned that the MACROTEA is for experienced users of the 6502 at the machine language level!!! If you are a beginner with machine language, please get some of the books on the 6502 (such as those by Sybex) and know that learning MACROTEA will take some time and effort.

For a first experience with MACROTEA, glance through the summaries and then try the examples given with each feature in the details part. A few hours spent doing this will save many hours later.

If you are wondering where the installation instructions are, they are in an Appendix at the very end of this Manual. After all, you are going to use these only once.

NOTE: Blank pages such as this one will appear here and there in this Manual. Since this is your Manual, please feel free to use these pages for your notes.

Errors

ERROR MESSAGES

Errors are reported in the form: !EE AT LINE XXXX/YY with EE as the error number, XXXX the line number, and if a file is being used, /YY indicates the file number (PET physical device number, which is usually 01 or 02). In some cases, for example, an illegal command, the line number will have no meaning.

Example: !07 AT LINE 575

LIST OF ERROR MESSAGES

- ED Can't recognize or complete the command.
- 00 The address or label isn't in the zero page.
- 01 The branch is out of range (-126 to +129 from P.C.)
- 02 Illegal mnemonic for op code.
- 03 Unrecognized pseudo op.
- 04 .BA or .MC Operand isn't defined.
- 05 .DI isn't labeled.
- 06 Label is already defined (duplicated).
- 07 .EN missing at end.
- 08 Undefined or Illegal label.
- 09 Bad character in a decimal string. (Expected 0-9)
- 0A Missing or error in the Operand.
- 0B Addressing mode not available for this op code.
- 0C Bad character in label.
- 0D Bad character in hex string. (Expected 0-9 or A-F)
- 0E Symbol Table full. Label not inserted.
- 0F Workspace full. Line not entered.
- 10 Line # too large in execution of NUMBER.
- 11 A parameter is expected here.
- 12 COPY or MOVE destination within Start to End Line #s
- 13 (Not Used)
- 14 Cannot make a relocatable object tape. (See Note below.)
- 15 EDIT command has a syntax error.
- 16 Bad Tape Unit number. (Should be 01 or 02.)
- 17 Bad Tape Load.(Checksum Error)

----- Macro Related Error Messages are on the next page. -----

NOTE (14) A relocatable object tape requires that the previous assembly had no errors. An assembly is required to build the object tape as the label file must be present & correct.

An error will leave you in the Command Mode of the Editor. The .CE pseudo op will continue assembly except for errors !04, !07 and !17.

LIST OF MACRO RELATED ERROR MESSAGES

- 20 Macro not defined yet.
- 21 .EN found while a macro is being expanded.
- 22 .EN found in a conditional suppress.
- 23 .EN found in a macro definition.
- 24 Macros nested too deeply (32 maximum).
- 25 Bad characters or # of parameters don't match.
- 26 Macro already defined.
- 27 Macro definition crosses a file boundary.
- 28 (Not Used)
- 29 Macro definition found within another macro definition.
- 2A (label) in SET must be symbolic.
- 2B .ME found without prior .MD.
- 2C (Not Used)
- 2D (Not Used)
- 2E Too many macros (65536 maximum).
- 2F Too many files (65536 maximum).

TO START MACROTEA:

MACROTEA is a machine language program resident in the ROM on the MACROTEA board. Once you have installed MACROTEA, (the instructions to install are in an appendix in the back of this manual) use the BASIC POKE and SYS to the MACROTEA "cold start" address. If you leave MACROTEA for some reason, the "warm start" address will preserve the state of MACROTEA. In a "warm start" the text in the workspace and the symbol table will be preserved.

"cold start"	"warm start"	
POKE45056,199:SYS36864	SYS36998	from BASIC
G 9000	G 9086	from the Monitor
A	Z	from the Monitor

Here is a starting example:
POKE45056,199:SYS36864

```
0800-57FC 5800-67FC 0000
0800 5800
|x|
```

The cursor is the "~~x~~" symbol. See the SET command at page 76 for an explanation of the numbers that appear.

In some cases, mostly when running code you have assembled, the PET might "run away" - that is, the PET is caught in a machine language loop or the 6502 has choked on an illegal instruction code. Press the "reset" button on the MACROTEA board.

After pushing "reset" button SYS36864, then
"BR" back to the monitor, then "LOAD", the saved object code.

CONTROL OF OUTPUT

STOP key will abort any current operation. You will be in the MACROTEA Editor. (Exception - during tape or disk I/O, pressing STOP will leave you in BASIC. SYS 37001 to warm start MACROTEA.)

SPACE key will "freeze" any MACROTEA output or operations. Press any other key to resume operations.

DEFAULT SETTINGS

Many aspects of MACROTEA are controlled by flag values - which must start at one setting or another. If you do not explicitly tell MACROTEA otherwise, the settings shown below will be in effect.

EDITOR DEFAULT SETTINGS

The memory settings are defaulted to:

0800 - 5800 Text Workspace Area

5800 - 6800 Symbol Table

Note that the SET command will report these with a 3 byte offset at the upper end, ie, the Symbol Table is reported as 5800-67FC, and when the SET command is used to change the map, subtract 3 from the intended upper bound.

Due to the various combinations of character ROMs and PET models, MACROTEA POKes the PET to the Upper Case/ Graphics mode to provide uniformity. In most applications, there is no need to enter shifted characters. (In some cases, MACROTEA will not correctly understand them anyways.)

GET, PUT and OUTPUT default to Tape Unit #1.
GET without a "filename" will accept any file.
PUT without a "filename" creates a file with a null name.

Disc versions of MACROTEA will default GET and PUT to the disc.
Non-disc versions of MACROTEA default to the tape cassettes for GET and PUT.

ASSEMBLE defaults to the LIST option, starting at line 0.
An assembly will also make FORMAT SET and leave it that way.

AUTO mode is not in effect.

FORMAT is CLEAR (until an assembly)

MANUSCRIPT is CLEAR.

PRINT will print the entire workspace if not given a line number(s).

ASSEMBLER DEFAULT SETTINGS

The assembly will be from the source text in the workspace.

Object code will not be placed in memory.

An object listing will be generated.

The starting address is \$7800

If the .OS pseudo op is present, code will be stored starting
at \$7800

Assembly will stop if an error is detected.

Source on tape will be read from Tape Unit # 1.

Macros are not expanded during an assembly.

DISK VERSION DEFAULT SETTINGS

Those of you with the Commodore Disk version of MACROTEA should note that some of the MACROTEA defaults are different for your version:

GET, PUT commands will select the disk as the I/O device for file transfer & placement of relocatable code. You must provide the device & filename in the usual "drive:filename" form.

The command DISK is present and active. It performs the function of the "Wedge" (called DOS SUPPORT by Commodore). Disk commands in quotes after DI will be executed in the same way as "Wedge" does. DI by itself looks at and reports the error channel.

The .CT and .EN pseudo-ops will use the disk as the file I/O device. .CT must have the "drive:filename" of the next file to be assembled. .EN must have the "drive:filename" of the first file in the assembly chain.

MACROTEA GLOSSARY OF TERMS

As we all know, cpu's aren't standard, languages aren't standard, and (clearly!) neither are assemblers. The terms used in MACROTEA are shown below along with their meaning in this manual.

Argument	MACROTEA's macros can accept values which differ each time the macro is called. These arguments are enclosed in parenthesis and separated by spaces.
Command	This is always an instruction for the Editor part of MACROTEA, or for the Monitor. No line number is used with a command.
Directive	A command to the assembler, placed in the second field of the source text. (Directives don't have the period used in Pseudo-ops.)
Error	MACROTEA cannot understand all combinations of letters & symbols, or for some reason cannot do as you instruct. This is usually an error. Not all errors will get caught, however.
Expression	The operand in many cases is arrived at by computation from other labels and numbers. The combination of labels, numbers, and arithmetic operators is an expression.
Label	This is usually an address value, and sometimes just a handy number. Labels are defined in the first field, and referenced in the third field of the source text.
Macro	A macro defines a section of code under a name which is reassembled each time the macro is called. Placing the macro's name in the Op-Code field calls the macro.
Op-Code	This is always a 6502 mnemonic.
Operand	The operand will always result in a numerical value, and is the third field in an assembly source text.
Parameter	Many commands need some values or tags for their use. These are called parameters and follow the command. Parameters are separated by spaces.
Pseudo-op	This is a fake "Op-code" which instructs the assembler to perform actions related to assembly.
Symbol Table	As the assembler scans the source code, the labels are stored in this area of memory. When finished, the Symbol Table holds the labels and their values.
Workspace	This is the area in memory used by the Editor to store the text. For assembly, the source code is taken from the workspace or from tape.

Editor

EDITOR COMMANDS

The MACROTEA Editor has commands for both text editing and for the initiation and control of the assembly process. All Editor commands and parameters can be abbreviated to their first two characters. For example, NU is the same as NUMBER.

Text is entered into the workspace by numbering each line. A line that does not begin with a number is seen as a command. Individual lines may be deleted by entering their number. The PET Screen Editor is used to modify individual lines, and RETURN passes the modified line to the MACROTEA Editor.

A command and its parameters must be separated by spaces. Do not use commas, as the MACROTEA will see this as an error.

LIST OF EDITOR COMMANDS

Each command and its variations is shown below. The page number indicates where to find more detailed information.

(Line Number)	(Characters)	Enter into workspace.	
	nil	Delete line.	Page 41

AUTO (increment)		Automatic line numbering for text entry.	
0 or nil		Exit AUTO mode.	Page 45

NUMBER (Start Line#)(Increment)		Renumbers text lines in workspace.	
			Page 47

FORMAT SET		PRINT will now display workspace formatted in tabulated columns.	
CLEAR		Disable FORMAT mode.	Page 73

COPY (Destination#)(Start#)(End#)		Copys text lines in Start# to End# to just following Destination#.	
			Page 50

MOVE(Destination#)(Start#)(End#)		Moves text lines in Start# to End# to just following Destination#.	
			Page 53

DELETE (Start Line#)(End Line#) (Line#)	Deletes all lines included in range. Deletes one line.	Page 55

CLEAR	Delete <u>entire</u> workspace text.	Page 58

PRINT (Start Line#)(End Line#) (Line #) nil	Display indicated text lines on screen. Display one line. Print entire workspace.	Page 70

PUT (D1 or D2)("Filename") nil "drive:filename"	Write text file from workspace to tape. Tape #1, Null filename. Write from workspace to disk file.	Page 82

GET (D1 or D2)("Filename") nil "drive:filename"	Read text file from tape into workspace. Tape #1, Any filename. Read text file from disk into workspace.	Page 85

DUPLICATE	Copy tape from Drive 2 to Drive 1.	Page 88

HARD SET (First Page#)	Print on printer all MACROTEA output. If page # is indicated, start with given page number.	Page 72
CLEAR	Disable HARD mode.	

ASSEMBLE LIST (Start Line#) NOLIST (St Lin#) nil	Assemble starting at Start Line# in workspace. Also display listing on screen. Don't display listing on screen. Start at Line 0. LIST default.	Page 93

PASS	Perform second pass of assembly. (For assemblys using tape files).	Page 99

RUN (label)	Execute assembled program starting at label. (Return via RTS.)	Page 96

SYMBOLS	Display Symbol Table.	Page 101

SET (Text Start) (Text End) (Symbol Start) (Symbol End) nil	Change memory boundaries for workspace, symbol table. Report current values of parameters.	Page 76

DISK (any disk command) nil	Execute Commodore Disk Command. Read & report Disk Error Channel. Page 90
EDIT (argument)	----- Edit/change lines of text in work- space. Page 63
FIND (argument)	----- Find text string in workspace. Page 59
MANUSCRIPT SET CLEAR	----- PRINT will not display line numbers. PRINT will display line numbers. Page 74
BREAK	----- Jump to Monitor. (6502 BRK used.) Page 98
APPEND ("File Name")	----- Appends source code to the end of existing source code

Assembler

ASSEMBLER COMMANDS

The assembler reads the text in the workspace or from tape cassette files and produces 6502 machine language code.

Text that is intended for assembly should have this format:

(Line Number)(Label) (Op Code) (Operand) ; (Comment)

Labels must immediately follow the Line Number without any spaces. Op Codes must either be 6502 Mnemonics or Assembler Pseudo-ops. All Comments must be preceded by the semi-colon. Spaces are required between the Label and Op Code, the Op Code and any Operand, and the Operand and the semi-colon before any Comments.

LIST OF ASSEMBLER PSEUDO-OPS

The pseudo-ops accepted by the assembler are listed below. See the indicated pages for further information. All pseudo-ops have the period as their first character.

.BA (expression)	Begin assembly at (expression). Similar to ORG in other assemblers.	Page 120
.CE	Continue assembly if errors are found.	Page 112
.LS	Print source listing on Pass 2.	Page 110
.LC	Inhibit source listing on Pass 2.	Page 110
.CT "filename" "drive:filename"	Continue source from a tape file. If you have a disk, the "drive:filename" for the next file is required.	Page 112
.OS	Load object code into memory on Pass 2.	Page 120
.OC	Don't load object code into memory on Pass 2. (Default setting)	Page 120
.MC (expression)	Store object code starting at (expression). Code is assembled as specified by .BA.	Page 120
.DS (expression)	Skip (expression) bytes to define storage.	Page 125
.BY (values)	Store data per (values).	Page 125

.SI (expression)	Store (expression) in next two memory bytes as an address (internal address). Page 125
.DI (expression)	Define value of label to be that of (expression). .DI <u>must</u> be labeled (internal address). Page 125
.EN "filename" "drive:filename"	End of source text. .EN is <u>required</u> . Disk users must state "drive:filename" of <u>first</u> file if .CT is used. Page 112
.EJ	Send form-feed to printer if HARD SET. Page 110
.MD (arguments list)	Define a macro. Page 136
.ME	End of macro definition. Page 136
.EC	Don't expand macro on source listing. Page 136
.ES	Expand macros on source listing. Page 136

LIST OF ASSEMBLER CONDITIONAL DIRECTIVES

The type of IFx directive and the value of the (expression) determines if the following code is to be assembled.

***	End of IFx block.	Page 132
IEQ (expression)	Assemble if (expression) = 0	Page 132
IMI (expression)	Assemble if (expression) < 0	Page 132
INE (expression)	Assemble if (expression) <> 0	Page 132
IPL (expression)	Assemble if (expression) >= 0	Page 132
SET (label) = (expression)	Redefine the value of a label.	Page 125

Note that the If directives follow the 6502 Branch Instruction Conditions - for example, BNE is Branch If Not Zero, and INE is for IF Not Zero.

ON ADDRESSING MODES

MACROTEA will not assemble operands less than \$FF into the 6502 base page instructions unless explicitly told to do so. The addressing modes are:

#	-	Immediate	LDA #45+FOO
nil	-	Absolute	LDA 45+FOO
*	-	Base Page	LDA *45
,X	-	Absolute Indexed	LDA F00,X
,Y	-	" "	LDA F00,Y
*,X	-	Base Page Indexed	LDA *45,X
*,Y	-	" "	LDA *45,Y
(,X)-	-	Indexed Indirect	LDA (F00,X)
(),Y-	-	Indirect Indexed	LDA (F00),Y
()	-	Indirect	JMP (F00)
A	-	Accumulator	ROR A

To help with address manipulation, two extensions to immediate addressing are provided:

#H,	-	Hi Byte of Value	LDA #H,F00
#L,	-	Lo Byte of Value	LDA #L,F00

Also, ASCII values are available:

#'	-	ASCII Value	LDA #'Q
----	---	-------------	---------

ON LABELS AND EXPRESSIONS

In most cases a label is somewhat like a variable in BASIC. It is "assigned" (defined) by placing the label's name immediately after a line number. Labels may be defined only once, unless the SET directive is used.

Label names are from 1 to 10 characters. For readability, select the first character from A-Z, and further characters from A-Z or 0-9.

The operand field will accept numerical values, labels, or simple arithmetic expressions made from numbers & labels. The final value of the operand must be a positive number in the range \$0000 to \$FFFF.

Valid forms for numbers are shown below. If the digits string is too long, the rightmost digits are used to form a number in the legal range.

nnnnn	-	Decimal number.	nnnn taken from (0123456789)
\$nnnn	-	Hexadecimal number.	from (0123456789ABCDEF)
%nnnnn	-	Binary number.	from (01)

The program counter is indicated by =. Its value is taken from the counter's value when the current line was first encountered.

Expressions are numbers, labels, or = separated by the operators + or -. Unary minus is not permitted, and the expression is evaluated from left to right (like a simple calculator).

Please note that the arithmetic operations of multiply (*) and divide (/) are not available in MACROTEA.

Note: A label 'A' by itself in an operand field will indicate the accumulator addressing mode. Use A+0 to force evaluation of the label 'A'.

Expressions may not have embedded blanks. A blank will force the assembler to assume the rest of the line is a comment.

IN THE COURT OF THE DISTRICT OF COLUMBIA

In the case of the People of the District of Columbia, Plaintiff, vs. [Name], Defendant.

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

The Defendant, [Name], is charged with the crime of [Crime].

Monitor

MONITOR COMMANDS

The monitor expects a single letter command followed by any parameters separated by spaces or commas. If the monitor cannot decipher a command, it will print a question mark in the command's line or ignore the command. A period is printed as the monitor's prompt.

The PET's screen editor may be used to modify memory or registers.

MACROTEA MONITOR REFERENCE CARD

NO.	COMMAND-SYNTAX	MNEMONIC	ACTION	PAGE
1	A	allclear	Exit to MACROTEA coldstart(erases everything)	146
2	BREAK or BR	break	Initialize and/or enter Monitor from MACROTEA	146
3	B (addr) (count)	breakpoint	Set soft breakpoint for Q command	159
4	C	close	Send Monitor output to screen	166
5	D (addr) D (addr1) (addr2)23 lines....scroll range....	disassemble	Display memory in hex and 6502 source text	152
6	F (addr1) (addr2) (byte)	fill	Write memory range with specified byte	155
7	G (addr)	goto	Execute 6502 code	158
8	H (addr1) (addr2) (byte seq) H (addr1) (addr2) ('ASCII seq)	hunt	Display start addresses of specified sequence	153
9	I (addr) I (addr1) (addr2)23 lines....scroll range....	interrogate	Display and/or write memory in hex and ASCII	150
10	K	kleanup	Reset Monitor IRQ and PET I/O Status Byte	165
11	L "P:FILENAME", (device#) L "FILENAME", (optional device#)	load	Load disk or tape file into memory (P: or L: for disk drive 0 or 1)	164
12	M (addr1) (addr2)	memory	Display and/or write memory in hex	149
13	O ...the letter oh gives device# 4... O (device#) ...or specify any device#...	open	Send Monitor output to printer	166
14	Q (addr)	quicktrace	Execute 6502 code with soft breakpoint	159
15	R	registers	Display Monitor's copies of 6502 registers	148
16	S "P:FILENAME", (device#), (start addr), (end addr+1) S "FILENAME", (device#), (start addr), (end addr+1)	save	Save memory into disk or tape file (P: or L: for disk drive 0 or 1)	164
17	SYS4	sysenter	Re-enter Monitor from BASIC	146
18	T (addr1) (addr2) (addr3)	transfer	Copy from addr1-addr2 to addr3	154
19	W (addr)	walk	Execute 6502 code with singlestep disassembly	161
20	X	exit	Exit to BASIC	146
21	Z	zipout	Exit to MACROTEA warmstart(keep environment)	146

TO START THE BASIC PROGRAMMERS TOOLKIT OR COMMAND-O

X (exit) to BASIC then type

```
POKE46106,251:SYS 45056 (Toolkit)
POKE46106,251:SYS 36864 (Command-O)
```

You should see on the screen
(c)1979 PAICS or (c) 1981 Robin Chang

You are now in BASIC with the Toolkit or Command-O energized. Enjoy.

TO RESTART MACROTEA:

It is permissible to switch from the Toolkit to MacroTeA with the PET power on. You should be in the READY mode, with a blinking cursor, before switching. Many ROM's such as the BASIC Programmers Toolkit change some page zero RAM memory locations. Before you switch out of the Toolkit ROM you must return these locations to their previous state. You may do this with the "reset" button or if you wish to save the contents of memory below the screen, the following machine language instructions should be executed before leaving the Toolkit.

<u>for "old PET ROMs"</u>		
<u>Hexidecimal Code</u>	<u>Nemonic</u>	<u>Decimal "Poke"</u>
A2 1C	LDX #\$1c	162 28
BD B4 E0 LOOP	LDA \$E0B4,X	189 180 224
95 C1	STA \$C1,X	149 193
CA	DEX	202
D0 F8	BNE LOOP	208 248
60	RTS	96

<u>for "new PET ROMs"</u>		
<u>Hexidecimal Code</u>	<u>Nemonic</u>	<u>Decimal "Poke"</u>
A2 1C	LDX #\$1C	162 28
BD FB E0 LOOP	LDA \$E0F8,X	189 248 224
95 6F	STA \$6F,X	149 111
CA	DEX	202
D0 F8	BNE LOOP	208 248
60	RTS	96

Of course with Command-O, a simple KILL command solves the above problems.

Then type POKE45056,199:SYS36998 (Warm Start)

MACROTEA MONITOR REFERENCE CARD

NO.	COMMAND-SYNTAX	M. JNIC	ACTION	PAGE
1	A	allclear	Exit to MACROTEA coldstart(erases everything)	146
2	BREAK or BR	break	Initialize and/or enter Monitor from MACROTEA	146
3	B (addr) (count)	breakpoint	Set soft breakpoint for Q command	159
4	C	close	Send Monitor output to screen	166
5	D (addr) D (addr1) (addr2) ...23 lines... ...scroll range...	disassemble	Display memory in hex and 6502 source text	152
6	F (addr1) (addr2) (byte)	fill	Write memory range with specified byte	155
7	G (addr)	goto	Execute 6502 code	158
8	H (addr1) (addr2) (byte seq) H (addr1) (addr2) ('ASCII seq) ...hunt hex... ...hunt ASCII...	hunt	Display start addresses of specified sequence	153
9	I (addr) I (addr1) (addr2) ...23 lines... ...scroll range...	interrogate	Display and/or write memory in hex and ASCII	150
10	K	kleanup	Reset Monitor IRQ and PET I/O Status Byte	165
11	L "Ø:FILENAME", (device#) L "FILENAME", (optional device#) ...disk... ...tape...	load	Load disk or tape file into memory (Ø: or L: for disk drive Ø or L)	164
12	M (addr1) (addr2)	memory	Display and/or write memory in hex	149
13	O ...the letter oh gives device# 4... O (device#) ...or specify any device#...	open	Send Monitor output to printer	166
14	Q (addr)	quicktrace	Execute 6502 code with soft breakpoint	159
15	R	registers	Display Monitor's copies of 6502 registers	148
16	S "Ø:FILENAME", (device#), (start addr), (end addr+1) S "FILENAME", (device#), (start addr), (end addr+1)	save	Save memory into disk or tape file (Ø: or L: for disk drive Ø or L)	164
17	SYS4	sysenter	Re-enter Monitor from BASIC	146
18	T (addr1) (addr2) (addr3)	transfer	Move code addr1-addr2 to addr3	154
19	W (addr)	walk	Execute 6502 code with singlestep disassembly	161
20	X	exit	Exit to BASIC	146
21	Z	zipout	Exit to MACROTEA warmstart(keep environment)	146

6502 Stuff

6502 INSTRUCTION SET TABLES

The tables provided here are intended for convenient reference use only. For a detailed description of the 6502, see the MOS Technology 6502 Programmer's Manual or one of the several books about the 6502.

The alphabetical list and the instruction diagrams are from the MOS manual. The table of addressing modes & op codes is from a paper by T.G. Windeknecht in the 1979 NCC Personal Computing Proceedings.

6502 INSTRUCTION SET - ALPHABETIC LIST

ADC	Add Memory to Accumulator with Carry	JSR	Jump to New Location Saving Return Address
AND	"AND" Memory with Accumulator	LDA	Load Accumulator with Memory
ASL	Shift Left One Bit (Memory or Accumulator)	LDX	Load Index X with Memory
BCC	Branch on Carry Clear	LDY	Load Index Y with Memory
BCS	Branch on Carry Set	LSR	Shift Right One Bit (Memory or Accumulator)
BEQ	Branch on Result Zero	NOP	No Operation
BIT	Test Bits in Memory with Accumulator	ORA	"OR" Memory with Accumulator
BMI	Branch on Result Minus	PHA	Push Accumulator on Stack
BNE	Branch on Result not Zero	PHP	Push Processor Status on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
BRK	Force Break	PLP	Pull Processor Status from Stack
BVC	Branch on Overflow Clear	ROL	Rotate One Bit Left (Memory or Accumulator)
BVS	Branch on Overflow Set	ROR	Rotate One Bit Right (Memory or Accumulator)
CLC	Clear Carry Flag	RTI	Return from Interrupt
CLD	Clear Decimal Mode	RTS	Return from Subroutine
CLI	Clear Interrupt Disable Bit	SBC	Subtract Memory from Accumulator with Borrow
CLV	Clear Overflow Flag	SEC	Set Carry Flag
CMP	Compare Memory and Accumulator	SED	Set Decimal Mode
CPX	Compare Memory and Index X	SEI	Set Interrupt Disable Status
CPY	Compare Memory and Index Y	STA	Store Accumulator in Memory
DEC	Decrement Memory by One	STX	Store Index X in Memory
DEX	Decrement Index X by One	STY	Store Index Y in Memory
DEY	Decrement Index Y by One	TAX	Transfer Accumulator to Index X
EOR	"Exclusive-Or" Memory with Accumulator	TAY	Transfer Accumulator to Index Y
INC	Increment Memory by One	TSX	Transfer Stack Pointer to Index X
INX	Increment Index X by One	TXA	Transfer Index X to Accumulator
INY	Increment Index Y by One	TXS	Transfer Index X to Stack Pointer
JMP	Jump to New Location	TYA	Transfer Index Y to Accumulator

6502 INSTRUCTION SET - OP CODES & ADDRESSING MODES TABLE

The abbreviations at the top of the table are:

INSTR	Instruction	ZPY	Zero Page, Y
IMP	Implied	ABS	Absolute
IMM	Immediate	ABX	Absolute, X
ACC	Accumulator	ABY	Absolute, Y
REL	Relative	IND	Indirect
ZPG	Zero (Base) Page	INX	(Indirect, X)
ZPX	Zero Page, X	INY	(Indirect),Y

INSTR	IMP	IMM	ACC	REL	ZPG	ZPX	ZPY	ABS	ABX	ABY	IND	INX	INY
ADC	--	69	--	--	65	75	--	6D	7D	79	--	61	71
AND	--	29	--	--	25	35	--	2D	3D	39	--	21	31
ASL	--	--	0A	--	06	16	--	0E	1E	--	--	--	--
BCC	--	--	--	90	--	--	--	--	--	--	--	--	--
BCS	--	--	--	B0	--	--	--	--	--	--	--	--	--
BEQ	--	--	--	F0	--	--	--	--	--	--	--	--	--
BIT	--	--	--	--	24	--	--	2C	--	--	--	--	--
BMI	--	--	--	30	--	--	--	--	--	--	--	--	--
BNE	--	--	--	D0	--	--	--	--	--	--	--	--	--
BPL	--	--	--	10	--	--	--	--	--	--	--	--	--
BRK	00	--	--	--	--	--	--	--	--	--	--	--	--
BVC	--	--	--	50	--	--	--	--	--	--	--	--	--
BVS	--	--	--	70	--	--	--	--	--	--	--	--	--
CLC	18	--	--	--	--	--	--	--	--	--	--	--	--
CLD	D8	--	--	--	--	--	--	--	--	--	--	--	--
CLI	58	--	--	--	--	--	--	--	--	--	--	--	--
CLV	B8	--	--	--	--	--	--	--	--	--	--	--	--
CMP	--	C9	--	--	C5	D5	--	CD	DD	D9	--	C1	D1
CPX	--	E0	--	--	E4	--	--	EC	--	--	--	--	--
CPY	--	C0	--	--	C4	--	--	CC	--	--	--	--	--
DEC	--	--	--	--	C6	D6	--	CE	DE	--	--	--	--
DEX	CA	--	--	--	--	--	--	--	--	--	--	--	--
DEY	88	--	--	--	--	--	--	--	--	--	--	--	--
EOR	--	49	--	--	45	55	--	4D	5D	59	--	41	51
INC	--	--	--	--	E6	F6	--	EE	FE	--	--	--	--
INX	E8	--	--	--	--	--	--	--	--	--	--	--	--
INY	C8	--	--	--	--	--	--	--	--	--	--	--	--
JMP	--	--	--	--	--	--	--	4C	--	--	6C	--	--
JSR	--	--	--	--	--	--	--	20	--	--	--	--	--
LDA	--	A9	--	--	A5	B5	--	AD	BD	B9	--	A1	B1
LDX	--	A2	--	--	A6	--	B6	AE	--	BE	--	--	--
LDY	--	A0	--	--	A4	B4	--	AC	BC	--	--	--	--
LSR	--	--	4A	--	46	56	--	4E	5E	--	--	--	--
NOP	EA	--	--	--	--	--	--	--	--	--	--	--	--
ORA	--	09	--	--	05	15	--	0D	1D	19	--	01	11

INSTR	IMP	IMM	ACC	REL	ZPG	ZPX	ZPY	ABS	ABX	ABY	IND	INX	INY
PHA	48	--	--	--	--	--	--	--	--	--	--	--	--
PHP	08	--	--	--	--	--	--	--	--	--	--	--	--
PLA	68	--	--	--	--	--	--	--	--	--	--	--	--
PLP	28	--	--	--	--	--	--	--	--	--	--	--	--
ROL	--	--	2A	--	26	36	--	2E	3E	--	--	--	--
ROR	--	--	6A	--	66	76	--	6E	7E	--	--	--	--
RTI	40	--	--	--	--	--	--	--	--	--	--	--	--
RTS	60	--	--	--	--	--	--	--	--	--	--	--	--
SBC	--	E9	--	--	E5	F5	--	ED	FD	F9	--	E1	F1
SEC	38	--	--	--	--	--	--	--	--	--	--	--	--
SED	F8	--	--	--	--	--	--	--	--	--	--	--	--
SEI	78	--	--	--	--	--	--	--	--	--	--	--	--
STA	--	--	--	--	85	95	--	8D	9D	99	--	81	91
STX	--	--	--	--	86	--	96	8E	--	--	--	--	--
STY	--	--	--	--	84	94	--	8C	--	--	--	--	--
TAX	AA	--	--	--	--	--	--	--	--	--	--	--	--
TAY	A8	--	--	--	--	--	--	--	--	--	--	--	--
TSX	BA	--	--	--	--	--	--	--	--	--	--	--	--
TXA	8A	--	--	--	--	--	--	--	--	--	--	--	--
TXS	9A	--	--	--	--	--	--	--	--	--	--	--	--
TYA	98	--	--	--	--	--	--	--	--	--	--	--	--

6502 INSTRUCTION SET - INSTRUCTION DIAGRAMS & GORY DETAILS

ADC*Add memory to accumulator with carry***ADC**Operation: $A + M + C \rightarrow A, C$

N Z C I D V

✓ ✓ ✓ - - ✓

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	ADC # Oper	69	2	2
Zero Page	ADC Oper	65	2	3
Zero Page, X	ADC Oper, X	75	2	4
Absolute	ADC Oper	6D	3	4
Absolute, X	ADC Oper, X	7D	3	4*
Absolute, Y	ADC Oper, Y	79	3	4*
(Indirect, X)	ADC (Oper, X)	61	2	6
(Indirect), Y	ADC (Oper), Y	71	2	5*

* Add 1 if page boundary is crossed.

AND*"AND" memory with accumulator***AND**

Logical AND to the accumulator

Operation: $A \wedge M \rightarrow A$

N Z C I D V

✓ ✓ - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	AND # Oper	29	2	2
Zero Page	AND Oper	25	2	3
Zero Page, X	AND Oper, X	35	2	4
Absolute	AND Oper	2D	3	4
Absolute, X	AND Oper, X	3D	3	4*
Absolute, Y	AND Oper, Y	39	3	4*
(Indirect, X)	AND (Oper, X)	21	2	6
(Indirect), Y	AND (Oper), Y	31	2	5

* Add 1 if page boundary is crossed.

ASL*ASL Shift Left One Bit (Memory or Accumulator)***ASL**Operation: $C \leftarrow \begin{array}{|c|c|c|c|c|c|c|c|} \hline 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \end{array} \leftarrow 0$

N Z C I D V

✓ ✓ ✓ - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Accumulator	ASL A	0A	1	2
Zero Page	ASL Oper	06	2	5
Zero Page, X	ASL Oper, X	16	2	6
Absolute	ASL Oper	0E	3	6
Absolute, X	ASL Oper, X	1E	3	7

BCC

BCC Branch on Carry Clear

BCC

Operation: Branch on C = 0

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BCC Oper	90	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BCS

BCS Branch on carry set

BCS

Operation: Branch on C = 1

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BCS Oper	B0	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

BEQ

BEQ Branch on result zero

BEQ

Operation: Branch on Z = 1

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BEQ Oper	F0	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

BIT

BIT Test bits in memory with accumulator

BITOperation: $A \wedge M, M_7 \rightarrow N, M_6 \rightarrow V$

Bit 6 and 7 are transferred to the status register.

N Z C I D V

If the result of $A \wedge M$ is zero then Z = 1, otherwise $M_7 \checkmark \text{ --- } M_6$

Z = 0

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

BMI

BMI Branch on result minus

BMI

Operation: Branch on N = 1

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BMI Oper	30	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BNE

BNE Branch on result not zero

BNE

Operation: Branch on Z = 0

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BNE Oper	D0	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BPL

BPL Branch on result plus

BPL

Operation: Branch on N = 0

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BPL Oper	10	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BRK

BRK Force Break

BRK

Operation: Forced Interrupt PC + 2 + P +

N Z C I D V

--- 1 ---

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	BRK	00	1	7

1. A BRK command cannot be masked by setting I.

BVC*BVC Branch on overflow clear***BVC**

Operation: Branch on V = 0

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BVC Oper	50	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BVS*BVS Branch on overflow set***BVS**

Operation: Branch on V = 1

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Relative	BVS Oper	70	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

CLC*CLC Clear carry flag***CLC**

Operation: 0 → C

N Z C I D V

-- 0 ---

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	CLC	18	1	2

CLD*CLD Clear decimal mode***CLD**

Operation: 0 → D

N Z C I D V

----- 0 -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

CLI*CLI Clear interrupt disable bit***CLI**

Operation: 0 → I

N Z C I D V

--- 0 ---

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2

CLV

CLV Clear overflow flag

Operation: $\emptyset + V$

N Z C I D V
 - - - - - \emptyset

CLV

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

CMP

CMP Compare memory and accumulator

Operation: $A - M$

N Z C I D V
 ✓ / ✓ / - - -

CMP

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	CMP #Oper	C9	2	2
Zero Page	CMP Oper	C5	2	3
Zero Page, X	CMP Oper, X	D5	2	4
Absolute	CMP Oper	CD	3	4
Absolute, X	CMP Oper, X	DD	3	4*
Absolute, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper), Y	D1	2	5*

* Add 1 if page boundary is crossed.

CPX

CPX Compare Memory and Index X

Operation: $X - M$

N Z C I D V
 ✓ / ✓ / - - -

CPX

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	CPX #Oper	E \emptyset	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

CPY

CPY Compare memory and index Y

Operation: $Y - M$

N Z C I D V
 ✓ / ✓ / - - -

CPY

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	CPY #Oper	C \emptyset	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

DEC

DEC Decrement memory by one

DECOperation: $M - 1 \rightarrow M$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

DEX

DEX Decrement index X by one

DEXOperation: $X - 1 \rightarrow X$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

DEY

DEY Decrement index Y by one

DEYOperation: $Y - 1 \rightarrow Y$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

EOR

EOR "Exclusive-Or" memory with accumulator

EOROperation: $A \oplus M \rightarrow A$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	EOR #Oper	49	2	2
Zero Page	EOR Oper	45	2	3
Zero Page, X	EOR Oper, X	55	2	4
Absolute	EOR Oper	4D	3	4
Absolute, X	EOR Oper, X	5D	3	4*
Absolute, Y	EOR Oper, Y	59	3	4*
(Indirect, X)	EOR (Oper, X)	41	2	6
(Indirect), Y	EOR (Oper), Y	51	2	5*

* Add 1 if page boundary is crossed.

INC*INC Increment memory by one***INC**Operation: $M + 1 \rightarrow M$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	INC Oper	E6	2	5
Zero Page, X	INC Oper, X	F6	2	6
Absolute	INC Oper	EE	3	6
Absolute, X	INC Oper, X	FE	3	7

INX*INX Increment Index X by one***INX**Operation: $X + 1 \rightarrow X$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

INY*INY Increment Index Y by one***INY**Operation: $Y + 1 \rightarrow Y$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	INY	C8	1	2

JMP*JMP Jump to new location***JMP**Operation: $(PC + 1) \rightarrow PCL$ $(PC + 2) \rightarrow PCH$

N Z C I D V

- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR*JSR Jump to new location saving return address***JSR**Operation: $PC + 2 \rightarrow$, $(PC + 1) \rightarrow PCL$ $(PC + 2) \rightarrow PCH$

N Z C I D V

- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Absolute	JSR Oper	20	3	6

LDA

LDA Load accumulator with memory

LDAOperation: $M \rightarrow A$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	LDA # Oper	A9	2	2
Zero Page	LDA Oper	A5	2	3
Zero Page, X	LDA Oper, X	B5	2	4
Absolute	LDA Oper	AD	3	4
Absolute, X	LDA Oper, X	BD	3	4*
Absolute, Y	LDA Oper, Y	B9	3	4*
(Indirect, X)	LDA (Oper, X)	A1	2	6
(Indirect), Y	LDA (Oper), Y	B1	2	5*

* Add 1 if page boundary is crossed.

LDX

LDX Load index X with memory

LDXOperation: $M \rightarrow X$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	LDX # Oper	A2	2	2
Zero Page	LDX Oper	A6	2	3
Zero Page, Y	LDX Oper, Y	B6	2	4
Absolute	LDX Oper	AE	3	4
Absolute, Y	LDX Oper, Y	BE	3	4*

* Add 1 when page boundary is crossed.

LDY

LDY Load index Y with memory

LDYOperation: $M \rightarrow Y$

N Z C I D V

✓ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	LDY #Oper	A0	2	2
Zero Page	LDY Oper	A4	2	3
Zero Page, X	LDY Oper, X	B4	2	4
Absolute	LDY Oper	AC	3	4
Absolute, X	LDY Oper, X	BC	3	4*

* Add 1 when page boundary is crossed.

LSR

LSR Shift right one bit (memory or accumulator)

LSROperation: 0 →

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

 → C

N Z C I D V

0 ✓ ✓ - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

NOP

NOP No operation

NOP

Operation: No Operation (2 cycles)

N Z C I D V

- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

ORA

ORA "OR" memory with accumulator

ORA

Operation: A V M + A

N Z C I D V

✓ ✓ - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	ORA #Oper	09	2	2
Zero Page	ORA Oper	05	2	3
Zero Page, X	ORA Oper, X	15	2	4
Absolute	ORA Oper	0D	3	4
Absolute, X	ORA Oper, X	1D	3	4*
Absolute, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	01	2	6
(Indirect), Y	ORA (Oper), Y	11	2	5

* Add 1 on page crossing

PHA

PHA Push accumulator on stack

PHA

Operation: A +

N Z C I D V

- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

PHP

PHP Push processor status on stack

PHP

Operation: P+

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	PHP	08	1	3

PLA

PLA Pull accumulator from stack

PLA

Operation: A +

N Z C I D V

✓ / -----

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

PLP

PLP Pull processor status from stack

PLP

Operation: P +

N Z C I D V

From Stack

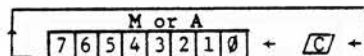
Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

ROL

ROL Rotate one bit left (memory or accumulator)

ROL

Operation:



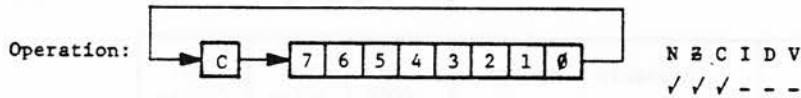
N Z C I D V

✓ / ✓ / -----

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

ROR

ROR Rotate one bit right (memory or accumulator)

ROR

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Accumulator	ROR A	6A	1	2
Zero Page	ROR Oper	66	2	5
Zero Page,X	ROR Oper,X	76	2	6
Absolute	ROR Oper	6E	3	6
Absolute,X	ROR Oper,X	7E	3	7

RTI

RTI Return from interrupt

RTI

Operation: P+ PC+

N Z C I D V
From Stack

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	RTI	40	1	6

RTS

RTS Return from subroutine

RTS

Operation: PC+, PC + 1 → PC

N Z C I D V
- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	RTS	60	1	6

SBC

SBC Subtract memory from accumulator with borrow

SBCOperation: $A - M - \bar{C} \rightarrow A$ N Z C I D V
✓ / ✓ - - /Note: \bar{C} = Borrow

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Immediate	SBC #Oper	E9	2	2
Zero Page	SBC Oper	E5	2	3
Zero Page, X	SBC Oper, X	F5	2	4
Absolute	SBC Oper	ED	3	4
Absolute, X	SBC Oper, X	FD	3	4*
Absolute, Y	SBC Oper, Y	F9	3	4*
(Indirect, X)	SBC (Oper, X)	E1	2	6
(Indirect), Y	SBC (Oper), Y	F1	2	5*

* Add 1 when page boundary is crossed.

SEC

SEC Set carry flag

SEC

Operation: 1 → C

N Z C I D V

-- 1 --

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

SED

SED Set decimal mode

SED

Operation: 1 → D

N Z C I D V

---- 1 --

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

SEI

SEI Set interrupt disable status

SEI

Operation: 1 → I

N Z C I D V

--- 1 --

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

STA

STA Store accumulator in memory

STA

Operation: A → M

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	STA Oper	85	2	3
Zero Page, X	STA Oper, X	95	2	4
Absolute	STA Oper	8D	3	4
Absolute, X	STA Oper, X	9D	3	5
Absolute, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper), Y	91	2	6

STX

STX Store index X in memory

STXOperation: $X \rightarrow M$

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	STX Oper	86	2	3
Zero Page, Y	STX Oper, Y	96	2	4
Absolute	STX Oper	8E	3	4

STY

STY Store index Y in memory

STYOperation: $Y \rightarrow M$

N Z C I D V

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Zero Page	STY Oper	84	2	3
Zero Page, X	STY Oper, X	94	2	4
Absolute	STY Oper	8C	3	4

TAX

TAX Transfer accumulator to index X

TAXOperation: $A \rightarrow X$

N Z C I D V

✓ ✓ -----

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

TAY

TAY Transfer accumulator to index Y

TAYOperation: $A \rightarrow Y$

N Z C I D V

✓ ✓ -----

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

TSX

TSX Transfer stack pointer to index X

TSXOperation: $S \rightarrow X$

N Z C I D V

✓ ✓ -----

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

TXA

TXA Transfer index X to accumulator

TXAOperation: $X \rightarrow A$

N Z C I D V

/ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

TXS

TXS Transfer index X to stack pointer

TXSOperation: $X \rightarrow S$

N Z C I D V

- - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

TYA

TYA Transfer index Y to accumulator

TYAOperation: $Y \rightarrow A$

N Z C I D V

/ / - - - -

Addressing	Assembly Language	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

DETAILS

Editor in Detail

The MACROTEA Editor serves several functions. First, of course, is the entry and modification of the source text in the workspace. Other functions include control for the hard copy display, a simple filing system, and commands related to the assembly of the source text.

ENTRY OF TEXT

Numbered lines are put into the MACROTEA workspace in the same way that BASIC lines are entered. Just type the line number, the text, and press RETURN. The longest possible line entry is 80 characters, with 4 characters for the line number, and 76 characters for text. The line number may be from 0 to 9999.

The PET's Screen Editor may be used to change lines on the screen, and RETURN will enter the changed line - again, just like PET BASIC does.

In the examples below, note that the FORMAT command has been set to CLEAR. If you want to try these, first enter:

FO CL (Return)

Examples

First, some simple text entry:

```
10 THIS IS AN EXAMPLE OF SOME TEXT (Return)
20 ENTRY IN MACROTEA. (Return)
PR (Return)
0010 THIS IS AN EXAMPLE OF SOME TEXT
0020 ENTRY IN MACROTEA.
//
```

When you entered PR, MACROTEA listed the lines stored in the workspace. The // is MACROTEA's way of saying that it has completed a command. (In this case, the PR.)

Inserting a line is simple enough:

```
15 WITH SOME ADDITIONS (Return) (We will ignore the Returns from
PR here on.)
0010 THIS IS AN EXAMPLE OF SOME TEXT
0015 WITH SOME ADDITIONS
0020 ENTRY IN MACROTEA.
//
```

And deletion of one line:

20

PR

0010 THIS IS AN EXAMPLE OF SOME TEXT

0015 WITH SOME ADDITIONS

//

Use the PET's Screen Editor (That is, move the cursor & use the INST key) to change Line 10 to:

0010 THIS IS ANOTHER EXAMPLE OF SOME TEX

T

Press Return, and the line will be entered. Note that up to 72 characters may be in a line.

Then there is one "ugly" - if you enter a space before a line number, the Editor thinks you meant a command - try:

(space) 123 HELLO

!ED AT LINE 0015

With a few experiments, you will discover that:

1) All the PET Screen Editor features are functional, including Quote Mode.

2) RVS and OFF will work, but the reversed field text will come back as normal text unless it is inside quote marks.

3) Extra long line numbers will only use the last four characters in the line number. For example, 1234567 HELLO will be entered as 4567 HELLO.

4) Spaces are important. If you enter 10 20, you now have a Line 10 with the text:(space)20.

ENTRY OF COMMANDS

Any line which is entered to MACROTEA without a line number is seen as a command. If the command has any parameters, these must be separated by spaces. All commands and parameters may be abbreviated to two letters - for example, HARD CLEAR can be abbreviated to HA CL.

MACROTEA only looks at the first two characters in each command or parameter - so PR, PRINT, PRUNES, PRIOR, and PREACHER are each seen as the command PRINT.

As with text lines, any command can be edited or re-entered via the PET's Screen Editor.

If MACROTEA cannot decipher a command, an error message will be printed. Here are a few examples:

SMOKEY THE BEAR

!ED AT LINE 0000

GREED

!ED AT LINE 0015

FORMULA ONE

Take heed - the FORMULA ONE was seen as the FO for FORMAT, and the remainder (ONE) was ignored, and the default value (SET) was applied instead.

Sometimes the error will come from inability to understand the parameters that are expected:

PRINT HELLO

!09 AT LINE 0000

If you look this up, MACROTEA was expecting a string of decimal numbers (ie, a line number).

When an error occurs in a command, the AT LINE part of the message should be ignored as it has no meaning.

If you want to abort a command, you may either move the cursor out of the line to a blank one, or press SHIFT-Return.

If you are used to PET BASIC, the operation of MACROTEA will seem a bit odd. Some commands, such as HARD, seem to do nothing - their effects are seen only when some other command is executed, such as PRINT. Another difference is the meaning of spaces. In BASIC, spaces are usually ignored. In MACROTEA, spaces serve as delimiters - that is, values are separated by spaces instead of commas.

TEXT CONTROL COMMANDS

The commands in this section are concerned with manipulating the source text stored in the workspace. In most cases these commands can be regarded as the pure "text editor" part of MACROTEA.

A Minor Bug:

In entering commands to the Editor, you may inadvertently enter a one letter line, like E or X or whatever. The first time you do this, MACROTEA will behave oddly. The second and later times you do this, MACROTEA will respond normally with the !ED error.

Get into BASIC via BR and X:

```
READY.
SYS36864          (MACROTEA Cold Start)
0800-57FC 5800-67FC 0000
0800 5800
Q                (Any letter will do)
0001Q
```

```
ATCH AREA. &*& %&*&*& (some letters and graphics junk)
!ED AT LINE 0001
```

```
Q                (Try it again)
```

```
!ED AT LINE 0000 (Normal response)
```

AUTO (Line Number Increment)

AUTO will provide the next line number in sequence after the entry of any numbered line. The cursor will be immediately after the AUTO generated line number.

If the next number in sequence is larger than 9999, the number "wraps around" through 0000.

To turn AUTO off, enter either AUTO 0 or just AUTO.

To "escape" a line number generated by AUTO, use SHIFT-Return. Or, you can type two slashes (//) followed by a normal Return.

Examples:

Clear the workspace via CL, and then enter:

AUTO 10

15 THIS IS ONE WAY OF GETTING MY NUMBERS RIGHT.
0025M

The M indicates the cursor. AUTO used the line number provided, 15, and added the interval, 10, to get the new line number, 25.

We can continue with this:

0025 SO I AM TOLD BY THE ILLUSTRIOUS
0035 MACROTEA!
0045 (SHIFT-Return)
M

If I continue with some other number, AUTO follows along:

16 THE MORE THE MERRIER
0026M

You will notice that AUTO takes no heed of any previously entered text. If you are inserting text with AUTO, beware of interleaving lines or obliterating lines.

Turning AUTO off is simple enough:

AUTO

22 HELLO THERE
M

If you give AUTO a parameter, it expects a line number:

AUTO OFF

!09 AT LINE 0022

PAGE 4 6

AUTO doesn't care if you enter text with the line numbers, which provides one way to delete text.

AU 5

10

0015

0020M

As a final touch, observe the wrap-around:

AUTO 100

9850 ROW, ROW YOUR BOAT

9950 MERRILY, MERRILY ALONG

0050 THE STREAM

0150M

NUMBER (Starting Line Number) (Line Number Increment)

NUMBER rennumbers the text lines in the workspace. If a new line number will be more than 9999, NUMBER will stop with an error message - and leave the text partially renumbered.

If NUMBER gives you a messy result, try NUMBER 0 10 or NUMBER 0 1. This will restore your workspace to a reasonable set of sequential line numbers.

To understand NUMBER, see the procedure below:

1) Using the Starting Line Number, begin at the start of the workspace and find a line equal to or larger than the Starting Line Number. (If you don't find one, you are done.)

2) Add the Line Number Increment to the Starting Line Number's value and replace the current line's number. Go through the rest of the workspace, adding the Line Number Increment each time.

3) If no Line Number Increment is provided, use an increment of zero. If the new line number is more than 9999, print an error message and quit, leaving the text file partially renumbered.

The best way to follow all this is with some examples:

Examples:

Suppose we have this in the workspace:

```
PR
0010 JUST ON TIME, THE KING AND HIS
0020 GRAND DUKE ARRIVED AT THE AIRPORT
0030 IN QUITO, PERU.
0040 AN IMPORTANT MEETING WAS ABOUT
0050 TO BEGIN WITH THE INDIANS OF
0060 THE HIGH ANDES.
//
```

(Note that FORMAT was CLEARED via FO CL.)

Now, let's see NUMBER in action:

NU 10 5

```
PR
0015 JUST ON TIME, THE KING AND HIS
0020 GRAND DUKE ARRIVED AT THE AIRPORT
0025 IN QUITO, PERU.
0030 .....(etc)....
0035 .....(etc)....
0040 .....(etc)....
```

PAGE 48

Note that the first line number ended up as 15. To make this more clear, try:

```
NU 27 10
```

```
PR
0015 ..... (You know what this text is,)
0020 ..... (and I am lazy.)
0025 .....
0037 .....
0047 .....
0057 .....
```

If you look at the procedure, the first line equal to or larger than 27 was Line 30. The increment, 10, was added to 27 to give 37, and so Line 30 was renumbered to 37. The following lines then became 47 and 57.

There is a reason to this madness - if you have a long text in the workspace, you can by successive applications of NUMBER eventually get the text into a series of blocks, like:

```
1000
1010
1020
...
...
2000
2010
2020
...
...
```

As an exercise, make the example text turn into this form!

Now, to make NUMBER crash and how to get back home again:

```
NU 10 10
```

```
PR
0020 .....
0030 .....
0040 .....
0050 .....
0060 .....
0070 .....
```

```
NU 45 4000
```

```
!10 AT LINE 2045
```

```
PR
0020 .....
0030 .....
0040 .....
4045 .....
8045 .....
2045 .....
```

As you can see, the last line didn't work out too well. Since NUMBER only changes line numbers, the order of the lines of text isn't changed. (If you move the cursor into Line 2045, and press Return, you will now have a second Line 2045.)

Recovery is simple - just do a NU 10 10 and things are restored. (It is safest to use NU 0 10 or NU 0 1)

Another problem appears if you don't provide a Line Number Increment. Try this one:

```
NU 12
```

```
PR
0012 .....
0012 .....
0012 .....
0012 .....
0012 .....
0012 .....
```

Can you see the way out of this one?

NUMBER expects at least one parameter, or you will get an error:

```
NUMBER
```

```
!!! AT LINE 0000
```

Note: The commands COPY and MOVE will generate blocks of lines with the same number in the workspace. Use NUMBER to straighten things out.

COPY (Destination Line Number) (Start Line Number) (End Line Number)

COPY makes a copy of the lines in the workspace from the Start Line Number to the End Line Number. These lines are placed in the text just following the Destination Line Number. All Copied lines will be given the Destination Line Number.

If the Start Line Number and the End Line Number are out of order, COPY will copy one line - if the line's number matches the Start Line Number. If the End Line Number is omitted, the same thing will happen.

Use NUMBER to re-order the lines after COPY.

Examples:

Suppose the following is in the workspace. (And FORMAT is CLEAR.)

```
PR
0010 LINE TEN
0020 LINE TWENTY
0030 LINE THIRTY
0040 LINE FORTY
0050 LINE FIFTY
0060 LINE SIXTY
//
```

Now, make a copy of Lines 10 to 30 and put it after Line 60:

```
CO 60 10 30
PR
0010 LINE TEN
0020 LINE TWENTY
0030 LINE THIRTY
0040 LINE FORTY
0050 LINE FIFTY
0060 LINE SIXTY
0060 LINE TEN
0060 LINE TWENTY
0060 LINE THIRTY
```

Just to see what happens, try a copy of Lines 40 to 60 starting at 25.....

```
CO 25 40 60
PR
0010 LINE TEN
0020 LINE TWENTY
0025 LINE FORTY
0025 LINE FIFTY
0025 LINE SIXTY
0030 LINE THIRTY
0040 LINE FORTY
0050 LINE FIFTY
0060 LINE SIXTY
0060 LINE TEN
0060 LINE TWENTY
0060 LINE THIRTY
```

First, the copied lines were numbered according to the Destination Line Number. Second, once COPY saw a Line 60, it stopped.

NUMBER can be used to clean this up somewhat. (Do it - the next example assumes you used NU 0 10.)

COPY needs at least two parameters -

COPY

!11 AT LINE 0000

COPY 1

!11 AT LINE 0000

COPY SMOKEY THE BEAR

!09 AT LINE 0000

and that the parameters be numbers.

If you don't provide an End Line Number, or there aren't any numbers in the area you are copying, COPY won't do anything. With the workspace NUMBERed by 10s (ie, 10 20 30 etc), try:

CO 25 55

PR

(nothing happened, so I won't show it to you)

CO 25 31 39

PR

(ditto - just the same old 10, 20, etc.)

However, be careful if a line is present at the Start Line Number:

CO 15 60

PR

0010 LINE TEN

0015 LINE THIRTY

0020 LINE TWENTY

0030 LINE FORTY

.... etc....

If lines with the same number are in the ones to be copied, they will come along too. Try this sequence:

CL (Removes the text from the workspace)

10 HELLO

20 THERE

CO 5 10 20

```
PR
0005 HELLO
0005 THERE
0010 HELLO
0020 THERE
//
```

CO 4 4 15

```
PR
0004 HELLO
0004 THERE
0004 HELLO
0005 HELLO
0005 THERE
0010 HELLO
0020 THERE
```

COPY will not tolerate an attempt to COPY lines to a destination within the lines being copied.

Assume we have:

```
0010 J
0020 K
0030 L
CO 15 10 20
```

!12 AT LINE 0015

The range, Lines 10 to 20, include the destination, Line 15.

Challenge: Suppose my text looks like this:

```
0000 THIS IS THE FIRST LINE
0001 THIS IS THE SECOND LINE
0002 THIS IS THE THIRD LINE
0003 THIS SHOULD BE THE ZERO LINE
```

How do you COPY Line 3 so that it is BEFORE Line 0? (Hint - you have to do something else first!)

Another Challenge: Given the text below, how do you use COPY (in a minimum number of steps) to eventually get the !OF AT LINE xxxx error? (ie, you are out of memory.)

100 HELLO

MOVE (Destination Line Number) (Start Line Number) (End Line Number)

MOVE takes the text from the Start Line Number to the End Line Number and places it following the Destination Line Number. The MOVED lines are given the Destination Line Number.

If the Start Line Number and the End Line Number are out of order, MOVE will copy one line - if the line's number matches the Start Line Number. If the End Line Number is omitted, the same thing will happen.

Use NUMBER to re-order the lines after MOVE.

Examples:

First, take a look at the examples for COPY. MOVE works the same way - but the lines are moved, not copied.

Let's start with:

```
PR
0010 WOMBATS MAKE NICE
0020 FRIENDS IF YOU
0030 TAKE CARE TO PUT THEM
0040 INTO STRONG CAGES
```

Let's use MOVE to reverse the order of all these lines.

MO 50 10 20

```
PR
0030 TAKE CARE TO PUT THEM
0040 INTO STRONG CAGES
0050 WOMBATS MAKE NICE
0050 FRIENDS IF YOU
```

MO 20 40

```
PR
0020 INTO STRONG CAGES
0030 TAKE CARE TO PUT THEM
0050 WOMBATS MAKE NICE
0050 FRIENDS IF YOU
```

MO 60 50

```
PR
0020 INTO STRONG CAGES
0030 TAKE CARE TO PUT THEM
0050 FRIENDS IF YOU
0060 WOMBATS MAKE NICE
```

PAGE 54

MOVE will fail in the same way that COPY does:

MOVE

!11 AT LINE 0000

MOVE 12

!11 AT LINE 0000

MOVE THE EARTH

!09 AT LINE 0000

NU 0 10 (We assume you still have the course on Wombats in the
workspace.)

MO 20 10 30

!12 AT LINE 0020

NOTE: If you are editing a large text file, the use of NUMBER, MOVE, and COPY can play havoc with any hardcopy listing you may have. A little planning & pencil and paper will help.

DELETE (Start Line Number) (End Line Number)

DELETE removes the lines from the Start Line Number to the End Line Number in the workspace. If the Start Line Number only is given, Delete will remove only that single line.

No lines will be removed if there aren't any with the indicated numbers in the workspace.

To understand DELETE, see the sequence below:

1) Scan the workspace until the Start Line Number (or a larger line number) is found.

2) Delete lines until a line with a line number equal to or larger than the End Line Number is found.

3) If the workspace line's number equals the End Line Number, delete this line too.

4) Finished.

Examples:

DELETE works very simply - provided the line numbers in the workspace are in order and each line number is unique. (After COPY or MOVE, DELETE acts a bit strangely.)

If we have:

```
PR
0010 ONE
0020 TWO
0030 THREE
0040 FOUR
0050 FIVE
0060 SIX
//
```

Removal of Lines 30 to 50 is simple:

```
DE 30 50
```

```
PR
0010 ONE
0020 TWO
0060 SIX
```

Removal of one line is also easy:

```
DE 10
```

```
PR
0020 TWO
0060 SIX
```

If the line isn't there, DELETE won't remove it:

DE 40

PR

0020 TWO

0060 SIX

Check for yourself that DELETE 60 1000 and DELETE 0 25 will work as expected.

DELETE likes to see at least one line number:

DE

!11 AT LINE 0010

DE F00

!09 AT LINE 0000

If line numbers aren't unique, DELETE will work a little strangely. (You are advised to NUMBER your text first!) Here are some examples:

Suppose:

PR

0005 HELLO 1

0005 HELLO 2

0005 HELLO 3

0010 HELLO 4

0010 HELLO 5

0010 HELLO 6

0020 HELLO 7

0020 HELLO 8

0020 HELLO 9

(How to make this text? Start with
100 HELLO 1, 110 HELLO 2, etc. Then
use MOVE.)

Now try a DELETE 20

DE 20

PR

0005 HELLO 1

0005 HELLO 2

0005 HELLO 3

0010 HELLO 4

0010 HELLO 5

0010 HELLO 6

0020 HELLO 8

0020 HELLO 8

DELETE removed the first Line 20 it found.

Now try:

DE 20 20

PR

0005 HELLO 1

0005 HELLO 2

0005 HELLO 3

0010 HELLO 4

0010 HELLO 5

0010 HELLO 6

Here, DELETE removed the first Line 20 (HELLO 8) and then the second Line 20 (HELLO 9).

Then there's:

DE 5 10

PR .

0010 HELLO 5

0010 HELLO 6

Here, DELETE removed all of the Line 5's and the first Line 10.

MORAL TO THIS LESSON: Make your text area have reasonable line numbers before you go and DELETE something.

CLEAR

CLEAR deletes all of the lines in your workspace. There's no second chance, so be sure of what you are doing.

Example:

Suppose:

```
PR
0010 WANT TO SEE THE FASTEST
0020 DELETE IN THE WEST?
0030 WANT TO SEE IT AGAIN?
```

CL

```
PR
//
```

```
FIND (Delim)(Search String)(Delim)(Don't Care Char)
      (Print Control)(Start Line Number)(End Line Number)
```

FIND searches for a specific string within the workspace and indicates where it was found.

The sequence (Delim)(Search String)(Delim) specifies the string being looked for. For example, if you are looking for HERE IS, the sequence can be *HERE IS* or /HERE IS/ or AHERE ISA.

The character % placed in the Search String means that the character in this position is not important and is to be ignored. This is a "don't care character." For example, if you were looking for F%X, FIND would locate FOX, FIX, FAX, FXX, etc.

If you need to change the Don't Care Character, place the sequence %(char) after the second Delimiter. If \$ were to be the Don't Care Character, a typical command would look like this:
FIND *F\$X* %\$

If the character # is placed in the FIND command, the located lines will not be printed on the screen.

To FIND over a range of line numbers, enter the Start Line Number and the End Line Number.

When FIND locates a line, the line is printed on the screen

After all of the workspace has been examined, FIND reports the number of occurrences of the Search String.

The Examples will tell you about these features in more detail.

Examples:

FIND isn't as formidable as the description above might seem. In most cases, you will be using the simpler variations.

Suppose the text in the workspace is:

```
0010 THERE ONCE WAS A PRETTY LITTLE
0020 COMPUTER. HER CRT WAS ROSY AND
0030 HER KEYBOARD WAS A LUSCIOUS PEACH
0040 HUE. WHEN SHE RAN HER PROGRAMS,
0050 SHE WOULD USUALLY RUN THEM WITH
0060 GRACE AND CORRECTNESS. AH, BUT IF
0070 SHE SAW A BUG, SHE WOULD SCREAM
0080 AND BE HORRID!
```

Let's first explore finding strings.

```
FIND *PRETTY*
0010 THERE ONCE WAS A PRETTY LITTLE
//0001
```

(We will ignore the
40 char screen width.)

So what does all this mean. First, to look for PRETTY, it must be enclosed in some character that isn't in the Search String - so the asterisk (*) was used. Second, one line, 0010, contains the word PRETTY - and FIND printed the line.

Since FIND prints the line on the screen in the same way that PRINT does, you can move the cursor up into the line and edit the line with the PET's Screen Editor. This is a nice method of editing lines which are similar but need different changes.

The last part, //0001, tells (in Decimal) the number of times PRETTY was found.

Now for some variations on this theme:

```
FIND ZPRETTYZ
FIND !A PRETTY!
```

will give the same result as above. (Try it and see.)

But be careful - not all characters will work as Delimiters:

```
FIND (PRETTY)
!15 AT LINE 0080
```

Of course, the Delimiter must be the same at both ends:

```
FIND *PRETTY!
!15 AT LINE 0000
```

Each instance of a located Search String will result in a line being displayed:

```
FIND *WAS*
0010 THERE ONCE WAS A PRETTY LITTLE
0020 COMPUTER. HER CRT WAS ROSY AND
0030 HER KEYBOARD WAS A LUSCIOUS PEACH
//0003
```

This is true for repetitions in one line as well:

```
FIND 'TT'
0010 THERE ONCE WAS A PRETTY LITTLE
0010 THERE ONCE WAS A PRETTY LITTLE
```

For fun, I leave it to you to see what happens with:

```
FIND *E*
```

To locate words, suffixes or prefixes, include a space in your Search String. Try this sequence:

```
FIND *HE*
(You will find 10 'hits' for *HE*.)
```

```
FIND *HE *
(Now it is 4 'hits'.)
```

```
FIND * HE*
(Three this time.)
```

This trick won't work for words at the end of a line:

```
FI * IF *
//0000
```

(IF is the last word in Line 0060.)

The character '%' is a Don't Care Character if you put it into a Search String. Letters in this position will always match.

```
FIND *R%N*
0040 HUE. WHEN SHE RAN HER PROGRAMS,
0050 SHE WOULD USUALLY RUN THEM WITH
```

Here, Line 40 has the word RAN, and Line 50 has the word RUN. The central letter was ignored.

If you really want fun, try:

```
FI *%*
```

For the rare case where you want to search for the % itself, the Don't Care Character can be changed by putting %(char) after the Search String:

```
FIND *RUN* %U
```

This gives the same thing that FIND *R%N* did.

To look for a Search String in a group of lines, just provide the expected range of line numbers:

```
FI *HE* 10 20
0010 THERE ONCE WAS A PRETTY LITTLE
0020 COMPUTER. HER CRT WAS ROSY AND
//0002
```

```
FIND *HE* 20
0020 COMPUTER. HER CRT WAS ROSY AND
```

PAGE 62

```
FI *E* 20
0020 COMPUTER. HER CRT WAS ROSY AND
0020 COMPUTER. HER CRT WAS ROSY AND
```

```
FI *HE* 55
//0000
```

The last one indicated a line that doesn't exist - so the search failed.

I leave it up to you to discover what happens with workspaces after COPY and MOVE. (I suspect it works like PRINT does.)

Sometimes it is handy to just count the number of 'hits' and not to list them. The character '#' placed after the Search String will accomplish this:

```
FIND *HE* #
//0010
```

Here's a neat trick:

```
FI !%# #
//0241
```

This means to find all occurrences of any character - so every letter in the workspace is counted - 0241 is the number of characters in the workspace.

This Print Control character must be put after the Don't Care Character if you are using both options at once:

```
FIND *RUN* %U #
//0002
```

```
FIND *RUN* # %U
```

```
!09 AT LINE 0000
```

All of this can be used with line numbers as well - just be sure the Start and End Line Numbers come after everything else.

So - you've just learned the hardest command in MACROTEA. (Except for EDIT, which comes next!)

```

EDIT (Delim)(Search String)(Delim)(Replace String)(Don't Care Char)
      (Print Control/Subcommands)(Start Line Number)
      (End Line Number)

```

EDIT searches the workspace for any occurrences of the Search String, and replaces the Search Strings with the Replace String. The Delimiter is any character that doesn't appear in the Search or Replace Strings.

The Don't Care Char will match any character in the workspace that "fits" in the Search String. (Just like the Don't Care Char in FIND.)

To suppress printing of any altered lines, use # for the Print Control. If # is not present, each line that is altered will be printed as it was before any changes were made.

If the asterisk * is used, EDIT will print each line in which the Search String is found, and then prompt you with another asterisk. You may now enter a Sub-Command, from one of the list below:

- A - Change the Found String to the Replace String and continue.
- D - Delete the line entirely, and then continue.
- M - Don't change the Found String, and continue.
- S - Skip to the next line and continue.
- X - Quit - and return to Editor Command Mode.

The Sub-Commands permit selective EDIT of the workspace.

To EDIT over a range of lines, provide the Start Line Number and the End Line Number.

Note that the Print Control/Sub-Commands are mutually exclusive. You can apply one or the other, but not both.

Examples:

As with FIND, EDIT will usually be used in its simpler forms. To get started, here is a rather concocted workspace:

```

PR
0010 THEY HAD TO GLOWER AT THE POWER
0020 OF THE TOWER AS IT GOT LOWER AND
0030 LOWER TO CRUSH A FLOWER.

```

If you think that's strange, just wait.....

The simplest EDIT is the most powerful - it simply changes all occurrences of the Search String to the Replace String. Here goes:

```
EDIT *WER*TERRA*
0010 THEY HAD TO GLOWER AT THE POWER
0010 THEY HAD TO GLOTERRA AT THE POWER
0020 OF THE TOWER AS IT GOT LOWER AND
0020 OF THE TOTERRA AS IT GOT LOWER AND
0030 LOWER TO CRUSH A FLOWER.
0020 LOTERRA TO CRUSH A FLOWER.
//0006
```

As most of this was described in FIND, we will be brief here. EDIT lists each line each time the Search String is found. Then the change is made.

The display is the same as FIND produces. You can move the cursor up into the lines and change them via the PET's Screen Editor if needs be. HOWEVER, doing so will re-enter the lines as displayed, and the EDIT changes will not be incorporated. In short, what you see is what you get with the Screen Editor.

Since each line has two occurrences of WER, each line is shown twice. The first time no changes are present, and the second time shows the first change is already in effect.

The //0006 tells how many times the Search String was found.

Here is the changed result of the above EDIT:

```
PR
0010 THEY HAD TO GLOTERRA AS THE POTERRA
0020 OF THE TOTERRA AS IT GOT LOTERRA AND
0030 LOTERRA TO CRUSH A FLOTERRA.
```

The Search String can use the % character as a Don't Care Character. If % is put into the Replace String, it will end up in the text. Here goes:

```
EDIT *%T*Z%P*
```

(Do it yourself to see the EDIT printout.)

We end up with:

```
PR
0010Z%PHEY HADZ%PO GLZ%PERRA Z%PZ%PHE PZ%PERRA
0020 OFZ%PHEZ%PZ%PERRA AS Z%P GZ%P LZ%PERRA AND
0030 LZ%PERRAZ%PO CRUSH A FLZ%PERRA.
```

The Search String of %T effectively found every letter T in the text. The Replace String was Z%P - and the % is not a Don't Care Char in the Replace String.

There are two ways to get our text back:

```
EDIT .Z%P.TO.
EDIT .Z%P.TO. %!
```

In the first case, it is unlikely that some other form of Z?P exists so this is reasonably safe. The second case makes completely sure by changing the Don't Care Char to one that isn't in the text. Note that a different delimiter is used, the period.

Here's the result:

```
PR
0010TOHEY HADTOO GLTOERRA TOTOHE PTOERRA
0020 OFTOHETOTOERRA AS TO GTO LTOERRA AND
0030 LTOERRA TOO CRUSH A FLTOERRA.
```

OOPS!!!! - This points out a moral - Be Careful with EDIT. In fact, the original text cannot be restored due to the use of % in the Search String.

Here is one attempt:

```
EDIT :TO:T: #
//0015
```

Here the # suppressed the listing of a line for each occurrence of the Search String.

After application of ED :TERRA:OWER:, we end up with:

```
PR
0010THEY HADTO GLOWER TTHE POWER
0020 OFTHETOWER AS T GT LOWER AND
0030 LOWERTO CRUSH A FLOWER.
```

EDIT may be used to delete material by making the Replace String a null:

```
ED ?THE?? #
//0003
PR
0010Y HAD TO GLOWER T POWER
0020 OFTOWER AS T GT LOWER AND
0030 LOWERTO CRUSH A FLOWER.
```

But you can't get away with:

```
ED **H*                (Attempting a Null Search String)

!15 AT LINE 0030
```

You can easily verify that EDIT will change a given range of lines if you supply the Start and End Line Numbers. One safe way to do this is to specify a non-substitution:

```
EDIT :T:T: 10 20
```

There are some cases where EDIT will make a line longer. Suppose the workspace contains:

```
PR
0010 XXXXXXXXXX
0020 XXXXXXXXXX
```

EDIT *X*YY*

(junk)

```
PR
0010 YYYYYYYYYYYYYYYYYYYY
0020 YYYYYYYYYYYYYYYYYYYY
```

The question is, can this go on forever?

EDIT *Y*ABCDEF*

(more junk)

```
PR
0010 ABCDEFABCDEFABCDEFABCDEFABCDEF
ABCDEFABCDEFABCDEFABCDEFABCDEFAB
0020 ABCDEFABCDEFABCDEFABCDEFABCDEF
ABCDEFABCDEFABCDEFABCDEFABCDEFAB
```

No, EDIT will limit a line to 72 characters - if the insertion goes beyond, the extra characters are thrown away.

Beware of using EDIT if your text has lines with the same line number - EDIT will go into an infinite loop! To escape, press the DEL key.

```
CL
10 HELLO
20 HELLO
CO 30 10 20
```

```
PR
0010 HELLO
0020 HELLO
0030 HELLO
0030 HELLO
//
```

EDIT *HELLO*THERE*

(poof! EDIT gets stuck in Line 30) Press DEL to get out.

```
PR
0010 THERE
0020 THERE
0030 THERE
0030 HELLO
//
```

Sub-Command Examples:

EDIT will stop after each Found String if you follow the Replace String with an asterisk:

EDIT .HELLO.THERE. * is an example.

You can then enter a Sub-Command to decide what to do with this particular case. Let's go back to the original text and see how this works:

```
PR
0010 THEY HAD TO GLOWER AT THE POWER
0020 OF THE TOWER AS IT GOT LOWER AND
0030 LOWER TO CRUSH A FLOWER.
```

```
EDIT *OWER*AMBLE* *
0010 THEY HAD TO GLOWER AT THE POWER
*
```

EDIT found the first OWER (At GLOWER) and printed the line. Now EDIT is waiting for you to select an action.

```
*A
0010 THEY HAD TO GLAMBLE AT THE POWER
*
```

The Sub-Command A tells EDIT to make the change for this instance of the Found String. If you want to ignore the Found String, use M.

```
*M
0020 OF THE TOWER AS IT GOT LOWER AND
*S
0030 LOWER TO CRUSH A FLOWER.
```

The Sub-Command S told EDIT to skip this line - so the second instance of OWER (in LOWER) was ignored - and we are now at Line 30.

```
*D
//0004
PR
0010 THEY HAD TO GLAMBLE AT THE POWER
0020 OF THE TOWER AS IT GOT LOWER AND
//
```

The D Sub-Command deletes the current line. Naturally any preceding changes in a deleted line will be lost. When EDIT was finished, four Found Strings were detected.

```
ED .T.. *
0010 THEY HAD TO GLAMBLE AT THE POWER
*X
//0001
```

The final Sub-Command is X to exit EDIT.

You may include Start and End Line Numbers with the Sub-Commands form of EDIT. This will not be illustrated here in any detail. One example:

```
EDIT .T.TEETH. * 10 20
```

This will do the Sub-Command form of EDIT for Lines 10 through 20.

DISPLAY OF THE WORKSPACE

This group of Editor Commands is concerned with the presentation of the text in the workspace to the PET screen or as listed to a printer.

If you are using a printer with MacroTeA (it is really quite convenient), there are some ritual items that need observance - or the Gods of MacroTeA will frown upon you!

MacroTeA Printer Rituals:

- 1) Thy Printer must be an IEEE-488 compatible device.
- 2) Thy Printer Device Number must be 4.(or see below)
- 3) If Thy Printer needs to be fed unusual characters, do it in this manner:
 - a. Get into BASIC.
 - b. OPEN 1, (Printer's Device Number)
 - c. PRINT#1, (Set up characters)
 - d. Now start MacroTeA.

For example, with a COMPRINT which needs to be started in non-paginate mode to prevent flying form-feeds in the MacroTeA listings. The ritual becomes:

```
OPEN 4,4
PRINT#4, CHR$(30)"HELLO THERE"
SYS36864 (cold start)
```

With a Skyles PAL or PAL 80 printer, no ritual needs to be done.

For a BASE2 printer, it has been reported to me by Ltjg. P.J. Rovero that the best ritual is:

```
OPEN 5,4: Print#5, CHR$(27); CHR$(55);
CHR$(27); CHR$(66); CHR$(27); CHR$(68):
SYS36998 (warm start)
```

The Printer Ritual is handy for checking if the printer is operating correctly.

The Printer device number may be changed by "POKE32417,(dev. #)" in BASIC or by placing the device # in \$7EA1 from the monitor. Remember that the device number is reset to 4 on every cold start of MacroTeA.

Copyright 1982 Skyles Electric Works 8/14/82

(c)1980 Skyles Electric Works
6/10/80

PRINT (Start Line Number)(End Line Number)

PRINT prints the contents of the workspace from the Start Line Number to the End Line Number. If only one line number is given, only the specified line will be printed.

If no line numbers are provided, PRINT will list the entire workspace.

The format of PRINT's output is modified by the commands FORMAT and MANUSCRIPT.

Examples:

To see what's in the workspace, use PRINT without a line number:

```
PRINT
0010 ONE
0020 TIME
0030 TOO
0040 MANY
0050 FOR
0060 IMAGINATIVE
0070 EXAMPLES
//
```

PRINT may be abbreviated to PR. Here is a listing of Line 30:

```
PR 30
0030 TOO
//
```

A range of lines is easily specified:

```
PR 15 45
0020 TIME
0030 TOO
0040 MANY
//
```

PR 0 (line) and PR (line) 9999 will list from the start of the workspace to the specified line number, and from the line number to the end of the workspace, respectively.

If COPY or MOVE have been applied and some line numbers are identical, PRINT acts like this:

```
MO 15 50 70
PR
0010 ONE
0015 FOR
0015 IMAGINATIVE
0015 EXAMPLES
0020 TIME
0030 TOO
0040 MANY
//
```

```
PR 10 15
0010 ONE
0015 FOR
0015 IMAGINATIVE
0015 EXAMPLES
//
```

```
PR 15 20
0015 FOR
0015 IMAGINATIVE
0015 EXAMPLES
0020 TIME
//
```

But beware of:

```
PRINT 15
0015 FOR
//
```

PRINT listed the first Line 15 it found. PR 15 15 will list all of the Line 15's.

In many of the examples for other commands, PRINT is used to show the results. The // at the end of a listing will often be omitted.

HARD (SET/CLEAR/PAGE) (Start Page Number)

HARD tells MACROTEA to output both to the PET Screen and to a printer. The options are:

SET - All of MACROTEA's output will be sent to the printer. When SET, MACROTEA will keep track of the number of lines printed, and will paginate the output on a basis of a 66 line page, with 58 lines of output per page.

None of your input to MACROTEA will appear on the printer.

CLEAR - This disables the SET mode.

PAGE - If HARD is SET, the command HARD PAGE will send 66 line feed character to the printer.

The Start Page Number may be given to any variation of HARD. The current page number will be updated to the new value. Page numbers may be from 00 to 99.

FORMAT (SET/CLEAR)

FORMAT is used to tabulate MACROTEA's output when PRINT or ASSEMBLE produces a listing of the workspace. The SET option forces tabulations (Sent as spaces to the printer) to provide a readable listing of assembler code. The CLEAR option will output the workspace as it is stored internally.

If MANUSCRIPT is used to remove line numbers in PRINT, FORMAT will remain active. In an ASSEMBLE listing, FORMAT controls the output after the line number. FORMAT SET/CLEAR does not affect the memory usage - only the form of the output.

FORMAT CLEAR is the default when MACROTEA starts.

If you perform an assembly, FORMAT will become SET and will remain that way until you CLEAR it.

Examples:

First, let's enter a very short text to help see how FORMAT works:

```
CL
1Z123456789012345678901234567890
2LABEL LDA #FOO ;COMMENT
FO CL
PR
0001Z123456789012345678901234567890
0002LABEL LDA #FOO ;COMMENT
//
```

Line 1 is intended to help keep track of where the columns end up.
Line 2 is an example of some assembly code.

When FORMAT is CLEAR, the text is printed in just the same way it was entered.

```
FORMAT SET
PRINT
0001Z123456789012345678901234567890
0002LABEL      LDA #FOO      ;COMMENT
//
```

The label field is not tabulated. The Op-Code, which is always assumed to start after the first space, is moved to begin at column 10. Comments, which are preceded with semicolons, are moved to the next available tab stop, which is at column 34.

You can experiment with other combinations to find where the tab columns are.

MANUSCRIPT (SET/CLEAR)

If MANUSCRIPT is SET, PRINT will not display the line numbers. This permits the Editor to be used for normal text, such as letters. If MANUSCRIPT is CLEAR, PRINT will display the line numbers in the workspace.

Examples:

Suppose that:

```
CL
FO CL
10 HERE IS A TEST OF
20 THE MANUSCRIPT FEATURE
30 OF MACROTEA.
PR
0010 HERE IS A TEST OF
0020 THE MANUSCRIPT FEATURE
0030 OF MACROTEA.
//
```

Now,

```
MA SET
PR
  HERE IS A TEST OF
  THE MANUSCRIPT FEATURE
  OF MACROTEA.
//
```

When SET, MANUSCRIPT removes the line numbers. If you leave FORMAT SET, the tabulation rules will be in effect:

```
FORMAT SET
PR
      HERE IS A TEST OF
      THE MANUSCRIPT FEATURE
      OF MACROTEA.
//
```

FILES CONTROL COMMANDS

Once a text is completely edited and ready for use, there remains the problem of transferring the information to some other medium, such as a disc or a cassette tape. The commands in this section are mostly concerned with "Files" - or copies of data present on a tape or a diskette.

One unique command, SET, is used to control the allocation of memory in the PET's RAM for the workspace and Symbol table.

SET (Start of Workspace)(End of Workspace)(Start of Symbol Table)
(End of Symbol Table)

The SET command rearranges the MACROTEA memory map for use of the PET's RAM memory. If the parameters are provided, SET rearranges the memory map and then reports the result. If no parameters are given, SET reports the current memory map pointers.

When parameters are entered for SET, they are assumed to be in decimal. If you wish to enter a parameter as a hexadecimal number, precede the number with \$. (% permits entry as a binary number.) SET reports its results in hexadecimal.

When using SET all five address parameters must be entered.

This odd behavior comes from the convention that much machine language code and nearly all addresses are referred to in hexadecimal and not in decimal. In SET, the MACROTEA Assembler's expression analyser is used to let you make entries in hex, and the report is in hex to follow the convention. All other Editor commands will only accept decimal values for their parameters.

No checks of any kind are made for the SET parameters. If care is not taken, the memory map pointers can be given absurd values.

Examples:

When MACROTEA is started, either by the cold or warm start addresses, an implicit SET is performed to report to you the memory map pointers. With the PET in BASIC mode,

```
SYS 36864
```

```
0800-57FC 5800-67FC 0000
0800 5800
```

This report tells you that:

```
Workspace:      0800 to 5800      (About 20K characters)
Symbol Table:   5800 to 6800      (About 4K characters)
```

Actually, SET subtracts 3 from the upper values in the report - so the workspace is from 0800 to 57FF in reality. When parameters are given for SET, you will have to subtract 3 from the value you really want.

The second line of the report says that:

```
Free Area in Workspace:  0800 to 5800
```

If a line of text is entered, and SET applied, the report will change a little:

```
10 THIS IS SOME LINE OR ANOTHER
```

```
SET
```

```
0800-57FC 5800-67FC 0000
081F 5800
```

Each line of text in the workspace takes 2 bytes for the line number, and 1 byte for each character in the line. The Line 10 takes up locations 0800 through 081E. The free area is from 081F to 5800 (The report is too optimistic by one byte here.)

If you want to, try BREAK - and then X from the monitor. Then do a SYS 36998 to the warm start, and MACROTEA will provide you with the same report as above.

You will notice that this report is the same one which MACROTEA gives when it is started by either starting address.

The 0000 on the right edge of the SET report is an artifact from MACROTEA's ancestor, the Relocating Assembler by Carl Moser. It has no meaning for MACROTEA users. (If you must know why, see the DIGRESSION in the section on the .DI, .SE, and .SI pseudo-ops (Page 125) in the Assembler.)

When using SET all five address parameters must be entered.

You will use SET to configure MACROTEA for your memory requirements. Let's see how this works. First, let's make the workspace start at \$0400 (Hexadecimal):

```
SET $0400 $1BFC $1C00 $1FFC 0
0400-1BFC 1C00-1FFC 0000
0400 1C00
```

Now let's set the workspace using decimal SET command:

```
SET 2048 7164 7168 8188 0
0800-1BFC 1C00-1FFC 0000
0800 1C00
```

For the die-hard, % will specify a binary number in SET. Have fun!

PR
//

Note that when you perform a SET, the workspace will be cleared. SET should be performed first when MACROTEA is turned on.

SET will accept parameters and change the memory pointers accordingly. Bewares! SET does not check to see if the parameters make any sense:

SE 321 456 333 122 8888

0141-01C8 014D-007A 22B8
0141 014D

As you can see, the text goes into the sacred first 1K of the PET's memory, and the Symbol Table overlaps the text area. (ugh)

To recover from this awful doom,

SE \$800 \$57FC \$5800 \$67FC 0

0800-57FC 5800 67FC 0000
0800 5800

Note you must take care to respect the 3-byte difference on the upper bounds for the workspace and the symbol table.

DIRE WARNINGS:

DW #1: MACROTEA reserves the top 1K of RAM for its own use. Make sure when you are assembling source that you don't load object code into this area, 'cause if you do, all kinds of unpredictable but awful things could happen! This area is \$7C00-8000 for a 32K PET.

DW #2: Incorrect or unusual values of SET may bomb the system. For example:

SE 0

0000-3FFC 4000-5FFC 0000
0000 4000
10 HELLO

(poof! PET has gone to never-never land.)

DW #3: When MACROTEA performs the PUT, GET, and .CT functions for cassette tape, the PET SAVE and LOAD ROM routines are called. These routines store and replace data from fixed areas of memory, and if you arbitrarily change the SET values, MACROTEA will no longer be able to correctly read files. (This applies only to tape.)

ON USING BASIC WITH MACROTEA

Often there are machine language programs which are intended to be used from a BASIC program, for example, a game program would call some machine language to perform an animation sequence. The default setting of MACROTEA leaves a 1K space for short BASIC programs.

When SET is used, the PET's BASIC pointers will be changed so that the "Top of BASIC's Memory" pointers are at the bottom of the workspace. When you enter BASIC from MACROTEA, the first thing to do is to enter NEW, which will reorganize all of the BASIC pointers to appropriate values.

Coldstart MACROTEA, and then leave via:

```
BR
B*  PC  IRQ  SR  AC  XR  YR  SP
.;  9085 E455 F1 F0 80 50 FF      (BASIC 4.0 display)
.X
READY.
?FRE(0)
1020
```

If you must squeeze out every byte for the MACROTEA workspace, be sure to leave at least 7 bytes in BASIC (and preferably around 100) to let you use loops like:

```
FOR J=1 TO 123:POKE 850,J:SYS851:NEXT
```

to test machine language code. The variable J requires 7 bytes. If you use strings, you will need 7 bytes plus the length of the string.

To summarize: MACROTEA leaves you with a 1K PET as far as BASIC is concerned. If you use SET, it will change the "Top of BASIC" to the lower edge of the workspace.

Preamble for GET and PUT

GET: The GET command will load MACROTEA source files starting at the location specified by the beginning of source as defined with the SET command. The power-on value is \$0800.

PUT: PUT will save the contents of memory between the start and end source pointers as defined with the SET command. The power-on range is \$0800-3FFC .

After executing a GET or PUT statement to the disk, MACROTEA will return with the disk error channel message:

00, OK,00,00

...but this only applies to the disk. When using cassette the only response after the cassette "OK" is the return of the cursor after the process is complete. (You won't see a READY.)

Specifying GET and PUT filenames using MACROTEA is identical to the way you would specify SAVE and LOAD filenames in BASIC.

Disk User's Note

If you have a Commodore Disk system, check the DISK command for how MACROTEA interfaces with the disk. Disk versions of MACROTEA will default to the DISK for most file operations instead of to the tape units. Disk filenames should be in quotes and have the disk drive number included, for example, "0:TESTFILE"

PUT (D1/D2) "filename" (start line#) (end line #)

PUT writes the contents of the workspace to a cassette tape file. The filename may be omitted, or consist of from 1 to 16 characters.

The default Tape Unit is #1. If provided, the parameter D1 will specify Unit #1 and D2 specifies Unit #2.

PUT will write a portion of the workspace if the line numbers are specified. If only one line number is given, PUT writes a one-line file. If two numbers are given, the file will contain the specified portion of the workspace. The part written corresponds with the lines that PRINT would list to the screen - to check PUT, use PRINT.

If you are using the disc version of MACROTEA, be sure to include the D1 or D2 to define a tape file with PUT. (NOTE: You might try it with the disc anyways - this hasn't been checked yet as your humble manual writer only had access to disk MACROTEA for a short time.)

If you have the Commodore Discs, PUT will default to the disc for writing a file. The usual form is PUT "drive:filename", ie, PUT "1:TESTFILE". With the disc system, you must use the D1 or D2 values to write to the tape.

The Commodore Disc system's rules are in effect - for example, if you try to PUT a file that already is on the disc, you will have an error condition.

Examples:

Suppose the workspace contains:

```
PR
0010 THERE ONCE WAS A PRETTY LITTLE
0020 COMPUTER. HER CRT WAS ROSY AND
0030 HER KEYBOARD WAS A LUSCIOUS PEACH
0040 HUE. WHEN SHE RAN HER PROGRAMS,
0050 SHE WOULD USUALLY RUN THEM WITH
0060 GRACE AND CORRECTNESS. AH, BUT IF
0070 SHE SAW A BUG, SHE WOULD SCREAM
0080 AND BE HORRID!
```

To save this on a tape, enter:

```
PUT "CYBERFAIRYTALE"
```

```
PRESS PLAY & RECORD ON TAPE #1
OK
```

```
WRITING CYBERFAIRYTALE
```

The PET acts the same way as with normal BASIC LOAD and SAVE.

If you want to use the 2nd Cassette Unit, specify the Unit number by using the D2 option:

```
PUT D2 "CYBERFAIRYTALE"
```

```
PRESS PLAY & RECORD ON TAPE #2
OK
```

```
(etc)
```

Of course, D1 will specify Tape #1.

To write a portion of the text in the workspace, just add the line numbers which specify the part to be written:

```
PUT "CYBERFAIRYTALE" 40 70
```

```
PRESS PLAY & RECORD ON TAPE #1
(etc)
```

To check this, clear the workspace and use GET:

```
CL
```

```
GET
```

```
PRESS PLAY ON TAPE #1
(etc - see the GET examples)
```

```
PR
0040 HUE. WHEN SHE RAN HAR PROGRAMS,
0050 SHE WOULD USUALLY RUN THEM WITH
0060 GRACE AND CORRECTNESS. AH, BUT IF
0070 SHE SAW A BUG, SHE WOULD SCREAM
//
```

If you care to spend the time, you can check that PUT follows the same rules as PRINT does to select the lines to write to the file.

PUT needs the quotation marks around the filename and if D1 or D2 are used, they must precede the filename.

```
PUT FUBAR
```

```
!09 AT LINE 0000
```

```
PUT "FUBAR" D2
```

```
!09 AT LINE 0000
```

Also, D1 and D2 are the only legal device names.

PUT D3 "FUBAR"

!OA AT LINE 0000

There is no equivalent to VERIFY with MACROTEA. The suggested approach is to follow these rules:

- 1) Be sure your tape unit is correctly maintained. (clean, demagnetized, good tape, etc.)
- 2) Make 2 copies of any important files.
- 3) Dedicate one cassette side per file - don't put two different files on one side except for archival tapes.

Disc Examples:

To save our little tale of the rosy computer, just use the wedge type save command:

```
PUT "0:CYBERFAIRYTALE"
00, 0K, 00, 00
CYBERFAIRYTALE
```

(Naturally you must have 1) a disc in drive number 0 and 2) this disc is initialized and formatted. See the DISK command for how the example diskette was formatted.)

We may check the directory with the DI command:

DI "\$0"

```
0 "SCRATCHDISC-----" SD
2 "CYBERFAIRYTALE" PRG
668 BLOCKS FREE
```

Underline shows reverse field.

If we try to save it again with the same name, the disc runs for a while and then the error LED flashes. The disk error channel returns:
63, FILE EXISTS, 00, 00

The solution is to save the workspace under a different name, such as "MOREFAIRYTALE". You are advised to: 1) PUT the new version, 2) then scratch via DI (if you wish), 3) use DI alone to get a disk error message:

```
PUT "0:MOREFAIRYTALE"      (response to PUT not shown)
DI "S0:CYBERFAIRYTALE"
DI
01, FILES SCRATCHED, 00, 00
```

Don't kill your old file first - or you'll regret it! While we are on the subject, you are advised to make backup copies of any important text files via the disk Duplicate or Copy commands. Remember that some of the Commodore Disk System's commands, like Save, have some nasty bugs in them and a backup may save you a lot of time later. Backups should always be on a different diskette and the diskette put into a safe place. Putting the date or at least a version number in your filenames is also helpful.

GET (D1/D2) "filename"

GET reads a file from the cassette tapes and puts it into the workspace. If a filename is provided, the tape will be searched until a "matching" filename is found. A "match" is defined by the PET operating system as described earlier. If no filename is given, GET will accept any file.

If the D1 or D2 parameter is provided, GET will read from the specified Tape Unit. D1 specifies Unit #1, and D2 Unit # 2.

Caution: GET simply attempts to load the file into the PET's memory. If the SET values were different when the tape file was originally PUT, or the wrong kind of file is read (like machine language, or BASIC programs), GET doesn't care.

If text is present in the workspace, it will be lost. GET clears the workspace and then loads the file.

Be sure that the tape being read was created by the MACROTEA PUT command, and that the memory pointers of SET are the same.

For Commodore Disc users, GET will default to the Commodore Disc System. You should use the form "drive:filename", such as GET "0:TESTFILE" to fetch a file from your discs. Since MACROTEA merely passes the commands on to the disc system, the usual disc rules are in effect.

Disc users must specify D1 or D2 to read a file from the cassette tape units.

Examples:

We assume that you have the tape created in the example for PUT. If not, go make one.

Using GET is simple enough:

```
CL
PR
//
```

```
GET
```

```
PRESS PLAY ON TAPE #1
```

```
OK
```

```
M
```

(no message or prompt except cursor)

```

PR
0010 THERE ONCE WAS A PRETTY LITTLE
0020 COMPUTER. HER CRT WAS ROSY AND
0030 HER KEYBOARD WAS A LUSCIOUS PEACH
0040 HUE. WHEN SHE RAN HER PROGRAMS,
0050 SHE WOULD USUALLY RUN THEM WITH
0060 GRACE AND CORRECTNESS. AH, BUT IF
0070 SHE SAW A BUG, SHE WOULD SCREAM
0080 AND BE HORRID!
//

```

Since no filename was given, GET accepted the first file on the tape. Once loaded, a report on the file is given. The first number is the length of the file in hexadecimal. The other two numbers show the location of the first byte, and the location of the last byte plus one for the text in the workspace.

If text were already in the workspace when GET was used, this text will be destroyed, and the text from the tape file will appear instead. This is easily checked.

According to the PET Operating System rules, subsets of the filename are acceptable. GET "CYBER" or GET "C" will also find and read the file CYBERFAIRYTALE.

The default tape unit is #1. If you want to specify the tape unit, use D1 or D2 before the filename:

```

GET D2 "CYBER"

PRESS PLAY ON TAPE #2
OK
M                                     (cursor is only response at completion)

```

Like PUT, the filename must be in quotation marks and the parameters in the right order.

Disc Examples:

Let's remove our little saga of the pretty computer and see if we can get her back:

```

CLEAR

GET "0:CY"
62, FILE NOT FOUND,00,00

```

Well..... the disc system needs the full filename. Try it again:

```

GE "0:CYBERFAIRYTALE"

00, OK,00,00
CYBERFAIRYTALE

```

MACROTEA prints only the disk error message. If you wish to find the size of the remaining workspace:

```
SE
0800-0901 4000-5FFC 0000
0901 4000
```

Use of PR could verify that the file was correctly loaded into the workspace.

You may, if you wish, specify D8 for the disk - though there's no good reason to:

```
GET D8 "0:CYBERFAIRYTALE"
00, OK,00,00
CYBERFAIRYTALE
(And the same with PUT.)
```

DUPLICATE

DUPLICATE lets you copy from Tape Unit # 1 to Tape Unit # 2. A tape with several files can be copied automatically. (When you are assembling a large program, this facility is helpful in setting up your multi-file tape.)

DUPLICATE operates by reading the first file into the workspace, copying onto Tape #2, reading the second file into the workspace, copying, and so on.

DUPLICATE must be stopped by pressing STOP. To return to MACROTEA, SYS 41104. DUPLICATE will destroy the contents of the workspace and leave it with the last file that was copied.

If you wish to have DUPLICATE stop at the end of a tape, an end-of-tape marker must be written as the last file on the original tape. This is done with the command PUT X. The duplicate tape will not have the end-of-file mark on it.

The disc version of MACROTEA doesn't have the DUPLICATE command. If you enter DU, you end up in the Monitor. Since you can copy files and discs via the DISC command, this isn't much of a loss.

Examples:

Note: To use DUPLICATE, you will need the Commodore Second Cassette Unit for the PET.

CL
GET

PRESS PLAY ON TAPE #1
OK

FOUND CYBERFAIRYTALE
LOADING

This loads our little fairy tale into the workspace. Leaving the tape in the drive, and not rewinding, perform these PUTs:

PUT "FILE ONE"	(No need to "PRESS PLAY & RECORD after
PUT "FILE TWO"	the first one....)
PUT "FILE THREE"	

Now, rewind the tape, and CLEAR the workspace:

CL

DUPLICATE

PRESS PLAY ON TAPE #1
OK

FOUND CYBERFAIRYTALE
LOADING

PRESS PLAY & RECORD ON TAPE #2

Put a blank tape in the second unit and do as directed.

OK

WRITING CYBERFAIRYTALE
FOUND FILE ONE
LOADING
WRITING FILE ONE
FOUND FILE TWO
LOADING
WRITING FILE TWO
FOUND FILE THREE
LOADING
WRITING FILE THREE

(Now that both recorders are correctly set, no more prompts will appear.)

We are done, but DUPLICATE doesn't know that. Press the STOP key when you hear Tape Unit # 1 start turning again. The PET is now in BASIC mode, and to return to MACROTEA, SYS 41104.

A note of caution: DUPLICATE is like GET in that it doesn't care where the tape being read thinks the workspace is. Be sure SET is the same, and that the tape was formed via PUTs.

To stop this nasty behaviour, the original tape must have an end-of-file marker on it. Repeat the process used above to generate the original tape, and then execute:

PUT X

M (the cursor appears. It is assumed that PLAY and RECORD were already pressed.)

When you try DUPLICATE again, it will read and copy the files as before. When the last file is read, the end-of-tape marker will force a return to the MACROTEA Editor and the FILE: heading will be printed.

Two cautions: First, no end-of-tape marker will be written onto the duplicated tape. If you want this, use:

PUT D2 X

Second, the end-of-tape marker used by MACROTEA is not the same as the one used by the PET's BASIC. Attempts to create a data file with your favorite word processor on tape and then to GET this tape in MACROTEA are totally doomed to fail.

DISK (Commodore Disk Command)

DISK implements the "Wedge" (or, as Commodore calls it, the DOS Support) for MACROTEA. Any disc command of the form:

```
PRINT#1,"some command or another"
```

may be executed in MACROTEA via:

```
DI "some command or another"
```

In the same manner as Wedge, DI without a string in quote marks will interrogate the disc drive for an error message and print the four error strings.

The / and ↑ forms of the wedge are not implemented, as MACROTEA does not handle BASIC programs.

Naturally, all the quirks and foibles of the disc system are still there - MACROTEA merely uses DI to pass your commands on to the discs. If you must use the non-PRINT# commands (OPEN CLOSE SAVE VERIFY LOAD), these must be done from BASIC's direct mode as usual.

Small warning: If you leave MACROTEA and enter the PET's BASIC mode, it is very easy to close the disc file inadvertently. (Just as in normal BASIC, editing a line, RUN, NEW, and CLR will remove all variables, including the disc's OPEN specification.) If you then return to MACROTEA via the warm start (SYS 41104), the use of DISK will create a crash!!!! Your options are either to 1) use the MACROTEA cold start (SYS 40960) or 2) OPEN 1,8,15 and then use the MACROTEA warm start.

Though DISK does the Wedge for MACROTEA, if you exit to BASIC mode, the Wedge isn't there.

Examples:

If you don't have the Commodore Discs, these examples won't work.

Let's do a few basic disk utility jobs to get the feel of MACROTEA and the Commodore Disk system. First, get MACROTEA running, and find a blank diskette. Now let's format the diskette:

```
DI "NEW:SCRATCHDISC,SD"
```

```
M
```

While the cursor blinks, the disc drives take about 30 seconds to format the disc. Note we have used the default drive number 0 (zero). To see if there are any errors,

```
DI
00, OK,00,00
```

Nope. Now for a peek at the directory:

DI "\$0"

O "SCRATCHDISC-----" SD
670 BLOCKS FREE.

Underline indicates reverse field.

In normal use, you have to initialize the disk drive each time you place a new disk into the drives. (New in the sense of different). Turn off the disks, turn them back on again, and re-insert the disk we just created. Then:

DI "I0"

Letter I, number zero

Now the PET disks know that the new disk is present.

To check the error reports, try:

DI "I1"

The disk will eventually turn on the error LED, and to see what's wrong, just use DI:

DI
21,READ ERROR,79,02

Be sure there isn't a disk in the Drive #1 for this example.)

See the sections on GET and PUT to see how MACROTEA text files are stored on the disk. Once you have a few small files, see if you can create a backup disk by using the disk DUPLICATE via DI.

As with the Wedge, all DI commands must be surrounded with quotation marks and have one space between DI and the quotation marks.

REPRISE:

The following disk commands are available via the DISK command:

PUT	GET	NEW	INITIALIZE
DIRECTORY	COPY	DUPLICATE	SCRATCH
VALIDATE	RENAME	ERROR REPORT	

THE ASSEMBLY PROCESS

Since MACROTEA is intended for assembly of 6502 code, there have to be commands related to the assembly of the text in the workspace (or from tape files). Many of the facilities described here may be confusing to the beginner - but as time passes, one appreciates their worth.

ASSEMBLE (LIST/NOLIST) (Starting Line Number)

The ASSEMBLE command directs MACROTEA to begin assembly of the source code in the workspace. (For the details concerning the modes of assembly, see the section on the Assembler.)

When ASSEMBLE is used without any parameters, it defaults to the LIST option and will start assembly at Line 0000 in the workspace.

If you specify ASSEMBLE LIST, the second pass of the assembly will provide an assembly listing to the screen. ASSEMBLE NOLIST will not provide an assembly listing. (NOTE: LIST or NOLIST will be overridden by the .LS or .LC pseudo-ops if they appear in the source text.)

When a Starting Line Number is given, the assembly will start at this line instead of from the first line in the workspace.

Examples:

The following program is used as an example for all of the commands in this section. When executed, the program simply prints the PET character set on the top of the screen.

```
FO SE

PR
0010          ;DRAW CHAR SET ON SCREEN
0020          .BA $033A
0030          .OS
0040DRAW      CLD
0050          LDX #$00
0060LOOP      TXA
0070          STA $8000,X
0080          INX
0090          BNE LOOP
0100          RTS
0110          .EN
```

To assemble this program, simply enter:

```
ASSEMBLE
//0000,0345,0345
```

The report at the end of assembly explains that:

```
There are 0000 errors. (This is a decimal number)
The program counter is at $0345 at the end. (Hex number)
The loading counter is at $0345 at the end. (Hex number)
```

For a clearer understanding of these, see the pseudo-ops .CE, .BA, and .MC. Briefly, .CE tells the assembler to continue assembly even if errors are detected. The first number in the report tells how many errors were detected. .BA sets the assembler's program counter. When the assembly is finished, the program counter will be one more than the address of the last assembled byte. The second number indicates this in hexadecimal. .MC tells the assembler to store the code in a location different than .BA is assembling to. The loading counter indicates where the byte following the last byte of code will go.

If you now execute this program by RUN DRAW, the top of the screen will immediately fill with the 256 characters of the PET's screen. This program is a much faster equivalent of the BASIC program:

```
FORJ=0 TO 255: POKE 32768+J,J :NEXT J
```

To see the assembly listing, use:

```
AS LI
```

```

                                0010          ;DRAW CHAR SET ON SCREEN
                                0020          .BA $033A
                                0030          .OS
033A-D8      0040DRAW          CLD
033B-A200    0050          LDX #$00
033D-8A      0060LOOP          TXA
033E-9D0080  0070          STA $8000,X
0341-E8      0080          INX
0342-D0F9    0090          BNE LOOP
0344-60      0100          RTS
                                0110          .EN
LABEL FILE: [ / = EXTERNAL ]

DRAW=033A          LOOP=033D

//0000,0345,0345
```

The 12 columns on the left contain the object listing. First is the program counter's value, then a dash, and then from 1 to 3 bytes of object code, depending on the instruction's length.

In the center are the line numbers from the workspace, and to the right is displayed the source text. The format of the source text part will be controlled by the state of FORMAT (SET or CLEAR). Here the format is SET.

Following the listing, the Symbol Table (Called the LABEL FILE just to be confusing!) with a display of the labels and their values. This program has two labels, DRAW at 033A, and LOOP at 033D.

Finally, the end-of-assembly report is given.

The [/ = EXTERNAL] note in the Symbol Table's title is a remnant from MACROTEA's ancestor, the relocating assembler by Carl Moser. It has no meaning in MACROTEA.

You may start the assembly at a specified line number if you desire:

```
ASSEMBLE LI 40
```

```
9800-D8      0040DRAW      CLD
9801-A200    0050          LDX #$00
.... (etc ) ....
```

Note that the pseudo-ops in Lines 20 and 30 were ignored. The code is also assembled to the default address of \$9800. If you tried AS 30, the code would end up at 9800, and a SYS 826 might result in a crash (If the previous assembly wasn't performed).

When errors are encountered, ASSEMBLE will usually stop the assembly and report the error message - and leave you in the Editor.

```
5 HELLO
AS
          0005          HELLO
!02 AT LINE 0005
```

The .CE pseudo-op will continue assembly in the face of minor errors:

```
1 .CE
AS
          0005          HELLO
!02 AT LINE 0005
          0005          HELLO
!08 AT LINE 0005

!02 AT LINE 0005
//0002,0345,0345
```

Note that the report now indicates two errors.

RUN (Label/Label Expression)

RUN executes the machine language code in memory starting at the address specified by the Label or Label Expression. To return from the machine code, an RTS instruction is used.

Examples:

Again, let's use the example program in ASSEMBLE. First, assemble the program:

```
AS LI
..... (shown earlier) ....
LABEL FILE: [ / = EXTERNAL ]
DRAW=033A          LOOP=033D
//0000,0345,0345
```

Note that the starting point, DRAW, is at address \$033A. We can use RUN to see what happens....

```
RUN DRAW
```

The top of the PET's screen will now be a jumble of the 256 characters used by the PET. If the screen didn't scroll, the top will start with:

```
@ABCDEFGHIJKL .....(etc)
```

RUN will accept any Label Expression - for example, since DRAW is at \$033A,

```
RUN $033A
```

will also work just fine.

Another variation is:

```
RUN LOOP-3
```

or:

```
RUN 826          (Decimal equivalent of $033A)
```

RUN does require a legal value - or you will get an error message:

```
RUN
```

```
!0A AT LINE 033D
```

Here, RUN needed a value to execute from.

RUN POOHBEAR

!08 AT LINE 033D

Here, POOHBEAR is a non-existent label.

Some caution is needed with RUN:

- 1) Be sure the machine code is really there! The default setting for the assembler does not load the object code.
- 2) Label Expressions stop at the first space. RUN LOOP -3 would execute at \$033D, the value of LOOP - not at 033A as desired.
- 3) Be sure your machine code will eventually find an RTS and stop. The PET will not come back from a program if the program doesn't tell it to.

BREAK

BREAK transfers you to the Monitor part of MACROTEA. See the section on the Monitor for the commands used by the Monitor. To return to MACROTEA, use:

G 9086

which will jump to the "Warm Start" address of MACROTEA.

Examples:

Suppose we have the example program mentioned in ASSEMBLE. To check if the code was really present in memory,

```
ASSEMBLE
//0000,0345,0345
BREAK
```

```
B*  PC  IRQ  SR AC XR YR SP
.;  9086 E455 F1 F0 80 50 FF (BASIC 4.0 display)
.   M                        (M indicates the cursor.)
```

The BREAK command jumped to the MACROTEA Monitor. Now for a look at the code:

```
.   M 0338,0345

.:  0338 20 20 D8 A2 00 8A 9D 00
.:  0340 80 E8 D0 F9 60 36 FF 65
.   M
```

The underlined portion is the object code for the program.

To get back to MACROTEA, restart at 9086 (hex).

```
.   G 9086
0800-57FC 5800-67FC 0: 0000
087D 580C
M
```

The address \$9086 is the "Warm Start" address for MACROTEA. A jump to the "Warm Start" turns MACROTEA on without any changes to the workspace or the various commands settings.

If for some reason you suspect that the workspace is mangled, the "Cold Start" for MACROTEA is \$9000. (Same as a SYS 36864.)

PASS

PASS initiates Pass 2 of the assembly process for multifile assemblies. (Ie, those requiring the .CT pseudo-op.) If your source text is entirely in the workspace, MACROTEA will automatically start PASS 2 when it sees the .EN pseudo-op - and PASS is not required.

If your files are assembled from tape, PASS gives you a "breathing spell" in which to rewind the tapes. MACROTEA will read the tapes during Pass 1, and then announce READY FOR PASS 2 and wait. When you have mounted your rewound tapes, enter PASS and the second part of the assembly will proceed.

PASS is limited to tape unit #1.

If you have the disc version of MACROTEA, there is usually no need for PASS. As described with the .EN pseudo-op, .EN "drive:filename" will start Pass 2 with the specified file. (This is usually the first file in the assembly - unless the first file consists entirely of .DI and .DE label definitions.)

Examples:

See the examples for .CT.

SYMBOLS

SYMBOLS will display the contents of the Symbol Table, provided an assembly has been performed.

Examples:

Suppose we have saved the example program on a tape - and then do the following:

(Cold Start MACROTEA via SYS 36864)

GET "EXAMPLE PROGRAM"

PR

(Example Program is listed.)

Now to try SYMBOLS

SYMBOLS

LABEL FILE: [/ = EXTERNAL]

//00FF,9800,9800

Since no assembly was performed, the Symbol Table is empty. SYMBOLS displays the empty table, and then a meaningless Assembly report.

If an assembly is performed,

ASSEMBLE

//0000,0345,0345

SY

LABEL FILE: [/ = EXTERNAL]

DRAW=033A

LOOP=033D

//0000,0345,0345

Now the expected contents of the Symbol Table are present. Note the repetition of the assembly report.

If the assembly had an error, the Symbol Table will be empty. Try adding Line 5 HELLO, assembling, and LABELS - the "null" Symbol Table will be shown.

NOTE:

As MACROTEA starts, the symbol table is printed in two columns to fit nicely on the PET's screen. The symbol table appears after each ASSEMBLE or SYMBOLS command. If you are using HARD, the number of columns can be changed by:

POKE 32257, 18 (two columns)

POKE 32257, 36 (three columns)

If you prefer to use the Monitor, the hexadecimal values are:

Address: \$ 7E01

Two Columns: \$ 12

Three Cols: \$ 24

Assembler in Detail

THE ASSEMBLER

STANDARDS USED HERE

MACROTEA is intended for professionals and experienced amateurs. If you have not used an assembler before, it will be some time before the material in this section will make sense. Learning machine language is not trivial - it takes as much time and effort as learning BASIC. Also, like BASIC, once you grasp how machine language works, learning a new machine language is much simpler than learning your first one.

There are several books available which purport to teach 6502 assembly coding. As each book emphasizes different things, you are advised to get two or three to begin with. Try entering some of the book examples and see if you can get them to work. (Don't be surprised if some of the examples don't work. Each author has his very own, slightly different assembler for the 6502, and some (I won't say who!) examples are just plain wrong!)

Please take a look at the Summaries section on the Assembler before reading this part. Several important details, including the assembly text format, opcodes, addressing modes and default settings are covered in the Summaries and will not be repeated here. This section will refer you to the Summaries as needed.

The standard MOS Technology 6502 Op-Codes are used in MACROTEA. See the "6502 Instruction Set" and related parts in the Summaries section of this Manual.

ADDRESSING MODES

The 6502 addressing modes are many and diverse. In MACROTEA, the form of the operand determines the addressing mode to be used. See the "On Addressing Modes" page in the Summaries for the Assembler.

ASSEMBLER DEFAULTS

As you will have noticed in the Editor, some of the commands merely decide which of several options are to be used. For example, FORMAT is CLEAR unless you tell MACROTEA otherwise. The Assembler has its own host of options, and their initial settings, or "Defaults" are described in the "Assembler Default Settings" part of the Summaries for the Assembler. Take due heed - or nasty surprises are lurking for you!

LABELS

In assemblers, labels play several roles. First, a label indicates an address - and the uses of addresses is a major part of machine language coding. If you are used to BASIC, the versatility of labels & addresses will be a bit confusing.

Some applications of addresses are:

JMP F00	Here, F00 indicates where the 6502 is supposed to
JSR F00	get the next instruction for execution.
BEQ F00	If the branch condition is true, the 6502 will jump to F00.
LDA F00	Here, F00 refers to the location of a specific
STA F00	byte needed for the instruction's operation.
ROR F00	
LDA F00,X	The meaning of F00 grows more abstract. In the first
STA (F00),Y	instruction, F00 indicates the starting point of a table. In the second instruction, the value of the two bytes at F00 and F00+1 are used to tell where the table begins.
LDA #L,F00	This short sequence takes the value F00 and stores
STA BAZ	it as an address in the locations BAZ and BAZ+1. This
LDA #H,F00	kind of manipulation is used quite often, especially
STA BAZ+1	when first "setting up" values for a program to use.

One way to view labels is to treat them like BASIC variables with a few limitations: 1) A label may be given a value only once. (Unless the SET directive is applied.) 2) A label must have an integer value between 0 and 65535. 3) If you refer to a label, it must be defined somewhere, or you will get an error.

A MACROTEA label may have from 1 to 10 characters - the first character should be from A-Z, and the others taken from A-Z and 0-9 (just like BASIC names). If you follow this convention, all will go smoothly. MACROTEA will accept a few other characters in label names, like @ [] ? / . and so forth, but this isn't a good idea. (Your programs will get harder to read, and 10 letter labels permits a lot of variety. If you are used to short 5 or 6 letter label names, please take note.)

MACROTEA labels must begin in the first column of the text line immediately after the line number. If you don't do this, MACROTEA tries to see the label as an opcode or pseudo-op and will get confused. This requirement does not make for pretty assembly listings, but you can get used to it. A quite legible listing can be made if you use MANUSCRIPT SET and FORMAT SET and then PRINT.

For examples illustrating labels, see the Examples section following "Expressions" (which comes up next.)

EXPRESSIONS

In many cases, some simple computations must be performed before you know the value for an operand. For example, suppose a table of bytes starts at BARSOOM, and you want to use the third entry in the table. It is simpler and clearer to use BARSOOM+2 to indicate this value instead of a set of labels BARSOOM, BARSOOM1, BARSOOM2, etc.

Though the use of expression in MACROTEA is a great convenience, bear in mind that these are used to assist in assembly, and once your program is assembled, their values may not change.

MACROTEA supports these features in expressions:

- | | | |
|----|---------------|--|
| 1) | (label) | The value of the label is used. |
| 2) | + - | Addition and subtraction. |
| 3) | nn, \$nn, %nn | Decimal, Hexadecimal and Binary numbers. |
| 4) | = | Value of the program counter. |

Labels: Any legal MACROTEA label may be used in an expression. Since MACROTEA is a two-pass assembler, the definition of a label's value can appear after the label's use in an expression.

It is possible with the .DI pseudo-op to define labels with ambiguous or meaningless values - however, MACROTEA will cheerfully assign values to these (but not the ones you intended) and not tell you of your errors. The moral is: Labels which are created via .DI or .DE shouldn't be used before they are defined.

Arithmetic: The only arithmetic operations are +, addition, and -, subtraction. (I would like to see * / AND OR NOT XOR at least.) The values are computed by scanning from left to right - just like a simple four-banger calculator. All arithmetic is computed modulus 2^{16} (zero to 65535) as integers. This means that there are no negative numbers in MACROTEA, so the expression 1-2 yields FFFF. (Bewares!)

Take care not to put spaces in an expression - if MACROTEA sees a space, it assumes the expression is finished, and the remaining characters in the line will be seen as a comment.

Numbers: MACROTEA assumes a number is in decimal unless you provide the prefix \$ or %. The \$ prefix indicates the number is in hexadecimal (0-9 or A-F) and the % prefix indicates a binary number (0 or 1).

If the number provided is too large (more than 65535 or FFFF), MACROTEA will take the rightmost digits to compute the number. If the number is decimal, a further modulus 2^{16} will be performed to numbers from 65536 to 99999.

Program Counter: As MACROTEA assembles the source text, it keeps a counter which is used to define addresses and in most cases to store the assembled bytes of code into memory. This is called the program counter, and:

THE VALUE OF THE PROGRAM COUNTER IS ALWAYS THE ADDRESS OF THE OP-CODE BEING PRESENTLY ASSEMBLED FOR INSTRUCTION OPERANDS, AND THE ADDRESS OF THE NEXT BYTE TO BE ASSEMBLED FOR PSEUDO-OP OPERANDS.

A bit of thought will tell you that this is the only reasonable way to do this.

If we have:

```
9800-AD0002 0010      LDA =
9803-03      0020      .BY =
```

In line 10, the program counter is the address of the op-code, AD, for the LDA. The operand of the LDA evaluates to \$200. When you are setting up short branches, like BEQ F00+3, you must count from the first byte of the BEQ.

In line 20, the program counter is the next address to be assembled, which is in this case 203.

Note: MACROTEA does not see these features as expressions:

LDA #H, (expression)	- the (expression) part will be
LDA #L, (expression)	computed as usual.
LDA #'Q	- ASCII values won't work in expressions.

but you should be aware of them. (See Addressing Modes.)

Examples for Labels & Expressions:

It isn't possible to explore all possible combinations of Labels, Numbers, etc in the space available here. Have a look at these examples, and you will see how it goes together.

Suppose we have this program fragment:

```
PR
0010 LDA 10+20+2
0020 FUBAR INX
0030 INC $FFEE
0040 BNE FUBAR
0050 .EN
```

Now to assemble it:

```

AS LI
9800-AD2000 0010      LDA 10+20+2
9803-E8      0020FUBAR    INX
9804-EEEEFF 0030      INC $FFEE
9807-DOFA    0040      BNE FUBAR
              0050      .EN
LABEL FILE: [ / = EXTERNAL ]

FUBAR=9803
//0000,9809,9809

```

If we get out our Programmer's Magnifying Glass & Nit-Picker, a few things can be gleaned from this mess.

A look at line 10 reveals that the operand of the LDA is \$0020, which happens to be 32 or 10+20+2 in decimal.

Line 30 reveals that the hexadecimal number \$FFEE was correctly entered as the operand of the INC instruction.

Line 40 shows the branch to FUBAR is assembled as a FA, or -6 in decimal. Since the 6502 adds the branch value to the address of the byte following the branch instruction, this value is also correct. (Try it by hand for a while!)

The symbol table displays FUBAR as 0203, as is doubtless true.

Now let's try a few more features:

```

AS LI
              0005 .BA $0200
              0010TOM .DI DICK+5
0200-AD1004 0020 LDA TOM+TOM
0203-E8      0030DICK INX
0204-290F    0040 AND #%00001111
0206-A948    0050 LDA #'H
0208-8D1802 0060 STA HARRY
020B-A908    0070 LDA #L,TOM
020D-8D1902 0080 STA HARRY+1
0210-A902    0090 LDA #H,TOM
0212-8D1A02 0100 STA HARRY+2
0215-6C1902 0110 JMP (HARRY+1)
0218-        0120HARRY .DS 3
              0130 .EN
LABEL FILE: [ / = EXTERNAL ]

TOM=0208      DICK=0203
HARRY=0218

//0000,021B,021B

```

If you read this program, the program will run, and not do very much (besides crash your PET). However, there are several expressions worth note.

Line 10 uses the .DI pseudo-op to set the value of TOM. 0410 turns out to be twice 208 in hexadecimal, so the expression TOM+TOM works out correctly in line 20.

Line 40 shows the use of a binary number as an AND mask. This is more readable than \$0F or 15 would be.

Line 50 illustrates the use of ASCII data - "H" is 48 in hexadecimal.

Lines 70 and 90 show the use of the #H and #L addressing modes to simplify loading the address TOM into the locations HARRY+1 and HARRY+2.

Note that the .DS pseudo-op generates a "hole" in the assembly listing.

Exercise:

Using the short program:

```
10 LABEL .DI 100
20 .EN
```

By changing the "100" to the right of the .DI, check out how the MACROTEA expressions are calculated. For example, what is the result of .DI \$FFFF+123? Take some care to investigate the 2^{16} numerical limits and decimal numbers in the range 65536 to 99999.

PSEUDO-OPS AND DIRECTIVES

An assembler which only handled labels and the 6502 op-codes wouldn't be very useful. For example, how would you tell it to start the assembly at an address other than zero?

MACROTEA handles this by providing pseudo-ops and directives. A pseudo-op is a "fake" op-code used to control various "housekeeping" tasks - such as to list or not list the assembly, deciding where the assembly should start, and so on. In MACROTEA, pseudo-ops always begin with a period and are placed in the Op-Code field. A directive is used to decide whether to assemble blocks of source text or not. In MACROTEA, directives are mostly confined to the conditional assembly instructions. Directives are not preceded by a period.

PRINTER CONTROL

<u>.LS</u>	List Source
<u>.LC</u>	Don't List Source
<u>.EJ</u>	Send Form-Feed to Printer if HARD SET.

The MACROTEA assembler normally takes two passes. In Pass 1, the source text is scanned for label values, and when complete, the symbol table is filled with the labels and their values. In Pass 2, the source text is scanned again, and the 6502 machine language code is generated.

If the .LS is in effect, PASS 2 prints the assembled text and the source text on the screen, which provides you with the assembly listing. When the .LC is in effect, the listing is not generated.

By combining .LS and .LC, you can select the parts of the assembly that you want listed. This is quite handy with long programs.

The .EJ pseudo-op sends a form-feed - which is ignored by the PET's screen, but ejects a page on the printer if HARD is SET. .EJ may be used to make your listing (in hard copy) simpler to read; for example you can put subroutines on separate pages.

As you might expect, .LS and .LC will interact with the ASSEMBLE LIST and NOLIST options. The simplest way to view this is that a flag exists which tells the assembler whether to list at Pass 2 or not. During Pass 1, the .LS and .LC set and reset this flag - with the last one seen determining the final value of the flag. On Pass 2, the .LS and .LC again affect the flag, and the listing is provided or not.

This results in:

If you have no .LS or .LC, AS LI and AS NO will perform as expected.

If you have .LS or .LC, the last one will determine whether the start of the assembly will be listed. This overrides the options for ASSEMBLE.

Examples:

Let's start with this simple non-program:

```

PR
0010          ;ONCE UPON
0020          ;A TIME
0030          ;THERE WAS A
0040          ;BASHFUL
0050          ;PROGRAM
0060          .EN
    
```

You can check that AS LI and AS or AS NO work as expected. Now let's try out the .LS pseudo-op:

35 .LS

AS

```

0010      ;ONCE UPON
0020      ;A TIME
0030      ;THERE WAS A
0035      .LS
0040      ;BASHFUL
0050      ;PROGRAM.
0060      .EN

```

(We will skip the symbol table & other junk.)

AS NO

(Same thing.)

You would expect the listing to begin at Line 35 or 40, and not to see the whole thing! During Pass 1, MACROTEA saw the .LS and set the flag to "list". This overrode the "nolist" set by the ASSEMBLE command. During Pass 2, the flag was checked and found to be in the "list" state, and so the whole program was listed.

Now to try the .LC pseudo-op:

55 .LC

AS

```

0035      .LS
0040      ;BASHFUL
0050      ;PROGRAM

```

Here, Pass 1 saw the .LC last, and didn't start listing until the .LS in Line 35. Note that the .LC and .LS start with their own lines, and not the immediately following line.

You can verify that AS LI will give the same result.

I won't provide an example of .EJ for it would take a lot of room & blank space. You can try:

45 .EJ

and see the result. Note that the form-feed appears on the PET screen. MACROTEA is conservative and sends linefeeds (actually Cr/Lf) to the printer, so the screen is affected too.

ASSEMBLY CONTROL

.EN End of source text.

.CE Continue in the face of errors.

.CT "filename" Continue source from tape.

"drive:filename" Continue from disc.

The .EN pseudo-op tells MACROTEA to stop examining the source text in Pass 1 and to begin Pass 2 of the assembly. MACROTEA is a bit literal-minded and requires the .EN at the end of the text that is to be assembled. In the same spirit, you may not include .EN within a conditional section of code. (See the Conditional Assembly Directives.)

When an error is seen by MACROTEA, the assembly usually is stopped and the error number is printed. When you are first attempting to assemble a program, there will normally be a number of errors. The .CE pseudo-op tells MACROTEA to continue the assembly if at all possible and to report the errors as they are found.

Some errors are too serious to permit MACROTEA to continue the assembly. These are:

!04 .BA or .MC Operand isn't defined.

!07 .EN missing at end.

!17 Bad Tape Load. (Checksum in Error)

One of the nicest features of MACROTEA is the .CT pseudo-op. .CT tells MACROTEA to continue the assembly by loading a new source file into the workspace from tape or disc. This can be repeated as many times as required. The use of .CT permits assembly of programs whose source is much larger than the workspace. (In most assemblers, and MACROTEA is no exception, the source text is much longer than the final machine language program. The typical ratio is around 16-20 characters of source per byte of final machine language.)

If desired, a filename can be specified after the .CT. The PET searches for the file according to the usual rules. Each instance of .CT clears the workspace, loads the file, and performs Pass 1 or Pass 2 of the assembly. When .EN is finally encountered, MACROTEA will announce that it is ready for PASS 2 and will wait for you to rewind your tape(s). The PASS command will then execute PASS 2.

When MACROTEA links files, it ignores the first line of every linked file. To avoid this problem, all of your programs should contain a ";" in the first line:

```
0005      ;
0010      LDA #$AA
0020      (etc.)
```

Some caution must be exercised with .CT to ensure that the workspace has the correct contents when Pass 2 is started. A simple method is to carefully follow these steps:

1. CLEAR the workspace.
2. Enter the one line program: 10 .CT ""
3. ASSEMBLE
4. When MACROTEA states that it is ready for Pass 2,
5. CLEAR the workspace.
6. Enter the one liner again: 10 .CT ""
7. Now execute the PASS command.

Failure to do this will cause MACROTEA to assemble the last file in the series, as this file and its associated .EN will be in the workspace at Step 4.

The use of .CT also demands that:

A. - The sequence of files to be assembled should be placed on one physical tape. MACROTEA won't stop between files to let you physically mount different tape cassettes.

B. - MACROTEA tends to remember the last filename that it loaded, and if you use .CT without the "" filename, MACROTEA will search for the remembered name instead of the null filename.

(NOTE: In some cases you can get around these rules - my concern here is to ensure that your multifile assemblies work properly and is not to look at all of the exceptions.)

C. - If you are using macros, each new file must have all of its macros defined in the file. MACROTEA looks at the workspace to find the expansion for a macro, and each new file will obliterate any macro definitions in the previous file. This permits you to redefine macros under the same name from file to file, but this isn't advised to make debugging easier.

For the PET Disc owners, MACROTEA uses .CT in a slightly different manner. Each file in a series to be assembled together must end with .CT "drive:nextfilename". Each file's .CT must specify the name of the next file to be assembled. Note that this requires that all of the files to be assembled together must reside on one or two diskettes.

The last file in a series to be assembled via .CT on the disk must end with the .EN "drive:firstfilename". The filename after .EN must be the first file in the series to be assembled. If this is not provided, MACROTEA tries Pass 2 with the file in the workspace (the last file of the series) which won't work too well!

Examples:

You will quickly come to appreciate the requirement for .EN for MACROTEA won't assemble your programs without it. (Not that you will ever like this!)

Suppose we have:

```
PR
0010          LDA #00
0020          STA 0
0030          RTS
```

Now to try and assemble this thing:

```
AS NOLIST
          0030          RTS
!07 AT LINE 0030
```

Oops We need the .EN:

```
40 .EN

AS NO
//0000,9806,9806
```

Now it works.

To see that .EN is really serious business, let's add a few lines after the .EN:

```
50 CMP #11
60 BNE 123
```

And try it again - this time with a listing:

```
AS LI

9800-A900 0010          LDA #00
9802-8D0000 0020          STA 0
9805-60 0030          RTS
          0040          .EN
LABEL FILE: [ / = EXTERNAL ]

//0000,9806,9806
```

As you can see, the .EN was really the end, and Lines 50 and 60 were never seen by MACROTEA.

The .CE pseudo-op is illustrated with this inaccurate program:

```
PR
0010          LDA #5*5
0020          STA MONKEYS
0030          ROR X
0040          .EN
```

The usual assembly gives:

```
AS
      0010      LDA #5*5
!0A AT LINE 0010
```

The errors in lines 20 and 30 aren't seen for MACROTEA never gets there. Here is the fix:

```
5 .CE
AS
      0010      LDA #5*5
!0A AT LINE 0010
      0010      LDA #5*5
!0A AT LINE 0010
      0020      STA MONKEYS
!08 AT LINE 0020
      0030      ROR X
!08 AT LINE 0030
//0003,9808,9808
```

If you don't know what the errors are, go look them up! Note that the assembly report has a count of the number of bad lines (0003) in its first part.

If you make a listing, the errors will appear mixed in with the listed lines. When the assembled operands are examined, note that MACROTEA tries to fill them in with "reasonable" values. The LDA ends up as A9 05, and the STA and ROR address location zero.

MACROTEA will stop for really serious errors:

```
25 .BA MOTHER
AS
      0010      LDA #5*5
!0A AT LINE 0010
      0025      .BA MOTHER
!04 AT LINE 0025
```

So, you just gotta fix these ones first.....

Once you are error-free (from the viewpoint of MACROTEA), the interesting and challenging task of a multi-file assembly is presented to pique your imagination.

To illustrate .CT, enter and PUT these two files onto one tape:

```
CL
10BEES LDA 00
20 LDA 11
30 JMP HONEY
40 .CT ""
```

PUT "ONE"

(etc)

CL

10HONEY STA 22

20 STA 33

30 JMP BEES

40 .EN

PUT "TWO"

(etc)

Note that each file cannot be assembled by itself due to the labels BEES and HONEY.

Now rewind the tape, and follow this sequence:

CL

10 .CT ""

AS LI

PRESS PLAY ON TAPE #1

OK

(do so)

READY FOR PASS 2

(only with linking assembler and cassette when there are no more .CT files)

M

(cursor is only indicator of completion with cassette)

Now rewind the tape, and continue with PASS:

CL

(If you forget this step, MACROTEA will continue with the file TWO which happens to be in the workspace until you CLEAR it.)

10 .CT ""

PASS

0010

.CT ""

PRESS PLAY ON TAPE #1

OK

(be obliging)

```

9800-AD0000 0010BEES      LDA 00
9803-AD0B00 0020          LDA 11
9806-4C0998 0030          JMP HONEY
                   0040      .CT ""

```

```

9809-8D1600 0010HONEY     STA 22
980C-8D2100 0020          STA 33
980F-4C0098 0030          JMP BEES
                   0040      .EN
LABEL FILE: [ / = EXTERNAL ]

```

```

BEES=9800              HONEY=9809

```

```
//0000,9812,9812
```

You may of course put more than two files on a tape to be joined together via the .CT pseudo-op.

To instill a sense of caution with .CT, you are advised to try these mistakes out:

ERROR ONE: Rewind the tape, and again start the assembly. When you arrive at READY FOR PASS 2, don't clear the workspace. Enjoy the spectacle of MACROTEA's assembling the file TWO only!

ERROR TWO: Using a scratch tape, CLEAR the workspace, enter the 10 .CT line (without the two quote marks), and then PUT "TURKEY". When finished, put the first tape back into the recorder and try an assembly. MACROTEA will print the word TURKEY after the OK message, and will then search for TURKEY (and ignore your file ONE). The .CT by itself does not update the filename stored by MACROTEA. (A GET "ONE" would leave ONE in the filename area, etc...) The moral is to always specify a filename, even if it is the humble "".)

ERROR THREE: Make two files; in the first one define the macro FOOH and in the second file, place a call to FOOH. When this is assembled, there will be an error in the second file which tells you that FOOH isn't a legal op-code. Macro definitions must be repeated for each file that uses their macro calls.

Disk Examples:

To show .CT on disk, first create & save these two files:

```
GET "TAKE ONE"
00, OK,00,00
TAKE ONE
PRINT
0010TOFU      LDA #$AA
0020          LDA #$BB
0030          LDA #$CC
0040          .CT "O:TAKE TWO"
```

```
GET "TAKE TWO"
00, OK,00,00
TAKE TWO
PRINT
0010BEANS     LDA #$DD
0020          LDA #$EE
0030          LDA #$FF
0040          .EN "O:TAKE ONE"
```

To make this operate, simply GET "TAKE ONE" and then assemble:

```
GE "TAKE ONE"
00,OK,00,00
TAKE ONE
```

AS LI

```
00, OK,00,00
O:TAKE TWO
```

```
00, OK,00,00
O:TAKE ONE
```

```
7F83-A9AA  0010TOFU      LDA #$AA
7F85-A9BB  0020          LDA #$BB
7F87-A9CC  0030          LDA #$CC
```

```
00, OK,00,00
O:TAKE TWO
```

```
0040          .CT "O:TAKE TWO"
7F89-A9DD  0010BEANS     LDA #$DD
7F8B-A9EE  0020          LDA #$EE
7F8D-A9FF  0030          LDA #$FF
0040          .EN "O:TAKE ONE"
```

```
LABEL FILE: [ / = EXTERNAL ]
TOFU=7F83    BEANS=7F89
//0000,7F8F,7F8F
```

Pass 1 looked at both files, and then Pass 2 looked at them again - this time printing the assembly listing.

If you PRINT the workspace, the last module, TAKE TWO is present. Be sure to have the first module in the workspace when you start a multifile assembly!

(If you try assembling TAKE TWO by itself, it will happily assemble starting at \$9800. Since there was no .CT, the filename after the .EN was ignored and a single workspace assembly results.)

Do take some care with disk multifile assemblies. Here is one kind of disaster:

```
GET "TAKE ONE"
00, 0K,00,00
TAKE ONE
PR
0010 LDA #$AA
0020 LDA #$BB
0030 LDA #$CC
0040 .CT "O:TAKE ONE"
```

As far as I know, this will run until the disk drive fails.... Of course you will usually be stopped with a duplicate label error....

As with the tape version, .CT replaces the workspace with the contents of each file as the assembly proceeds. The cautions re macros still apply to disk assemblies using .CT.

LOCATION OF CODE

<u>.BA</u>	Begin Assembly at the address specified in the operand.
<u>.OS</u>	During Pass 2, load the machine language result into memory.
<u>.OC</u>	On Pass 2, don't load the machine language code into memory.
<u>.MC</u>	Load the machine language code starting at the operand, which may differ from the address specified in .BA. (Offset loading)

The object of an assembler, of course, is to produce the machine language code which was specified by the mnemonics, etc of the source text. The pseudo-ops included here are used to tell MACROTEA where, how, and whether the machine code is to be placed in the PET's memory.

The .BA pseudo-op instructs MACROTEA to set the program counter to the value of .BA's operand. This permits the code to be assembled with an arbitrary starting address. For example, .BA \$400+2 will tell MACROTEA to assemble starting at \$402.

.BA may be used as often as needed to build blocks of code arranged in any place and order in memory. If there is no .BA in the source text, MACROTEA assumes the value \$9800, which will place the code in the user's RAM at \$9800-9E7F.

Normally, MACROTEA will not put the assembled code into the PET's memory - a nice touch, for most programs start their life with many errors. When it comes time to test the code, the .OS pseudo-op will tell MACROTEA to place the subsequent code into the PET's memory.

The .OC pseudo-op will tell MACROTEA to not load the following code into memory. Any number of .OS and .OC pseudo-ops may be placed in the source text to load selected parts of the program.

The .MC pseudo-op tells MACROTEA to load the code into the memory starting at the address provided in the operand. This permits code to be placed in a location different from the address it is intended to run at.

Some care must be used with .MC. When MACROTEA sees the .BA pseudo-op, the loading address is reset to the .BA's operand - in other words, .BA cancels any prior .MC's.

You may use .MC as you desire to place code in different places. Each .MC will reset the loading counter and the following lines of code will be placed accordingly.

One way to visualize these pseudo-ops is via three values somewhere within MACROTEA:

- 1) A program counter, used to decide the values of operands and labels.
- 2) A loading counter, used to place the assembled code into the PET's memory.
- 3) A loading flag, which if TRUE, permits the loading of assembled code into the memory.

Then we have:

.BA sets both the program counter and the loading counter to the value of .BA's operand.

.MC sets the loading counter to .MC's operand.

.OS makes the loading flag TRUE.

.OC makes the loading flag FALSE.

Examples:

Some of these examples will require the use of the Monitor. See the Monitor section for explanations of the Monitor's commands.

Let's start with this program:

AS LI

9800-A901	0020	LDA #01
9802-A902	0040	LDA #02
9804-A903	0060	LDA #03
	0100	.EN

(We will skip the Symbol Table & Assembly reports)

Note that the assembly's program counter started at the default value of \$9800.

To illustrate what .BA does, let's add a line,

10 .BA \$300

and assemble again:

AS LI

	0010	.BA \$300
0300-A901	0020	LDA #01
0302-A902	0040	LDA #02
0304-A903	0060	LDA #03
	0100	.EN

Here the assembly started at \$0300. If you wish to add a label or two, it is simple to see the values of the labels change.

.BA may be used more than once - try this one:

```
10 .BA $0350
30 .BA $0250
50 .BA $0300
```

AS LI

	0010	.BA \$0350
0350-A901	0020	LDA #01
	0030	.BA \$0250
0250-A902	0040	LDA #02
	0050	.BA \$0300
0300-A903	0060	LDA #03
	0100	.EN

And here you can see that MACROTEA follows the instructions of .BA exactly, and will assemble code to live anywhere. Be careful of one thing (not usually a problem for PET owners):

```
10 .BA $FFFF
20 LDA #01
30 LDA #02
40 .EN
```

AS LI

	0010	.BA \$FFFF
FFFF-A901	0020	LDA #01
0001-A902	0030	LDA #02
	0040	.EN

MACROTEA will wrap around if any of its values exceeds \$FFFF - this includes operands, program counters, and loading counters. Bewares of this if you intend to down-load from the PET to a KIM or other 6502 system....

To observe the action of .OS and .OC, we need to prepare some memory - so get into the Monitor via BREAK and set up the pattern below:

```
. M 0300-0320

.: 0300 20 20 20 20 20 20 20 20
.: 0308 20 20 20 20 20 20 20 20
.: 0310 20 20 20 20 20 20 20 20
.: 0318 20 20 20 20 20 20 20 20
.: 0320 20 20 20 20 20 20 20 20
```

This pattern will already be in place if you haven't used the first tape unit for file operations. (We are using part of the First Cassette Buffer for these examples.)

Using X and SYS 41104, return to MACROTEA's Editor and enter this short program:

```
10 .BA $300
20 LDA $11
30 LDA $22
40 .EN
```

When you assemble this, (Do this one yourself!) MACROTEA will produce the following bytes of code starting at \$0300: AD 11 00 AD 22 00. When we use the Monitor, however, the code at \$0300 has not changed from the pattern above.

To actually load the code, the .OS pseudo-op must be used:

```
15 .OS
```

After assembly, we go back to the Monitor and here's what you will see:

```
. M 0300-0310
.: 0300 AD 11 00 AD 22 00 20 20
.: 0308 20 20 20 20 20 20 20
.: 0310 20 20 20 20 20 20 20
```

Now, change the pattern back to the 20's in the 0300 line, go back to the Editor, and;

```
15 (delete Line 15)
25 .OS
```

This moved the .OS to a position between the two LDA's. Assemble, and the Monitor will reveal:

```
. M 0300-0307
.: 0300 20 20 20 AD 22 00 20 20
```

MACROTEA has loaded the second LDA but not the first one. Remember that an assembly assumes that the code is not to be loaded into memory - so MACROTEA starts loading the code when the .OS was seen.

Again, clear the memory to the 20's pattern - and in the Editor, change our program to:

```
10 .BA $300
15 .OS
20 LDA $11
25 .OC
30 LDA $22
40 .EN
```

Back at the Monitor, the 0300 line is now:

```
.: 0300 AD 11 00 20 20 20 20 20
```

When MACROTEA saw the .OC in line 25, the loading was suspended. You can easily verify that other combinations of .OS and .OC will work as expected.

At the risk of being tedious, let's work this exercise yet again to illustrate the operation of .MC, which permits the loading of code into locations the code isn't assembled for.

The example program is:

```
10 .BA $300
20FOO JMP FOO
30BAZ JMP BAZ
40 .EN
```

This will generate the machine language sequence: 4C 00 03 4C 03 03. Now, go to the Monitor and make sure that \$0300-\$0320 are set to 20's. To illustrate the action of .MC, add these lines:

```
15 .OS
16 .MC $0310
```

Now assemble and take a look with the Monitor. (NOTE: In the assembly report you will see

```
//0000,0306,0316
```

which shows that the loading counter is 10 more than the program counter.)

The Monitor reveals:

```
. M 0300-0320

.: 0300 20 20 20 20 20 20 20 20
.: 0308 20 20 20 20 20 20 20 20
.: 0310 4C 00 03 4C 03 03 20 20
.: 0318 20 20 20 20 20 20 20 20
.: 0320 20 20 20 20 20 20 20 20
```

If you examine the code carefully, the JMP at 0310 is to 00 03, or \$0300 as we humans view things. The code has been assembled to start at \$0300, and loaded starting at \$0310.

Using MACROTEA, you can try the following short programs and see if they do what you expect them to. Remember the rules stated just before this set of examples.

1) 10 .OS	2) 10 .OS
20 .MC \$0300	20 .BA \$0300
30 .BA \$0350	30 LDA #A1
40 LDA #FF	40 .MC \$0350
50 .EN	50 LDA #A2
	60 .EN

HOUSEKEEPING

<u>.BY</u>	Store bytes of data.
<u>.DS</u>	Define storage space.
<u>.SI</u>	Store address. (Internal Address)
<u>.DI</u>	Define label value. (Internal Address)
<u>SET</u>	Redefine a label value. (NOTE: This is a directive and not a pseudo-op)

The pseudo-ops described here are used to simplify the definition and use of storage areas and "canned" data by the program being assembled. Also included are some pseudo-ops and one directive used for definition and storage of address values.

NOTE: MACROTEA's ancestor, the Macro Assembler by Carl Moser, had some facilities for the relocation of code. This resulted in two varieties of labels and addresses. "Internal" meant that the relocater could change these values when the code was moved. "External" values were not to be changed. Relocation is not provided in MACROTEA. If you must move code, use .BA, .MC and .OS to generate the code in a "safe" area, and then use the Monitor's T command to move it. Most situations that imply relocation are just as easily dealt with by reassembling.

DIGRESSION: The evolution of ROM is the primary reason that relocation is obsolete. In an all-RAM system (as the computers of yore usually were), you must first load the assembler, then load the source text, and finally assemble. The result was usually a paper tape of the object code. To debug this, you had to re-boot the machine and load the object tape. Then came patching, which involved twiddling the front panel instead of using a Monitor. Since re-assembly meant first loading the text editor, loading the source, punching a new source, loading the assembler, reading the source and finally making a new object tape, it was far simpler to load via a relocating loader and then to make the patches. Isn't nostalgia neat?

Here are some brief examples to refer to while reading the following descriptions:

```
.BY 100 $100 %1101 'HO HUM'
.DS 25
.SE FUBAR
.SI FUBAR+2
TAG .DI $2345
TAG2 .DI TAG+5
SET TAG=$5432
```

The .BY pseudo-op places defined data values into memory. After the .BY one or more values may be placed. Each value may be one of:

Decimal	for example,	100 101 102
Hexadecimal	for example,	\$FF \$FE \$FD
Binary	for example,	%0001 %0010
ASCII	for example,	'HELLO THERE'

For ASCII values, the single quote (or apostrophe) must surround the string to be stored. This facility is intended for the normal ASCII characters, and though most PET graphics characters can be included in the string, not all will be correctly entered by the Editor.

Since this pseudo-op defines bytes of memory, only the 8 least significant bits of a value will be stored. (More concisely, the value modulus 256 will be stored.)

The .BY pseudo-op will accept expressions and use their 8 least significant bits for the stored value. Perhaps there is a use for this?

Each value must be separated from the others with a space, and at least one value must be present after the .BY pseudo-op.

The .DS pseudo-op defines a block of memory for the program's use. The value of the operand after the .DS indicates how many bytes are reserved. In most cases, the line containing .DS will be labeled. If the .OS pseudo-op is in effect, MACROTEA simply skips the required number of bytes, and no initialization is performed. Always write before you read any locations in a .DS block.

The operand of .DS may be any legal MACROTEA expression.

The .SI pseudo-ops will evaluate the operand and store the result in the Low-High form required by the 6502 for addresses. The value will be placed in the next two bytes of memory. Nearly all instances of .SE and .SI will be in the zero page.

It is often necessary to assign an arbitrary value to a label - for example, the PET's display at \$8000 might be labeled SCREEN. The .DI pseudo-op assigns the value of the operand to the label. .DI must be on a labeled line in the source text, for it is this label which is given the operand's value.

The SET directive permits the reassignment of a label's value. The new value for the label will be used in all lines following the SET directive. The label must be defined prior to the SET directive.

Examples:

To illustrate the .BY pseudo-op, here is a small program:

```
10 .BA $1001
20 START .BY 100
30 .BY $FF
40 .BY %10101010
50 .BY 'SOMETHING NEW HERE'
60 .BY START-4090
70 .EN
```

AS LI

	0010	.BA \$1001
1001-64	0020	START .BY 100
1002-FF	0030	.BY \$FF
1003-AA	0040	.BY %10101010
1004-534F4D	0050	.BY 'SOMETHING NEW HERE'
1007-455448		
100A-494E47		
100D-204E45		
1010-572048		
1013-455245		
1016-07	0060	.BY START-4090
	0070	.EN

By careful examination you can see that the values 100, \$FF and %10101010 were correctly computed and stored in locations \$1001 to \$1003. More detailed examination shows the storage of the string in \$1004 through \$1014. MACROTEA will assemble strings in 3 byte blocks for as long as required.

Line 60 contains an expression whose value is 7 (\$1001 is 4097 in decimal). MACROTEA successfully evaluated it and stored the result. (NOTE: Though MACROTEA didn't place the results in memory as .OS wasn't in effect, it is the ultimate result that is of interest here and a lot simpler to write 'store'.)

Values may be combined in one .BA for convenience:

AS LI

9800-65FF0E	0010	.BY 101 \$FF %1110 'HI'
9803-4849	0020	.EN

And at least one value is required:

10 .BY		
AS		
	0010	.BY
!0A AT LINE	0010	

To illustrate .DS, first go into the monitor and set \$0300 to \$0320 with the value \$20 (See the examples on .LC and .LS .) Then enter and assemble this program:

```
10 .BA $300
20 .OS
30 LDA $FFFF
40 .DS 10
50 LDA $EEEE
60 .EN
```

If you look at the assembly listing, there is a gap between \$303 and \$30D at line 40. MACROTEA advanced the program counter by 10 at this point. We will soon see that the loading counter was also advanced.

Now, go into the monitor and examine \$0300 to \$0310:

```
. M 0300-0310

.: 0300 AD FF FF 20 20 20 20 20
.: 0308 20 20 20 20 20 20 AD EE EE
.: 0310 20 20 20 20 20 20 20 20
```

If you count the number of 20's after the AD FF FF, there are ten of them, as specified by the .DS operand in line 40. If you use the Monitor to change these bytes to some other value & re-assemble, the new values will remain unchanged.

The moral of this one is to understand that:

THE SPACE CREATED BY .DS WILL USUALLY BE FULL OF GARBAGE!

So, be sure your program writes to these locations before reading them.

.DS accepts expressions, and you can easily check this by making these changes & reassembling, Monitoring, etc.

```
15FX .DI 5
40 .DS FX+FX+FX
```

When the occasion arises to store addresses as data, .SE and .SI come to the rescue. Here is an assembled example:

```
AS LI

ABCD-EA      0010      .BA $ABCD
ABCE-CDAB    0020KNURD NOP
ABDO-COAB    0030      .SE KNURD
              0040      .SI KNURD-$D
              0050      .EN
```

The label KNURD has the value \$ABCD, which is stored in \$ABCD and \$ABCE as CD AB, which is the 6502's address format of Low, High bytes of the address. Line 40 shows the use of an expression and the .SI pseudo-op instead.

.DE and .DI are very handy to use - it permits you to use labels to name values which fall outside of the address range occupied by the program. Here is an example (which actually does something!) that shows how nice this is:

```

10 .BA $0300
20 .OS
30SCRN .DE $8000
40LINE .DI 40
50 LDA #00
60 LDX #00
70NEXT STA SCRN+LINE+LINE+LINE,X
80 INX
90 CPX #LINE
100 BMI NEXT
110 BRK
120 .EN

```

Line 30 defines SCRIN to be the start of the PET's screen at \$8000. Line 40 tells us that LINE is set to 40, the number of characters in one line on the PET's screen.

In line 70, we combine SCRIN and LINE to start placing \$00 on the fourth line of the PET's display. Here, SCRIN and LINE serve to specify an address. In line 90, LINE is used to check the loop counter, X for the branch in line 100. LINE is used as the value 40 in this part of the program.

If you assemble this program, and in the MONITOR use G 0300, a line of "@" will appear on the screen's fourth line. (This will end up on the top line if, as usual, the screen scrolls after executing the program.)

Simple challenge - change line 90 to fill three lines with the "@" character. Harder - how do you fill three lines with the "!"? Worse - Why won't another change to line 90 for four lines full work correctly? (Hint: Read what CMP does.....)

If a label was defined by .DE or .DI, its value can be changed with the SET directive. Here is an assembled example of this:

```

AS LI
                                0010KICKAPOO      .DI $1000
9800-AD0010 0020                LDA KICKAPOO
                                0030                SET KICKAPOO=$2222
9803-AD2222 0040                LDA KICKAPOO
                                0050                .EN

```

LABEL FILE: [/ = EXTERNAL]

KICKAPOO=2222
//0000,9806,9806

Line 10 defines KICKAPOO to be \$1000, and this is reflected in the operand for the LDA in line 20. When the SET in line 30 is executed, KICKAPOO becomes \$2222, and this is seen in the assembled code for line 40.

If line 10 is deleted, and assembly attempted, we get:

```
AS LI
          0030 SET KICKAPOO=$2222
!04 AT LINE 0030
```

Take note that the !04 error message is listed differently in the error messages list. View it as "Can't evaluate operand".

The operation of SET can be explained by the following sequence:

- 1) During Pass 1, when SET is encountered, evaluate the operand & change the symbol table's value for the label. If the label before the = cannot be found in the Symbol Table, there is an error. If any labels in the expression after the = cannot be found in the Symbol Table, this is also an error.
- 2) During Pass 2, when the operands for the op-codes are being calculated from the Symbol Table, re-execute the SET directive when found.

The assembly below illustrates the resulting hazard:

```
AS LI
9800-AD0001 0010FUNNY      LDA $100
9803-AD5555 0020          LDA FUNNY
              0030          SET FUNNY=$1234
9806-AD3412 0040          LDA FUNNY
              0050          SET FUNNY=$5555
              0060          .EN
```

We would expect the assembled Line 20 to become AD 00 98 which is the address of the label FUNNY before the SETs were executed. Why is it AD 55 55? During Pass 1, the final value for FUNNY was \$5555 due to the SET in line 50. Pass 2, which provides the operands, looked up FUNNY in the Symbol Table for line 20 and discovers the value \$5555 and then places this in the operand.

When we arrive at line 40, FUNNY was changed to \$1234, and Pass 2 now uses this as the operand.

If you use `.DI` to define a label which is later SET, this problem won't arise, for the `.DI` pseudo-ops will establish the value of the label in both passes of the assembly.

In normal use, there are two justifications for the SET directive:

- 1) There isn't any more room in the Symbol Table. (unlikely)
- 2) A label must be changed due to a conditional assembly directive.

When a program contains many label values, it is possible that a "circular" definition for a label can exist. MACROTEA will then give wildly unreasonable values for "circular" labels. Let's watch this in action:

```
AS LI
      0005          .BA $0200
      0010MAN      .DI MAN+MAN
      0020          .EN
LABEL FILE: [ / = EXTERNAL ]

MAN=0800
```

The story goes like this: During Pass 1, MACROTEA finds MAN at the program counter's value of \$200. This is put into the Symbol Table and the operand for the `.DI` pseudo-op is now evaluated. Well, $\$200 + \200 is \$400, and the Symbol Table's entry for MAN is changed to \$400.

Pass 2 repeats this story - since no new entries are placed into the Symbol Table, only the operand is evaluated, so $\$400 + \400 becomes \$800, and MAN is updated again - to \$800.

It is clear that any complex labelmaking will give wild and hairy results with this double-evaluation process. When you finish reading about SET, see if you can determine what `15 SET MAN=MAN+MAN+3` will do. (urk!)

CONDITIONAL ASSEMBLY DIRECTIVES

<u>IEQ</u>	Assemble if value of operand is zero.
<u>IMI</u>	Assemble if value of operand is less than zero.
<u>INE</u>	Assemble if value of operand is not zero.
<u>IPL</u>	Assemble if value of operand is equal to or more than zero.
***	End of conditional block.

As we all know, the passage of time brings changes to our lives and possessions. In the universe of PET, for example, there are two versions of the PET ROM operating system, and hints of more changes to come.

If you have either changed from "old" to "new" ROMs, or have written a BASIC program for operation on both versions of the PET, you know that many important values, such as the BASIC program's memory pointers, or the keyboard input buffer, have been moved - and your BASIC program must contain some code to account for these changes. (With the loss of some memory space for this otherwise "useless" code.)

In MACROTEA, this problem can be circumvented by the use of the conditional assembly directives. Use the .DI or .DE pseudo-op to define a label, MODEL to the value of 1 or 2. Then collect all of the values which depend on the PET's ROM version and build two blocks. The first block would be a series of .DE or .DI which define the values for the "old" ROM, and the second block does the same for the "new" ROM. By using the IFE directive, the first or second block will be assembled according to the value of MODEL. Here is a brief diagram:

```

10MODEL .DE      (insert 1 or 2 depending on which ROM you use)
20 IEQ MODEL-1
30 .....
40 .....      (label definitions for "old" ROMs)
50 .....
60 ***
70 IEQ MODEL-2
80 .....
90 .....      (label definitions for "new" ROMs)
100 .....
110 ***
120 .....      (now for the program proper)

```

The IEQ, IMI, INE and IPL directives evaluate their operands, and if the value of the operand agrees with the appropriate condition, the following code will be assembled. If the operand does not agree with the condition, MACROTEA will ignore the following lines until the *** for "end of block" is seen.

You can "nest" conditional blocks as much as you want in MACROTEA. If "extra" block ending markers are seen, MACROTEA will ignore them. However, be warned - the MACROTEA's conditional assembly mechanism is very simple: If the condition is FALSE, skip lines until the first *** marker is seen. Any conditional directives within the skipped portion will be ignored. If the condition is TRUE, keep on assembling just as if nothing has changed. Ignore any *** markers.

The mnemonics chosen for the IF directives follow the 6502 branch instruction conventions. The only difference is that MACROTEA computes the value of the expression as a 16 bit signed integer and then applies the test condition. Here is a suggested way of remembering these:

IEQ - If Equal to zero.

INE - If Not Equal to zero.

IPL - If sign bit PLus. (ie, sign bit not set.)

IMI - If MINus. (ie, sign bit is set.)

Examples:

Just to check out the various conditional directives, try this short program:

```
FO CL
PR
0010 TASK .DI 5
0020 LDA #11
0030 IEQ TASK-1
0040 LDA #22
0050 ***
0060 IMI TASK-2
0070 LDA #33
0080 ***
0090 INE TASK-2
0100 LDA #44
0110 ***
0120 IPL TASK-2
0130 LDA #55
0140 ***
0150 LDA #66
0160 .EN
```

AS LI

```

0010TASK .DI 5
9800-A90B 0020 LDA #11
          0030 IEQ TASK-2
          0040 LDA #22
          0050 ***
          0060 IMI TASK-2
          0070 LDA #33
          0080 ***
          0090 INE TASK-2
9802-A92C 0100 LDA #44
          0110 ***
          0120 IPL TASK-2
9804-A937 0130 LDA #55
          0140 ***
9806-A942 0150 LDA #66
          0160 .EN

```

As expected, lines 20 and 150 were assembled. TASK-2 evaluates to 3, so lines 40 and 70 were skipped, and lines 100 and 130 were assembled. If you change line 10 to other values, a different pattern of assembly will appear:

Value of TASK	TASK-2	IEQ	IMI	INE	IPL
5	3	No	No	Yes	Yes
2	0	Yes	No	No	Yes
0	-3	No	Yes	Yes	No
\$FFFF	\$FFFD	No	Yes	Yes	No
\$8005	\$8003	No	Yes	Yes	No
\$8001	\$7FFF	No	No	Yes	Yes

The first three values of TASK assemble as expected with the different conditional directives. The last three values tell us that MACROTEA sees the most significant bit in a 16 bit value as a sign bit for these directives - so all values over \$7FFF (32767) are evaluated as negative. (In practical terms, this has little effect as conditional assemblies aren't usually made by testing address values.)

Here is an example of a nested condition:

```

0010TAG .DI 0
0020BAG .DI 0
0030 LDA #11
0040 INE TAG .....
0050 LDA #22 .....
0060 INE BAG .....
0070 LDA #33 ! Inner Condition ! Outer Condition
0080 *** .....
0090 LDA #44 .....
0100 *** .....
0110 LDA #55 .....
0120 .EN

```

When this is assembled with different values for TAG and BAG, we end up with:

TAG	BAG	Line	50	70	90
0	0		No	No	Yes
0	1		No	No	Yes
1	0		Yes	No	Yes
1	1		Yes	Yes	Yes

Study this carefully! In most cases, nesting conditional directives will not operate as you might expect. MACROTEA is very simpleminded, and as mentioned earlier, it skips code until the first *** marker it finds. Any conditionals within skipped code are ignored, and any extra *** markers are ignored.

When you nest conditions, try to not place any code after the *** in the innermost condition - that is, have the entire set of nested conditions terminate at the same *** marker.

MACROS

<u>.MD</u>	Begin a macro definition.
<u>.ME</u>	End a macro definition.
<u>.ES</u>	When listing the assembly, list the macro's code where a macro is referenced. (This is the default setting.)
<u>.EC</u>	When listing the assembly, don't list the code for any referenced macros.

NOTE: A discussion of macros & why they are useful is placed below. For reference reading, skip ahead to the section titled "The MACROTEA Macro Facility"

What Is A Macro & Why Are They Useful?

When you write a long program, or one that does a lot of highly similar operations, there will be segments of code which are repeated, but with enough variation to discourage writing a set of subroutines. Though a subroutine will often save some memory space, there are two major disadvantages:

- 1) A subroutine requires 12 cycles for the call and later return.
- 2) If a subroutine is handling changing data, you must arrange to pass the data to the subroutine. For small amounts of data, the 6502 registers must be set up. For medium amounts of data, you can place the data after the subroutine call, and within the routine manipulate the stack to first get the data using the stacked return address as a pointer and when finished changing the stacked address to return to some real code. For larger amounts of data, the registers are set up as a pointer to the data.

You pay a double price with data in that you must first set up for the subroutine, and when executing the subroutine, it must first fetch the data. This naturally takes more space & time.

Most repeated sequences of code perform standard operations on data placed at different addresses. If these addresses are directly assembled as operands for the instructions, the resultant code is usually more compact and faster than handling the addresses as data and then handling the data. Here is an example:

Suppose we want to add three numbers together:

```
CLC
LDA NUMBER1
ADC NUMBER2
ADC NUMBER3
```

This sequence takes 10 bytes and 14 cycles. Each time we want to do this, the code must be repeated with different values for NUMBER1, NUMBER2 and NUMBER3.

Now, let's see how to do this with a subroutine. When the routine is called, the numbers are in the A, X, and Y registers. Here is the routine and the calling sequence:

```

;ADDITION ROUTINE
ADD STX TEMP1
    STY TEMP2
    CLC
    ADC TEMP1
    ADC TEMP2
    RTS

;TO CALL THE ADDITION ROUTINE
LDA NUMBER3
TAY
LDA NUMBER2
TAX
LDA NUMBER1
JSR ADD

```

The routine itself requires 16 bytes (Remember that 2 bytes are needed for TEMP1 and TEMP2.) The time required (less the RTS) is 18 cycles.

The setup & call requires 14 bytes and 28 cycles (16 for setup, 12 for the call.) This is clearly a loser, no matter how many times the subroutine is called. The setup alone is more costly than doing the job directly.

A somewhat less costly approach is via the stack:

```

;ADDITION ROUTINE
;ASSUMES VALUES ON STACK JUST AFTER THE
;SUBROUTINE CALL
ADD PLP
    STA *00
    TAX
    PLP
    STA *01
    LDY #00
    CLC
    LDA (00),Y
    INY
    ADC (00),Y
    INY
    ADC (00),Y
    TAY

```

```

        CLC
        TXA
        ADC #03
        BCC HOP
        INC *01
HOP     LDA *01
        PHA
        LDA *00
        PHA
        TYA
        RTS

```

The saving obviously isn't in the subroutine - it is in the call:

```

JSR ADD
.BY NUMBER1 NUMBER2 NUMBER3

```

The subroutine call only takes 6 bytes, which saves 4 bytes over doing it directly. However, note these small points:

- 1) If NUMBER1, NUMBER2, and NUMBER3 are truly data which may be written into, the code can't be placed into ROM.
- 2) The subroutine takes around 100 cycles for all of the funny manipulations needed.
- 3) The space for the subroutine takes 36 bytes, which means you must call it at least 9 times to have a net savings of space. (Not to mention the 2 bytes in the zero page needed for the indirect pointer.)

A macro is an assembler construction that lets you give a section of code a name, which can then be used just like any other Op-Code or Pseudo-Op. This would be just a mere convenience except for a special feature - macros permit dummy variables. Here's our addition example in macro form:

```

!!!ADD .MD (A B C)
        CLC
        LDA A
        ADC B
        ADC C
        .ME

```

The call of the macro is quite simple:

```

ADD (NUMBER1 NUMBER2 NUMBER3)

```

When MACROTEA sees the call for ADD, the values for NUMBER1, NUMBER2, and NUMBER3 are substituted for A,B, and C, and code identical to our first sequence is generated. The next time ADD was used, we might see:

```

ADD ($101 FOOBAR+43 SIMPL-3)

```

Note that expressions are legal in the macro call, and that now a different set of values will be assembled in the places of A, B, and C.

The MACROTEA Macro Facility

The pseudo-ops .MD and .ME are used to define a macro. Here is a short example:

```
!!!ADD .MD (A B C)
      CLC
      LDA A
      ADC B
      ADC C
      .ME
```

The name of this macro is ADD. The three exclamation points are used by MACROTEA to distinguish the macro's name from an ordinary label. These are required.

After the .MD pseudo-op comes the arguments list. If the macro doesn't use any arguments, this may be omitted.

Macros which use arguments must have the dummy labels within a pair of parenthesis, and the labels must be separated with spaces. Don't use commas! The dummy labels can then be used within the macro's definition just like any other label.

When a macro is called, it is placed in the source text like any normal op-code. If the macro uses arguments, the macro's call must have the same number of arguments. The arguments must be used in the same order as in the definition (or thou shalt have much confusion). The arguments in a macro call may be expressions. In a macro definition, only labels are permitted as arguments. Here are some calls to our ADD macro:

```
ADD (1 2 3)
ADD (VALUE1 VALUE2 VALUE3)
ADD (OFFSET TAG+5 LINK+LINK)
```

Macros may be "nested", that is, one macro can refer to another macro in the first macro's definition. For example, our ADD macro could be extended:

```
!!!DOUBLEADD .MD (V1 V2 V3)
      ADD (V1 V2 V3)
      STA TEMP1
      ADD (TEMP1 TEMP1 ZERO)
      .ME
```

Here we assume that TEMP1 is a temporary memory location and that the cell at ZERO has the value \$00.

The maximum depth of "nesting" is 32 levels. In most cases, if you exceed 32 levels this indicates a programming error, such as a circular macro definition.

If you use any labels within a macro, they should be preceded with the ellipsis (...). If the ellipsis is omitted, and the macro is called more than once, you will receive the !06 error for duplicate labels. Here is an example:

```
!!!CRLF .MD (CHAR)
    LDA CHAR
    CMP #13
    BNE ...DONE
    JSR PRINTCHAR
    LDA #10
    JSR PRINTCHAR
...DONE .ME
```

If the contents of CHAR are a carriage-return, CRLF will print the carriage-return and then a linefeed. If not, nothing is done. (A more practical routine would place the label DONE at the last JSR PRINTCHAR & the macro would now be called PRINTIT.)

The ellipsis before DONE defines the label DONE as "local" to the macro, permitting more than one call of the macro in the source program. Note that if all labels were local to a macro there would be a problem with the label PRINTCHAR.

Due to the way in which MACROTEA tags labels used in macros, if you nest different macros which use the same labels (properly dotted ...) you might still get the !06 error. The cure to this is to use different label names within each different macro.

In summary on macro labels:

(LABEL)	This is a parameter, and must appear in the arguments list as a label.
LABEL	This is a "global" label and the label should be defined <u>outside</u> of the macro.
...LABEL	This is a "local" label. If you nest macros, use different ...LABELs within each macro.

When macros are used, take note that the macro's definition must come before you make use of that macro in the source text. Though nested macros aren't subject to this limitation (That is, you can define an inner macro after its use in another macro) inside other macro definitions, all macros that are used in a macro call must be defined before any of them are called.

When multiple files are used in an assembly, the macro definitions must be repeated. This comes from the fact that MACROTEA completely clears the workspace before loading each file in a multi-file assembly and that the source text is used for the macro's definition each time a macro is called. (When a macro is called, MACROTEA finds it in the source text in the workspace & reassembles accordingly. A newly loaded file will overwrite any old macro definitions.)

As a result, you can use different macros with the same name in multifile assemblies. Of course, it's up to you to keep all things straight.

Another thing to beware of is that you cannot define one macro within another macro's definition. In most cases this is no problem.

When MACROTEA reaches Pass 2, the assembly listing is generated, and a decision must be made concerning the code that results when a macro is called. The .ES and .EC pseudo-ops control an "expansion flag" for this purpose. Use of .ES will make MACROTEA list the assembled code in the listing, and use of .EC will not list the macro generated code.

When Pass 2 is started, the "expansion flag" is set by default to "off" and no expansion will be provided until the .ES pseudo-op is seen. Unlike .LS and .LC, the "expansion flag" is always set to "off" at the start of Pass 2. In practice, just be sure the .ES is placed just before or after the macro definitions.

Examples:

Since we are already familiar with ADD, let's use it for the first example:

```
FO CL
PR
0010!!!ADD .MD (X Y Z)
0020 CLC
0030 LDA X
0040 ADC Y
0050 ADC Z
0060 .ME
0070 STA $100
0080 STX $101
0090 STY $102
0100 ADD ($100 $101 $102)
0110 .EN
```

When this is assembled, we see:

```
AS LI
0010!!!ADD .MD (X Y Z)
0020 CLC
0030 LDA X
0040 ADC Y
0050 ADC Z
0060 .ME
9800-8D0001 0070 STA $100
9803-8E0101 0080 STX $101
9806-8C0201 0090 STY $102
0100 ADD ($100 $101 $102)
0110 .EN
```

LABEL FILE: [/ = EXTERNAL]

X=0100 Y=0101
Z=0102

//0000,9813,9813

The reason for the dummy labels X,Y and Z is that LDA A generates an error in MACROTEA as the 'A' is seen as an illegal addressing mode instead of as a label.

First, note that the labels X, Y and Z are placed in the symbol table. Their values are whatever resulted from the last call of the macro ADD. To check this, add: 101 ADD (1 2 3) and after assembly look again at the symbol table. X, Y and Z are now 1,2 and 3 respectively.

The code for ADD didn't appear since .ES wasn't in effect. However, the program counter value in the assembly report shows the end of assembly at \$9813, and line 90 was assembled at \$9806

If .ES is included, say at line 65, the assembled code is now displayed:

```

..... same as before .....
9806-8C0201 0090 STY $102
              0100 ADD ($100 $101 $102)
              0010!!!ADD .MD (X Y Z)
9809-18      0020 CLC
980A-AD0001  0030 LDA X
980D-6D0101  0040 ADC Y
9810-6D0201  0050 ADC Z

```

(The line with .ME isn't printed.)

```

              0110 .EN
..... same as later .....

```

For fun, try moving the .ES line 65 to line 45 - the listing of the macro will now commence within the macro.

With .ES at line 65, try the assembly again with line 101 included. (That's 101 ADD (1 2 3)). You will see that the operands will have changed to the new values for the macro's arguments.

Now for an attempt at nested macros. These will be very short so you can see the results on the PET's screen. (Almost! Press SPACE to "freeze" the screen and press any other key to resume the assembly listing.)

When things get tangled, .ES can be very helpful to see what's going on. For fun, try moving the .ES from line 65 to line 45. Now the macro's expansion will appear within the macro. (MACROTEA does things very literally.)

AS LI

```

0010 .ES
0020!!!MULCH .MD
0030 LDA #$22
0040 .ME
0050!!!FARM .MD
0060 LDA #$33
0070 MULCH
0080 LDA #$44
0090 MULCH
0100 .ME
9800-A911 0110 LDA #$11
0120 FARM
0050!!!FARM .MD
9802-A933 0060 LDA #$33
0070 MULCH
0020!!!MULCH .MD
9804-A922 0030 LDA #$22

9806-A944 0080 LDA #$44
0090 MULCH
0020!!!MULCH .MD
9808-A922 0030 LDA #$22

0130 MULCH
0020!!!MULCH .MD
980A-A922 0030 LDA #$22

0140 .EN

```

You will notice that MACROTEA has the nice habit of providing a line for each line within a macro, including comments, the .MD and .ME pseudo ops and the lines with some code on them. Examination of the listing above reveals that:

Address	Came From
9800	Main program line 110
9802	Macro FARM line 60
9804	Macro MULCH line 30 nested in FARM line 70
9806	Macro FARM line 80
9808	Macro MULCH line 30 nested in FARM line 90
980A	Macro MULCH line 30

As a challenge in nested macros, write a brief program which will fill all 65536 possible memory locations with a pattern of your choice. Try to do this with a minimum of source text lines. Start at address zero - so you will need .BA 0000. I did it in 43 lines, and am sure you can do better than that.

(Digression: A similar, and much tougher task is to write a 6502 program which will fill all of memory with zeroes. That includes the program itself, so it vanishes!) (Let me know if you succeed ... G. Y.)

Each time a more powerful capability is added to a programming tool, more powerful errors are possible. Debugging macro mistakes can be a non-trivial task.....

Here is a mystery error:

```
10!!!MACRO
20 LDA #$11
30 .ME
40 MACRO
50 .EN
```

AS LI

!OC AT LINE 7EOE

!OC is "Bad Character In Label" - with a rather strange line number. The cure is to see that line 10 didn't include the .MD pseudo-op.

Take a look at the List of Macro Related Error Messages in the Summaries part of this manual. Rather than to bore you with ten pages of ways and means to get these error messages, you are encouraged to write a few short programs to do these errors. The understanding gained is probably worth the time spent.

Some of the errors will be superceded by more obvious errors. For example, !20 will be masked by !02, which is the "Illegal Opcode" error. If you attempt to name a macro with an op-code's name, such as LDA, MACROTEA will either assemble the opcode or tell you that an error appeared in the operand.

A much more insidious error is to call a macro with the arguments in a different order. For example, suppose you have:

```
!!!SUBT .MD (FIRST SECOND)
  LDA FIRST
  SBC SECOND
  .ME
```

Clearly, SUBT (2 1) will give a different result than SUBT (1 2).

If you get the !06 error (duplicated label), the culprit is usually that you forgot to use the ...LABEL form for labels within your macros. Remember that this form defines a label to be "local" to a macro, and if you call the same macro twice, the ellipsis is required for any "local" addresses or values. Also be sure that each macro you use has different label names for internal labels or arguments.

Last, be sure that any multifile assemblies have their macros defined at the start of each file. If possible, don't use different macros of the same name in different files - or you will be condemned to many fruitless hours.

Monitor in Detail

THE MONITOR

The Monitor is the third major part of MACROTEA. Once you have assembled your program, the Monitor lets you directly examine & change the memory and registers of the 6502. The Monitor permits several other utilitarian operations at the machine code level.

MACROTEA MONITOR REFERENCE CARD

NO.	COMMAND-SYNTAX	MNEMONIC	ACTION	PAGE
1	A	allclear	Exit to MACROTEA coldstart(erases everything)	146
2	BREAK or BR	break	Initialize and/or enter Monitor from MACROTEA	146
3	B (addr) (count)	breakpoint	Set soft breakpoint for Q command	159
4	C	close	Send Monitor output to screen	166
5	D (addr) D (addr1) (addr2) ...23 lines...scroll range...	disassemble	Display memory in hex and 6502 source text	152
6	F (addr1) (addr2) (byte)	fill	Write-memory range with specified byte	155
7	G (addr)	goto	Execute 6502 code	158
8	H (addr1) (addr2) (byte seq) H (addr1) (addr2) ('ASCII seq) ...hunt hex... ...hunt ASCII...	hunt	Display start addresses of specified sequence	153
9	I (addr) I (addr1) (addr2) ...23 lines...scroll range...	interrogate	Display and/or write memory in hex and ASCII	150
10	K	kleanup	Reset Monitor IRQ and PET I/O Status Byte	165
11	L "Ø:FILENAME", (device#) L "FILENAME", (optional device#)	load	Load disk or tape file into memory (Ø: or L: for disk drive Ø or L)	164
12	M (addr1) (addr2)	memory	Display and/or write memory in hex	149
13	O (device#) ...the letter oh gives device# 4... ...or specify any device#...	open	Send Monitor output to printer	166
14	Q (addr)	quicktrace	Execute 6502 code with soft breakpoint	159
15	R	registers	Display Monitor's copies of 6502 registers	148
16	S "Ø:FILENAME", (device#), (start addr), (end addr+1) S "FILENAME", (device#), (start addr), (end addr+1)	save	Save memory into disk or tape file (Ø: or L: for disk drive Ø or L)	164
17	SYS4	sysenter	Re-enter Monitor from BASIC	146
18	T (addr1) (addr2) (addr3)	transfer	Copy memory to another location	154
19	W (addr)	walk	Execute 6502 code with singlestep disassembly	161
20	X	exit	Exit to BASIC	146
21	Z	zipout	Exit to MACROTEA warmstart(keep environment)	146

ENTRY AND EXIT COMMANDS

<u>A</u> ...same as .G 9000 ...	Exit Monitor to MACROTEA coldstart(erase all).
<u>BREAK</u> or <u>BR</u>	Initialize and/or enter Monitor from MACROTEA.
<u>SYS4</u>	Warm re-entry to Monitor from BASIC (usually).
<u>X</u>	Exit to BASIC (return with SYS4).
<u>Z</u> ...same as .G 9085 ...	Exit Monitor to MACROTEA warmstart(keep work).

These commands initialize, start, and stop execution of the MACROTEA Monitor program. It uses and works with the PET's built-in monitor.

NOTE: When testing machine language code, a common experience is to lose control of the PET (i.e., a crash). When this happens two conditions are possible:

- 1) The program is in a loop. If you used the Q command to start execution, press the STOP key to get back. Otherwise there's no hope, and you'll have to press the MACROTEA reset button.
- 2) The 6502 choked on an illegal op-code. Illegal op-codes will often 'hang up' the 6502, and the only cure is to press the reset button.

Once the reset button is pressed, SYS36997 to get into MACROTEA (via the warmstart). If your code was in the \$9800 user's RAM, it will still be there. Everything else will be lost (most likely).

A (allclear) This command causes exit from the Monitor to MACROTEA and it entirely resets MACROTEA in the process. This means any text in the workspace, assembly symbol tables, FORMAT, SET, HARD, printer device numbers, etc., are cleared and set to the default values, so be careful!

BREAK or BR (break) Use this command to leave MACROTEA for the Monitor:

```
BR
B*                               (shows monitor entry via BRK)
      PC  IRQ  SR AC XR YR SP
.; 9085 E455 F1 FO 80 50 FF      (your numbers may be different)
.M                               (the cursor (M) flashes on this line)
```

When the BREAK command is issued the first time after power-on, the MACROTEA Monitor initializes a tie to the built-in PET Machine Language Monitor. The tie adds 14 additional commands to the Monitor's repertoire.

SYS4 (sysenter) This command allows you to re-enter the Monitor from BASIC once it has been initialized via the BREAK command. Location 4 usually contains a 00, and a SYS to any 00 in memory will cause a BRK instruction execution of the Monitor (as long as the Break Vector points to the Monitor).

X (eXit) The X command exits the Monitor and puts you in PET's BASIC command mode. Here you may execute BASIC programs or direct statements.

Z (zipout) Use this command to routinely leave the Monitor for a warmstart of MACROTEA. The warmstart arrives in MACROTEA with everything exactly as you left it. All settings, text, etc., are still there.

CODE MANIPULATION COMMANDS

<u>R</u>	Display monitor's copies of 6502 registers.
<u>M (addr 1) (addr 2)</u>	Display contents of memory in hex bytes, over range (addr 1) to (addr 2).
<u>I (addr)</u> or <u>I (addr 1) (addr 2)</u>	Display memory in hex bytes and in ASCII for 24 lines starting at (addr); or, range (addr 1) to (addr 2).
<u>D (addr)</u> or <u>D (addr 1) (addr 2)</u>	Disassemble 24 lines of code, starting at (addr). Disassemble continuously, over range (addr 1) to (addr 2).
<u>H (addr 1) (addr 2) (seq)</u>	Search for sequence of hex bytes in (seq) over range (addr 1) to (addr 2).
<u>T (addr 1) (addr 2) (addr 3)</u>	Copy bytes in range (addr 1) to (addr 2), to destination range starting at (addr 3).
<u>F (addr 1) (addr 2) (byte)</u>	Write every memory location in range (addr 1) to (addr 2) with the hex byte given in (byte).

The code manipulation commands permit the examination and modification of the contents of memory.

To change the values in memory or the registers, all you have to do is use the PET's Screen Editor. Just move the cursor about, make the changes, and press RETURN for every line you want to have entered.

NOTE: All numeric values in the Monitor must be in hexadecimal and be in either 2 digit or 4 digit form. If the Monitor does not understand a command, it will either ignore the command or print the question mark.

NOTE: The Monitor's grasp of numbers is a bit weak - if you by error enter some value like DEFG, the Monitor will translate the number to a value it understands instead. The reason is that some shortcuts were taken in the translation of characters to a numeric value (specifically some masking).

R (registers) The R command displays monitor copies of 6502 register contents. This includes the Program Counter, Interrupt ReQuest vector, Status Register, Accumulator, X Register, Y Register, and Stack Pointer.

```
.R
      PC IRQ SR AC XR YR SP
.; A013 E455 32 00 78 00 F6      (Your numbers may be different)
.M
```

This is the same display that the Monitor gives on entry via SYS from BASIC (except the B* indicating entry via 6502 BRK). The parts of the display in detail are:

```
PC - The value of the 6502 program counter.
IRQ - " " " " " interrupt request vector.
SR - " " " " " status register.
AC - " " " " " accumulator or A register.
XR - " " " " " X register.
YR - " " " " " Y register.
SP - " " " " " stack pointer.
```

If you wish to change these values, use the PET's screen editor to do so. For example, suppose you entered:

```
(Cursor-Up)(3 Cursor-right) FFFF FFFF FF FE FD FC FB (R)return
```

The Monitor's Status Registers are now changed. To verify this enter the R command:

```
.R
      PC IRQ SR AC XR YR SP
.; FFFF FFFF FF FE FD FC FB
.M
```

If you know the purpose of the IRQ, you may be wondering why the system didn't crash when the IRQ address was changed to \$FFFF. This will be discussed in more detail under the K (kleanup) command, but the short answer is that the register values displayed are copies of the actual registers made during the last previous appearance of B*,C*, or S*. If you alter these copies (which I'll call the Monitor's Status Registers) the changes won't be written to the actual PET operating system until and unless you give a G (goto) command. If you did a G right now, the PET would indeed go away, since there is no valid IRQ code at location \$FFFF.

NOTE: B*,C*, and S* indicate the mode of monitor entry; via BRK instruction, via subroutine Call, or via STOP key, respectively.

M (memory) The M command will display the memory contents in hexadecimal starting at the first address given and continuing in 8 byte lines until the second address is reached or passed. The addresses must be given in the form AAAA BBBB - that is, M will recognize 0123 but will not recognize 123. The two addresses must be separated by one space (or any other character - the separating character is utterly ignored.) The STOP key will terminate an unfinished scroll.

```
.M 0330 0350
.: 0330 FB FB FB F9 FB FB FB F9
.: 0338 FB FB FB F9 FB FB FB F9
.: 0340 91 81 81 81 81 81 81
.: 0348 81 81 81 81 81 81 81
.: 0350 01 81 81 81 81 81 81
```

M displays the memory in 8 byte blocks. The number on the left is the starting address for the line.

Your display may differ in the values for the memory. By using the screen editor, these values may be changed. As an exercise (and to set up for the next example), change the display to:

```
.: 0330 00 01 02 03 04 05 06 07
.: 0338 08 09 0A 0B 0C 0D 0E 0F
.: 0340 10 11 12 13 14 15 16 17
.: 0348 18 19 1A 1B 1C 1D 1E 1F
.: 0350 20 21 22 23 24 25 26 27
```

Repeat the M 0330 0350 to check that the change was correctly entered. It is easy to miff and press the DEL key, or forget to press RETURN after each changed line.

If the Screen Editor doesn't appeal to you, you can enter the line you want to change in exactly the format used in the display. Don't forget the spaces, they are important. If you are changing only a few bytes, this may be simpler to do.

```
.M 0360 0360
.: 0360 81 81 81 81 81 81 81      (your numbers may be different)

.: 0360 AA BB CC ?
.M 0360 0360
.: 0360 AA BB CC 81 81 81 81
.M
```

Here the memory at 0360 to 0367 was displayed. Then the first three bytes were changed by entering the line in the same format. Note that the entire line wasn't required; although, the Monitor did take notice of the missing portion by printing a question mark. A reaplication of M verified that the change was correctly accepted.

Be a bit careful of typing errors. The monitor will cheerfully accept letters beyond F in creating hexadecimal numbers. For example, try:

```
.M D000 D00G
```

This will display from \$D000 thru \$D017. My classic error is to try:

```
.M D000 D0FF
```

This gives me a display from \$D898 to \$D8FF. Why, is because of the ancient letter O versus number 0 human bug.

Spend some time trying different memory changes. Attempt to force the Monitor to accept errors or unusual input - and learn how the Monitor looks at its input.

To see how MACROTEA places code in memory, look at the examples provided for the .OC and .OS pseudo-ops.

I (interrogate) The I command does everything that the M command does, and in addition it has an "ASCII map" - an interpretation of every memory byte as though it were an ASCII character. This map appears in reverse video at the right of the PET screen, next to the normal hex byte display. There are eight ASCII characters displayed in each line, corresponding to the eight hex bytes in the same line. A dot is displayed if no printable character corresponds to the hex byte.

The I command accepts the same address parameters that the D command does, and with the same restrictions. A single address will display 23 lines. A double address will scroll over the memory range specified. The RVS key will slow scrolling on most PETs. The backarrow key slows scrolling on machines with a business keyboard. The STOP key will stop the display on all models.

Here is an example of I command use:

```
.I 0330 0350
```

```
. ' 0330 00 01 02 03 04 05 06 07 .....
. ' 0338 08 09 0A 0B 0C 0D 0E 0F .....
. ' 0340 10 11 12 13 14 15 16 17 .....
. ' 0348 18 19 1A 1B 1C 1D 1E 1F .....
. ' 0350 20 21 22 23 24 25 26 27 !"#$%&'
.M
```

This is the same area of memory which you experimentally filled in and examined with the M command. It looks very similar, and is, but there are some subtle differences. First, notice that the character immediately following the prompt dot is an apostrophe instead of a colon as with M. Next, note that there is only one space between the apostrophe and the line address where the M command has two. The ASCII map to the right has nothing but dots in the first four rows and that makes sense since ASCII \$00 thru \$1F characters are not printable.

In the last row the first printable character, \$20, is a space, followed by special characters in ascending ASCII order.

Just like M, the I command has a memory writing capability. Move the cursor back up to the fifth line and over to the 20. Change the 20 to a 40, and press RETURN. The ASCII map is instantly rewritten to replace the space with a Commercial At (@) sign. To be absolutely certain that the change really occurred do:

```
.I 0350 0350
```

```
. ' 0350 40 21 22 23 24 25 26 27 @!"#$%&'
.M
```

Yep. That worked fine; the change was written to memory.

Just in case you continue to use the M command either through force of habit or to avoid the glare of the ASCII map when you don't need it, it's nice to know that a single line of M can be converted to a line of I. Change the colon to an apostrophe and press RETURN:

```
.M 0350 0350
.: 0350 40 21 25 23 24 25 26 27
.' 0350 40 21 25 23 24 25 26 27 @!"#$%&'
.M
```

Make sure that there is only one space between the line address and the first byte when you do this; otherwise the line will be rewritten with garbage. If necessary, you can pull the whole row of bytes over with the DEL key.

The ASCII map has a bug or a feature depending on your point of view. In generating the map it ignores the most significant bit 7. The feature is that you can read ASCII characters which have been encoded for font control by toggling bit 7 (you'd see a graphic otherwise on some PETs). The bug is that there are two hex bytes which can generate every ASCII character in the map. Thus, \$C9 and \$49 both interpret as an "I".

D (disassemble) The invocation of D will disassemble the machine language code until a screen full is displayed. The disassembly starts at the address specified after the D. To display more pages of disassembly, just press the RETURN key. Press SHIFT-RETURN and then RETURN to get back to the prompt dot of the Monitor's command mode.

By specifying two addresses after the D (AAAA BBBB) you can get a continuous disassembly over the address range specified.

A disassembled 6502 program with instruction mnemonics and addresses laid out can be quite convenient when you are attempting to find an instruction you want to patch, or to look at machine language programs which came without any listing.

Just for the fun of it (not to mention the example), let's go for a romp in the PETs BASIC 4.0 ROM:

```
.D D000

., D000 85 5B      STA $5B
., D002 86 5A      STX $5A
., D004 8A         TXA
., D005 F0 02      BEQ $D009

.... etc ....

., D026 85 62      STA $62
., D028 A5 61      LDA $61
.D D02A 79 CE D0   ADC $DOCE,Y
```

On the PET screen 24 lines of disassembled code will appear. The cursor will be on the 'D' in the last line. To continue the disassembly just press RETURN. The screen will now display another 23 lines of code.

(Press RETURN)

```
., D02A 79 CE D0   ADC $DOCE,Y
., D02D 85 61      STA $61

.... etc ....
```

Note that the top line is the same line that was on the bottom in the last frame of disassembly. To leave the disassembly, press SHIFT-RETURN and then RETURN, or move the cursor down once and then press RETURN.

If an illegal 6502 opcode is discerned, the disassembler will print ???. For example, try:

```
D B000

., B000 C7        ???
```

One gentle warning about disassemblies - the location you choose to disassemble from might not be where the first 6502 instruction begins. To wit:

D D000

```
., D000 85 5B      STA $5B
., D002 86 5A      STX $5A
```

D D001

```
., D001 5B        ???
., D002 86 5A      STX $5A
., D004 8A        TXA
., D005 F0 02      BEQ $D009
```

The catch is that sometimes there are no ??? marks, because some sequences of data just coincidentally happen to have valid "disassemblies". Sometimes a strange-looking branch will clue you, but not always. In general, always be suspicious of the disassembly at the top of the screen. The bottom of the screen can be trusted usually, since an incorrect disassembly will usually synchronize after a few op codes, as it did above.

H (hunt) The H command will search the memory in the range given by two addresses in form AAAA BBBB. The sequence of bytes to be searched for is placed after the second address as hexadecimal numbers, CC DD EE ... and so on up to 32 bytes. Alternatively, the search sequence may be in ASCII characters preceded by an apostrophe ('FIND ME). Up to 32 ASCII characters may be searched for.

With the memory set up as in the previous example for the M command, here is a search for \$0A:

```
.H 0330 0350 0A
033A
.M
```

Location 033A held the value 0A, which is easily verified with the M command. If more than one byte is being searched for (usually an instruction you want to find), just provide the sequence:

```
.H 0330 0350 0A 0C
.M
```

Here there is no 0A 0C sequence. However, 0A 0B or 0A 0B 0C will work fine. When a matching sequence is found, only the address of the first byte in that sequence is printed.

If we go looking through the PET ROM (BASIC 4.0 version), it's easy to illustrate how rare sequences of two or more bytes are. First we'll look for all the addresses of a single kind of byte:

```
.H D000 E000 A9
D00B D014 D062 D089 D091 D097 D09C D0B3 .....(and on, and on,
and on, for a total of 105 entries.)
```

So there are lots of single bytes of a given kind. Now we look for a two-byte sequence:

```
.H D000 E000 A9 00
D091 D0B3 D0BD D0C2 D1CE D26C D29B D2D5 .....(and so on for a
mere 17 entries, total.)
```

For this particular sequence there are about 80% fewer instances. Now look for a three-byte sequence:

```
.H D000 E000 A9 00 38
D091
```

Hmmm - only one instance found. But let's try another 3-byter:

```
.H D000 E000 A9 00 01
```

.M

None were found this time. The point is that your chances of getting a random response to a 3-byte hunt are small, while there's not much chance that a single byte hunt will be worth the keystrokes.

T (transfer) The T command is used to move blocks of code from one point to another. The first two addresses specify the start and finish for the block of code. The third address tells where the first byte of the code block is to be placed. Any code at the destination will be overwritten by the new code.

With the code from 0330 to 0350 set up as in the preceding examples, (That is, 0330 is 00, 0331 is 01 etc to 0357 as 27) here is a brief transfer:

```
.T 0338 033F 0330
```

This tells the Monitor to move the code starting at 0338 and ending at 033F to a new location starting at 0330. Use M to see if this is so:

```
.M 0330 0340
```

```
.: 0330 08 09 0A 0B 0C 0D 0E 0F
.: 0338 08 09 0A 0B 0C 0D 0E 0F
.: 0340 10 11 12 13 14 15 16 17
```

Indeed, the code at 0330-0337 has been replaced with the code at 0308-033F.

T will permit overlaps between the old code and the new code's locations:

```
.T 033A 0342 0340
.M 0330 0348

.: 0330 08 09 0A 0B 0C 0D 0E 0F
.: 0338 08 09 0A 0B 0C 0D 0E 0F
.: 0340 0A 0B 0C 0D 0E 0F 10 11
.: 0348 12 19 1A 1B 1C 1D 1E 1F
```

The range 033A-0342 overlaps with the destination 0340-0348. The underline is included to show the moved code more clearly. The other direction of transfer works as well:

```
.T 034A 034F 0348
.M 0348 0348

.: 0348 1A 1B 1C 1D 1E 1F 1E 1F
```

The reason for showing the variations on T is to reassure you that T is safe to use under all circumstances. An incorrectly written block transfer routine will typically write into an area it hasn't read yet, and if the moved code overlaps the destination area, this will rapidly create trash.

F (fill) The F command is very simple. It takes parameters in the form AAAA BBBB CC, where AAAA is first addresses to fill, BBBB is the last address to fill, and CC is the hexadecimal byte to fill it with:

```
.F 0330 0337 EA
.I 0330 0330

.' 0330 EA EA EA EA EA EA EA EA *****
.F 0330 0337 FF
.I 0330 0330

.' 0330 FF FF FF FF FF FF FF FF ????????
```

In the example, I just filled the same area of memory twice to show that it works as advertised. What to use F for is a tougher problem. I can think of two uses.

First, use F as an eraser when you have to type in tables of bytes. This is never an easy job, but it will be easier if you are typing on a smooth field of 1's. When I type over garbage, I'm always losing my place when I scroll over what I've done.

Second, an extension of the first idea. Use F to format the boundaries of tables which you are about to type:

```
.F 0330 0342 11
.F 0343 0349 22
.F 034A 0354 33
.F 0355 0357 44
.I 0330 0350
```

```
. ' 0330 11 11 11 11 11 11 11 11 .....
. ' 0338 11 11 11 11 11 11 11 11 .....
. ' 0340 11 11 11 22 22 22 22 22 ...""""
. ' 0348 22 22 33 33 33 33 33 33 ""333333
. ' 0350 33 33 33 33 33 33 44 44 44 333333DDD
```

If you have four different byte tables to type, it is now very obvious where they should start and stop. If you type the last byte from your worksheet for Table 2, but have one remaining 22 on the screen, then it's time for an investigation. You may have skipped a byte on the worksheet, which is easy to do.

CODE EXECUTION COMMANDS

<u>G (addr)</u>	Jump to location (addr) and execute code. Return to Monitor on BRK (\$00) instruction.
<u>Q (addr)</u>	Execute code at (addr) but return to Monitor: 1 - via a Breakpoint. 2 - by pressing STOP key.
<u>B (addr) (count)</u>	Set soft Breakpoint pointer to (addr) and return to Monitor when (addr) has been approached (count) times in a loop.
<u>W (addr)</u>	Execute code at (addr) in single-steps with disassembly at each step. Return to Monitor by pressing STOP key.

These commands provide the ability to execute code.

The commands G, Q, and W will use the value in the Program Counter if no address is provided. Sometimes this is good, sometimes you must be cautious! (In doing the examples. I lost the PET several times by not being observant.)

G (goto) The G command causes a jump to the specified address. The PET will then execute the program code at that location until a BRK (\$00) instruction is encountered. The BRK will cause a return to the Monitor registers display.

Here is a short program that prints "@" on the screen starting at the fifth line. You will recognize it as a simpler version of the program used to illustrate the .DI pseudo-op.

AS LI

```

0010 .BA $300
0020 .OS
0300-A900 0030 LDA #00
0302-AA 0040 TAX
0303-9DA080 0050NEXT STA $8000+200,X
0306-E8 0060 INX
0307-D0FA 0070 BNE NEXT
0309-00 0080 BRK
0090 .EN

```

This listing assembles to a start address of \$0300 and an end address of \$0309.

The M command will show you the program as it is assembled:

```

.M 0300 0308
.: 0300 A9 00 AA 9D C8 80 E8 D0
.: 0308 FA 00 20 20 20 20 20

```

Now exit to BASIC:

```

.X
READY.

```

Once assembled, you can execute this program from BASIC with SYS768 . When you do, 6 and one-half lines of "@" should appear on the screen - and guess what? You are now in the Monitor due to the BRK instruction at the end. To see the program again, and for the point of this example, clear the screen (for greater drama) and do:

```

.G 0300

```

So the G is just like a SYS but from within the Monitor.

Q (quicktrace) The Q command will execute code starting at the given address and keep track of whether the breakpoint has been approached, how many times the breakpoint was approached, and whether the STOP key has been pressed. When the (count) number of approaches to the breakpoint have elapsed, Q will halt and leave you in the W mode (walk or single-step) with a display of registers and disassembly. You may press the STOP key to leave Q or W.

The advantages of Q are 1) you can press STOP and get back from a loop, and 2) you don't have to always modify memory to set hard BRK's, run a bit of code, restore the original code, and so on. The price of Q is that it runs more slowly. Time-critical code must be debugged via G and the BRK instruction.

B (breakpoint) The B command is a subfunction of the Q command. B sets a soft breakpoint pointer to a specified address which will be skipped for the number of times determined by the (count) value. For example:

```
.B 1000 0010
```

...will set a breakpoint in an imagined space just before \$1000, which will be ignored for 16 times before becoming active. (Remember that 0010 is 16 in decimal.) On the 17th approach to \$1000, the breakpoint triggers and exit occurs prior to passing \$1000.

B does not alter memory. It just sets a pointer which Q examines. Warning: B works only with Q. If you attempt a G, be sure the memory contains a \$00 (a hard break, BRK) to permit a return to the Monitor.

Q & B Now let's see how Q and B can be used. First, let's set a breakpoint prior to the INX at \$0306 (in the G command example program):

```
.B 0306 0000
```

Now to execute the code, clear the screen and enter:

```
.Q 0300
```

```
22 00 00 00 F6 0306 E8      INX
```

```
@
```

The first "@" was printed on the screen and then the breakpoint at \$0306 was triggered. Next, the 6502 registers and the current instruction (not yet executed) are displayed. You are now in the W mode. The registers are not displayed in the same order as with the R command. The actual order of appearance is:

```
22 00 00 00 F6 0306 E8      INX
/  /  /  /  /  /  /      /
SR AC XR YR SP PC      CODE  DISASSEMBLY
```

Leave the W mode by pressing the STOP key, and now home the cursor. Press two cursor-rights and press spaces until you are left with the line:

Q

...and press RETURN. Now the registers display will show:

20 00 01 00 F6 0306 E8 INX

...and there will be two "@" on the screen.

If you look at the X register, it contains the value 01. Now since we have arrived at the INX twice, so this shows us that:

A BREAK VIA Q & B STOPS JUST BEFORE THE DISPLAYED INSTRUCTION.

The Monitor's B isn't very fancy - you will have only one breakpoint to use in this manner.

Also note that the Q command without an address used \$0306, which was the value of the Program Counter. To complete the program, press STOP, clear the screen, and enter:

.G

...and press RETURN. If all goes well, you will see the following:

B*

PC IRQ SR AC XR YR SP
.; 030A E455 32 00 00 00 F6
.M

@@.....and a lot of these!

G will also use the current value of the Program Counter if you don't supply an address. (Use G with caution... if you were to use it again right now without an address, you would lose your PET.)

OK, now clear the screen and enter:

.B 0306 0050
.Q 0300

20 00 50 00 F6 0306 E8 INX

@@
@@
@

The value 0050 is 80 in decimal, so we can expect to see 80 "@" appear on the screen. Since the program prints the "@" before the INX, we see 81 "@" on the screen. Press STOP, home the cursor, and type .Q (no address). Press RETURN and another 81 "@" will appear.

For a little fun, continue to STOP, HOME cursor, and RETURN, until you see the Monitor's B* (hard break) display... there's something a little funny here - the registers seem to print out very slowly. Clear the screen (gee - the screen clears from bottom up, and in sections!) and enter:

```
.M 0300 0310
```

```
...etc...
```


Hmmmmmm - that sure takes a long time!


The solution to this puzzle lies in that we are still in Q mode. After leaving the program, Q is still checking for the STOP key and the breakpoint, even though we are now executing the Monitor's code. Press STOP to get back to normal. (Don't try Q on top of Q - it makes the PET go away.)


When you press STOP to leave Q, the Monitor will display a S* before the registers instead of the usual B*. This tells you that the Monitor was called by the STOP key instead of the BRK instruction.

W (walk) W is the single-step command. The code will be executed starting at the specified address. Each instruction will generate a display of the registers, the program counter, and a disassembly of the instruction being executed. Pressing the < key will singlestep execute the next instruction. If RVS is held down, W will continue to step at about 2 steps per second. If SPACE is held down, W will step at about 12 steps per second. (SPACE will override RVS if both are pressed.) Press STOP to exit Walk mode.

If you have a PET with a business keyboard (the keyshifts are those of a standard typewriter) three differing keys are used for the three W step functions just mentioned:

For business keyboard singlestep use the  key - (not the numeral pad)

" " " slow step " "  "

" " " fast step " "  " - (not the numeral pad)

The firmware in the ROM provided with the business keyboard has an additional feature not available on other PETs. Anytime output is being scrolled on the screen, it can be temporarily halted with the colon (:) key. To restart the scroll, use the backarrow (←) key. This applies not just to the W command but all output, whether or not the MikroMan program is up.

W will provide a more detailed look at our little program which prints all the "@"s. To see how the program works, enter:

.W 0300

22 00 A1 00 F6 0302 AA TAX

Some examination of this is instructive. First, the instruction LDA #00 was executed, since the AC register holds \$00. Second, the XR register has the value A1 in it (or some other value depending on what you did last) which indicates that the TAX has not executed yet. To see the next instruction, press the RETURN key once:

22 00 00 00 F6 0303 9D C8 80 STA 80C8,X

The TAX is now executed, but the @ hasn't appeared, for the STA is the next instruction to be done. Press the RVS key and leave it down for a few seconds. Each instruction will be executed, and will be shown at about 2 instructions per second. A careful look at the screen will reveal the "@" character here and there. (Remember that the scrolling of the screen will move the "@" upwards as the walk is performed.)

To go faster, hold the SPACE key down. With a sharp eye, you can see the '@' appearing furtively amidst the register displays in the Walk.

To leave Walk, press the STOP Key. If you now enter W without an address, you will continue from where you left off.

FILE OPERATION COMMANDS

<u>S</u>	<u>"Ø:FILENAME",(device#),(start addr),(end addr+1)</u>	Save memory into disk file; Ø: or 1: for drive Ø or 1.
or S	<u>"FILENAME",(device#),(start addr),(end addr+1)</u>	Save memory into tape file.
<u>L</u>	<u>"Ø:FILENAME",(device#)</u>	Load disk file into memory; Ø: or 1: for drive Ø or 1.
or L	<u>"FILENAME",(optional device#)</u>	Load tape file into memory.
<u>K</u>		Reset Monitor's IRQ Status Register to normal and System's I/O Status Byte to zero.
<u>O</u>	...the letter oh...	Send Monitor output to printer device #4;
or O (device#)	...also the letter oh...	or to specified device #.
<u>C</u>		Restore Monitor output to screen device #3.

NOTE: The K command is needed only for computers with BASIC 2.0(Ver3) because the S command mangles the IRQ sometimes and the L command can mess up multiple disk loads. The K command exists in the BASIC 4.0 version of MACROTEA, but has little utility.

S (save) Once you have a working program in machine language, it is wise to immediately store it on diskette or tape. The S command does this in a format called "program file" (there are other kinds of files). When using S, the syntax must be followed with some care:

S "0:FILENAME",08,0300,0400

This will save the area of memory 0300 thru 03FF to a file on disk drive 0. The S is followed by a space. Next, a double quote mark. Then the drive number 0 or 1 followed by a colon. Then enter the filename followed by another double quote mark and a comma. Next the device number, usually 08 for the disk drive, followed by a comma. Then the starting address in hex of the memory area to be saved followed by a comma. And finally, the end address+1 in hex of the memory area to be saved.

The addresses must be in the usual 4-hexadecimal-digit form. Note carefully that the end address which S uses must be one more than the address of the last byte in your program.

If you wish to save a file on cassette tape, follow the same format as above with two changes:

S "FILENAME",01,0300,0400

The changes are to eliminate the drive number and colon, and make the device number 01 or 02 for tape drive #1 or #2 respectively.

If you wish to omit the filename for tape, type "" that is, make the filename no characters between the quotes. For disk drive you must specify a unique filename with at least one letter.

If you press the STOP key during a tape operation you will break into PET's BASIC mode. Type SYS4 to get back to the monitor.

After the tape operation is complete, you may use .X to exit to BASIC and do a VERIFY to make sure that the file is OK.

L (load) To load a file which has been saved on disk:

L "0:FILENAME",08

The format is the same as that for S except for no start-end addresses, and the same rules apply otherwise.

L "FILENAME",01

And likewise for tape.

K (kleanup) The K command resets the monitor's IRQ Status Register to its normal address, and resets the I/O Status Byte (ST) to zero.

.K
.M

K is needed to correct a pair of bugs left behind after use of the S (save) and L (load) commands which are part of the built-in BASIC 2(Ver3) monitor. The BASIC 4.0 monitor is ok and does not require use of this command, although it is present and functional.

S BUG: At the conclusion of a disk or tape save file operation done from the monitor, the first two letters of the chosen file name are incorrectly written into locations reserved for the monitor's IRQ Status Register (held in the BASIC Input Buffer). If an R command is done, the monitor's IRQ Status Register will be displayed containing a false IRQ address. It doesn't matter unless a G command is to be executed; other commands work ok.

If a G command is now executed, the false IRQ address will be written into the operating system's IRQ RAM Vector at locations \$90-91 (BASIC 2). The system will crash within 1/60th of a second when the next hardware interrupt tries to execute nonexistent code at the phoney IRQ address.

Use K after an S file save is finished. The K command will rewrite the monitor's IRQ Status Register to the correct address of \$E62E (BASIC 2). (If MACROTEA isn't up, you can fix the false address manually by typing the correct address over the registers display and pressing return.) Another application of R will now display a correct IRQ address.

L BUG: At the conclusion of a disk load file operation (not tape) done from the monitor, the operating system's I/O Status Byte at \$96 (BASIC 2)(same as BASIC's Status Word/reserved variable ST) may incorrectly retain an End-Of-Information flag set in bit 6. In such case, an attempt to load the next file will cease after one byte has been loaded, since a set E-O-I flag says the task is complete.

Use K between multiple L disk loads. The K command will rewrite the operating system's I/O Status Byte to the correct value of zero. (If MACROTEA isn't up, you can fix the false status manually by doing .M 0096 0096 ,typing 00 over the first displayed byte, and pressing return.)

The only use for the K command in BASIC 4 is to save yourself a few keystrokes in restoring a normal IRQ address (\$E455) after using an experimental value. Since K only restores \$E455 in the monitor, this value is then written to the system by doing a .G 0004 ,so the G rewrites the IRQ RAM Vector, and the 00 in location 4 BRK's back to the monitor.

O (open) The O (letter oh) command will send the Monitor's output to some device other than the screen, specifically to a printer. There have been numerous occasions when I wanted to get a printout of a disassembly for a machine language program. O will do this, assuming printer device 4:

```
.O                                (...the letter oh...)

.D 0300 030A                      (what you type appears on the screen)

                                   (but the output appears on the printer)
```

This example should give a printed listing of the program typed in for the G command. If you should just happen to have a printer with a different device number, O will accept an optional device number:

```
.O 05                             (...the letter oh, space, zero, five...)

.I 0300 030A                      (what you type appears on the screen)

                                   (output goes to device 05)
```

This should give a hex byte dump and an ASCII map on printer device 05 if all went well. However, your printer may choose not to cooperate with the output of the I command. The hex bytes will probably appear ok, but since the ASCII map is reverse video, your printer may or may not garbage those characters. If the output from I is unsatisfactory, use the M command instead.

C (close) The C command reverses what the O command did. It restores output from the monitor to screen device 3 again:

```
.C

.M 0300 0300
.: 0300 A9 00 AA 9D C8 80 E8 D0
.M
```

And it's back to business as usual. Of course, if you wanted to, you could achieve the same result by typing .O 03 (letter oh, space, zero, three), but why bother since that's four keystrokes instead of one?

MEMORY USAGE

When MACROTEA is in operation it requires the first 128 bytes of zero page, and also the top 1K of your computer's memory. Never load object code into that 1K region. If you need to load object code into that area, use the .BA and .MC assemble commands to place the object code in some other area. Then, with MACROTEA off, relocate that code to to where you want it. Of course, this will require that you have a separate relocating program.

12. 0000 2

1000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000

0000 0000 0000



MACROTEA

INSTALLATION

Congratulations on your purchase of a Skyles Electric Works MacroTeA. You have bought the latest, best and most cost effective 6502 micro-processor software development system available. There are several models of the Commodore PET computer. Additionally there are several competitive brands of memory expansion connected to the side port on the original PETs. Please confirm in which of the following models you will be installing MacroTeA: 2001-4, 2001-8, 2001-16N/B, 2001-32N/B. Memory expansion: Skyles Electric Works, ExpandaPET, ExpandaMem, or RC Factor-PME 1-Eventide Clockworks. The following installation instructions are written for all the above models of PET and memory expansion. Where there is a difference in installations, they are clearly noted.

First, let us prepare the PET (all models):

Place your PET on a well-lighted work surface. Make sure you have turned off the power and unplugged your PET from the wall socket.

Unscrew the four screws on the bottom of the top half of the PET (about 5" back from the front of the PET on each side). Use the correct size phillips-head screwdriver to avoid damaging the screws.

SLOWLY open the PET's case by lifting the front. Inside will be several cables, some of which might be too short to allow full opening of the PET. Find these cables and gently disconnect them from the PET's main circuit board. When the case is fully open, set the bar (on the left side) so that the case will remain opened.

Second, installation in the PET (by specific models):

A: For PET models 2001-4 and 2001-8 without internal memory:

1. Remove existing TK80P or TK160P BASIC Programmers Toolkit from the right side port of your PET. Carefully remove the Toolkit "chip" (the biggest device) and install it in the empty socket on your MacroTeA board. Note that the "notch" end of the Toolkit chip is down as shown in figure 1.

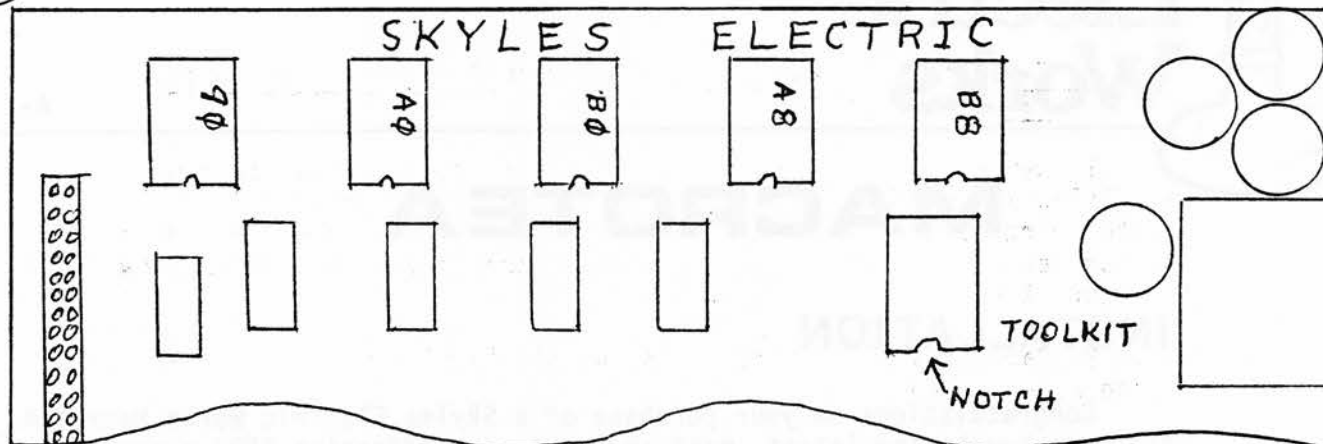


FIGURE 1

2. Remove single screw from center right side of main PET electronics board. Place Skyles MacroTeA board on top of PET board so that PET bus adapter cable is over at the right outside of the PET chassis. Locate screw mounting hole or holes (see A on photo) over center right side hold of main PET electronics board.

3. Place plastic spacer underneath the screw mounting hole or holes and insert the one inch self-tapping screw through the mounting hole and spacer and into center hole of the PET main board. Tighten screw securing the Skyles board to the PET board.

4. Plug PC board edge connector into memory expansion connector on right side of PET. The flat cable connector should be on the top inside of the PET bus adapter board when the PC board edge connector is plugged in.

5. Unplug power harness connector (brown, red and black wires) from five-pin header on left front side of PET main logic board. Plug this harness into five-pin header (C) on MacroTeA. Make sure that the five-pin power harness connector is not shifted to one side of the five-pin header.

6. Plug the power cable connector (D) from the MacroTeA board into the five-pin header on left front side of PET main logic board. Check that the five-pin connector is not shifted to one side of the header. All five header pins must go into cable connector and not be visible on either side of the connector.

7. Make sure that all internal wiring harnesses clear the heat sinks (E) at the back of the Skyles board.

8. Hold top cover of the PET, place brace bar back into holder. Close PET. Replace screws on each side of cover. Plug PET back into wall power socket.



B: For PET models 2001-4, 2001-8 with Skyles Electric Works memory expansion 8KB, 16KB or 24KB:

1. Remove the TK80S or TK160S BASIC Programmers Toolkit from the Skyles memory expansion board and ribbon cable receptacle. Carefully remove the Toolkit chip and install it in the empty socket on your MacroTeA board. Note that the "notched" end is down as shown in figure 1.

2. Carefully plug MacroTeA in the 50 pin ribbon cable socket (receptacle) connector.

3. For 8KB, 16KB and 24KB Skyles memory expansions, unplug the PET power harness (white connector brown, red and black wires) from the 8KB, 16KB or 24KB memory expansion board. Plug this harness into the five-pin header on the Skyles MacroTeA board. Make sure that the five-pin PET power harness connector has not shifted to one side of the five-pin header.

4. Plug the five-pin power harness connector from MacroTeA into the now empty five-pin header on the Skyles 8KB, 16KB or 24KB memory expansion board. Check that the five-pin connector is not shifted to one side of the header. All five header pins must go into the cable connector and not be visible on either side of the connector.

5. Carefully "snap" the MacroTeA plastic standoffs onto the Skyles memory expansion board. Note MacroTeA fits on top of the 8KB memory board, on the top "back" of the 16KB memory board and the top front of the 24KB memory expansion board assembly.

6. Make sure that all wiring harnesses clear the heat sinks at the back of the Skyles Memory and MacroTeA boards.

7. Hold top cover of the PET, place brace bar back into holder. Close PET. Replace screws on each side of cover. Plug PET back into wall power socket.

C: For PET models 2001-8N, 2001-16N/B, 2001-32N/B:

1. Carefully remove the existing TK160N BASIC Programmers Toolkit from the socket on the PET main logic board. Install the TK160N "chip" into the empty socket on the MacroTeA board. Note that the "notch" end of the Toolkit chip is down as shown in figure 1.

2. On the right side of the PET main electronics board located in the back near J9 is a jumper plug (see figure 2 for location). With a pair of clippers or a sharp pointed blade break link "P". With the small piece of bare wire (furnished) and a soldering iron reconnect link "N" just in front of P.



3. Holding the MacroTeA board in your left hand over the rear of the PET main electronics board with the ribbon cable on the right and the two power cables on the left, connect P9 to J9 and P4 to J4 with the labels (P4,P9) facing toward the right outside of the PET. P4 and P9 go onto the inside row of pins. The outside row of pins remains open.

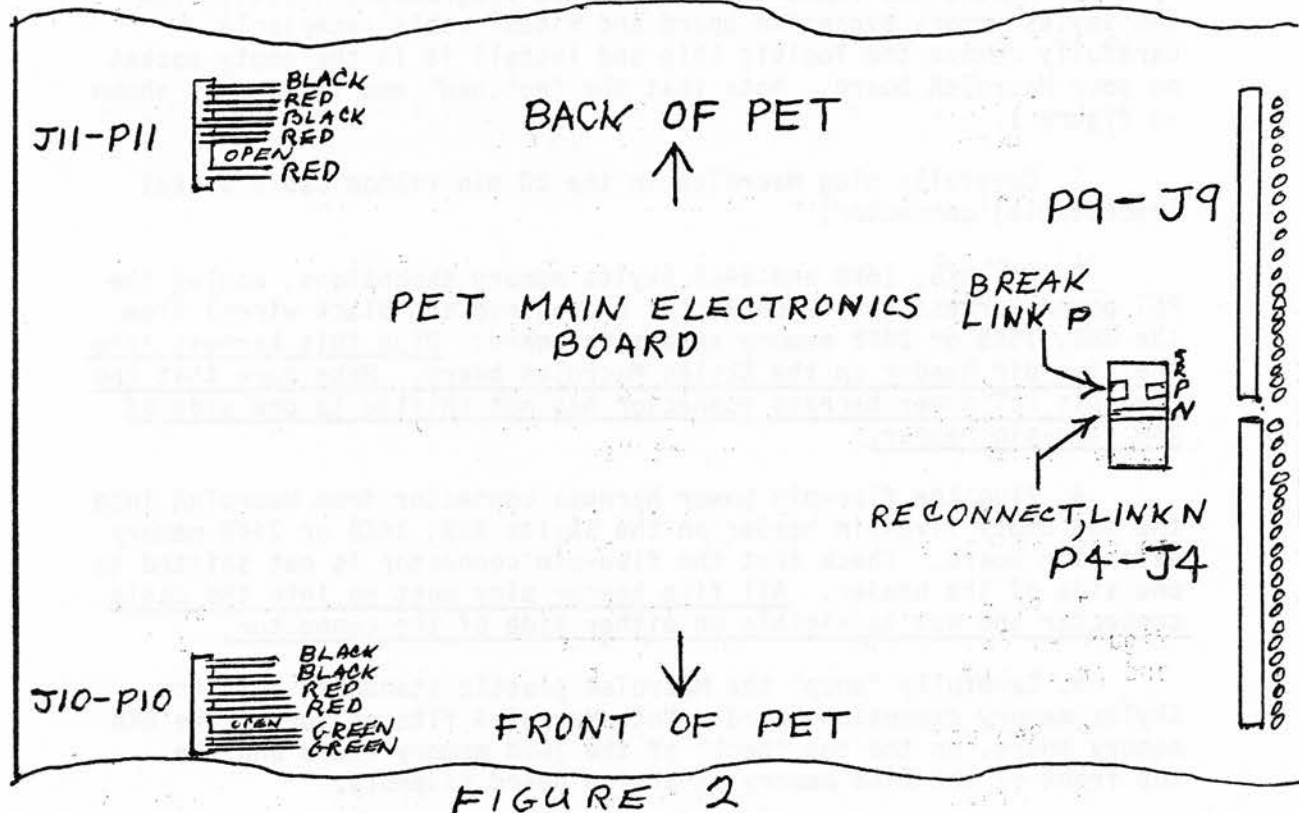


FIGURE 2

4. Holding the MacroTeA board in your right hand over the rear of the PET main electronics board, connect P11 to J11 (the red and black harness) and P10 to J10 (the red, green and black harness). Check that P10 and P11 have not slipped to one side of the headers J10 or J11.
5. Carefully place the MacroTeA board onto the right rear of the PET main logic board.
6. Remove single screw from center right side of main PET electronics board. Place Skyles MacroTeA board on top of PET board. Locate screw mounting hole over center right side hole of main PET electronics board.
7. Place plastic spacer underneath the screw mounting hole and insert the one inch self-tapping screw through the mounting hole and spacer and into center hole of the PET main board. Tighten screw securing the Skyles board to the PET board.
8. Make sure that all internal wiring harnesses clear the heat sinks at the back of the Skyles board.
8. Hold top cover of the PET, place brace bar back into the holder. Close PET. Replace screws on each side of cover. Plug PET back into wall power socket.



D. For PET Models 2001-4, 2001-8 with ExpandaPet or ExpandaMem memory expansion 8K, 16K or 24K:

1. Remove the TK80E or TK160E BASIC Programmers Toolkit from the ExpandaPet, ExpandaMem memory expansion board. Carefully remove the Toolkit chip and install it in the empty socket on your MacroTeA board. Note that the "notched" end is down as shown in figure 1.

2. Carefully remove the ExpandaPet ribbon cable (50 conductor) from the ExpandaPet and from the PET side expansion port.

3. Carefully remove the ExpandaPet/Mem from the PET.

4. Carefully remove the 2 to 4 extra links from the ExpandaPet, ExpandaMem header as shown in figure 3. For more information about this header, please refer to the ExpandaPet manual.

5. Holding the MacroTeA in one hand in approximately its final location (see figure 4), connect the 50 conductor MacroTeA ribbon cable connector to the ExpandaPet/Mem.

6. Check that the MacroTeA ribbon cable is identically connected to the ExpandaMem/Pet as it is to the MacroTeA.

7. Place the MacroTeA on top of the ExpandaPet/Mem as shown in figure 4. Remove the green backing paper from the plastic standoffs and press firmly to attach MacroTeA to the ExpandaPet/Mem.

8. Replace the Expanda - MacroTeA into the PET.

9. Remove the 5 pin power connector from the PET main electronics board. Connect this connector assembly to the 5 pin power header on the MacroTeA board. Be sure that the 5 pin connector has not been allowed to slip to one side of the MacroTeA power header.

10. Connect the 3 pin ExpandaMem/Pet power connector back into the ExpandaMem/Pet 3 pin header. Be sure that this connector has not slipped to one side of the header.

11. Now plug the ^{MacroTeA} 5 pin power cable connector to the 5 pin header on the PET main electronics board. Be sure that this connector has not slipped to one side of the header.

12. Plug the MacroTeA - Expanda ribbon cable assembly into the side port of the PET.

13. Review the above steps, checking for error.

14. Make sure that all wiring harnesses clear the heat sinks at the back of the ExpandaPet/Mem and MacroTeA boards.

15. Hold top cover of the PET, place brace bar back into holder. Close PET. Replace screws on each side of cover. Plug PET back into wall power socket.



- 3.1 STANDARD 16K OR 24K IN 8K PET.
- 3.2 STANDARD 32K IN 8K PET.
- 3.3 STANDARD 16K, 24K OR 32K IN 4K PET.
- 3.4 16K EXPANDAMEM 12K CONTINUOUS TO 8K PET,
4K WITH WRITE PROTECT IN BLOCK 7.
- 3.5 24K EXPANDAMEM 20K CONTINUOUS TO 8K PET,
4K WITH WRITE PROTECT IN BLOCK 7.
- 3.6 32K EXPANDAMEM 8K WITH WRITE PROTECT IN BLOCKS 6 & 7.
- 3.7 DISK SYSTEM ALL MEMORY SIZES: SINGLE DENSITY *NOT AVAILABLE WITH MACRI.*
- 3.8 DISK SYSTEM ALL MEMORY SIZES: DOUBLE DENSITY " " " "



BACK OF PET ↑

A-7

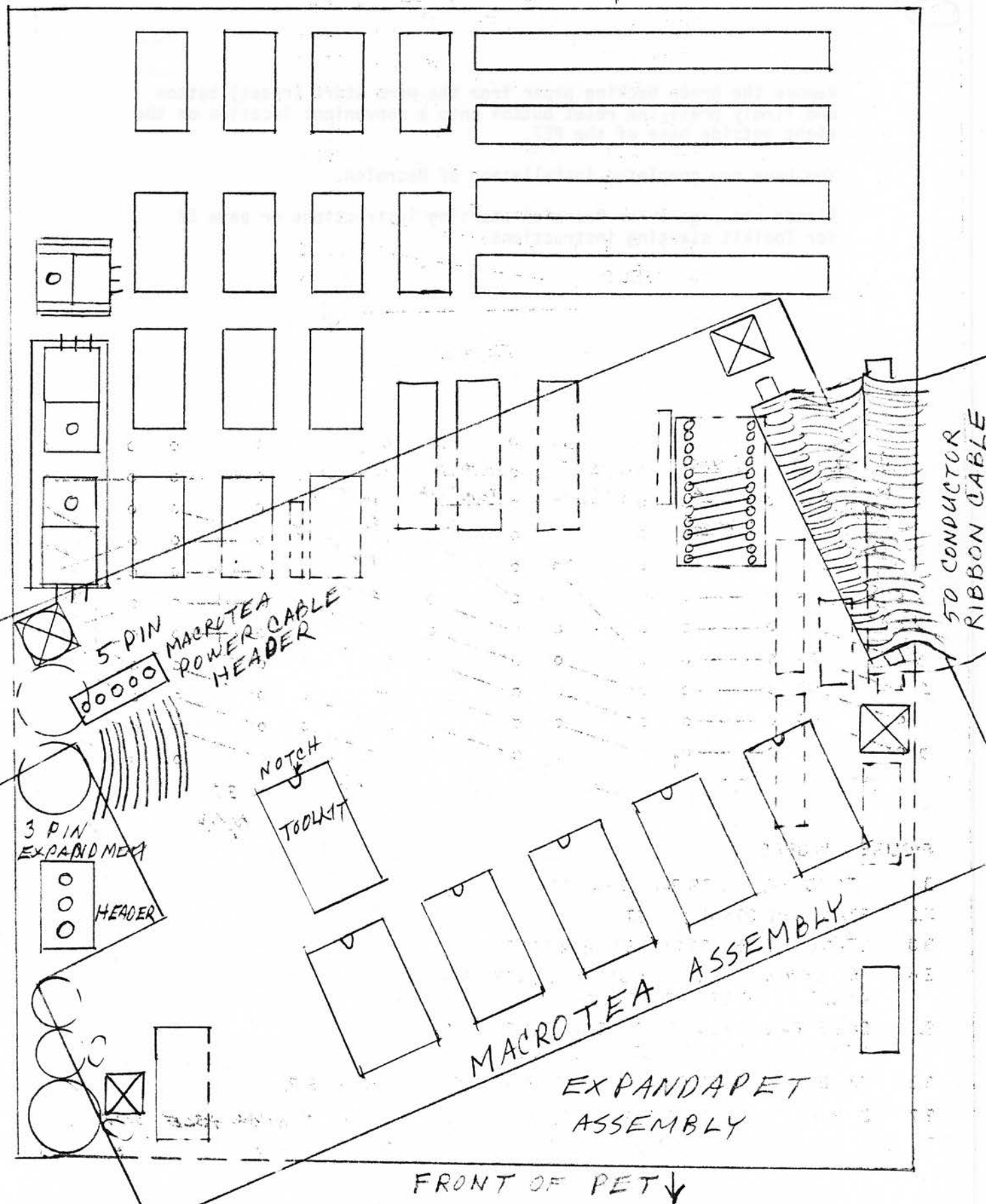


FIGURE 4



Remove the green backing paper from the warm start (reset) button and firmly press the reset button onto a convenient location on the right outside base of the PET.

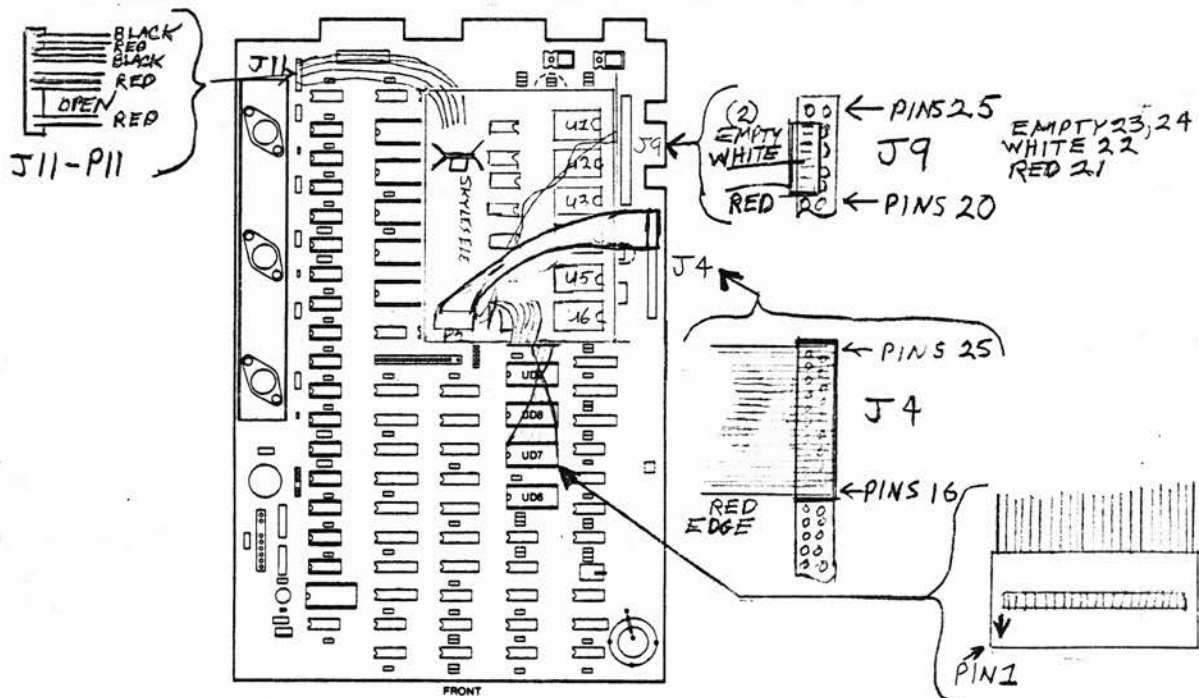
You have now completed installation of MacroTeA.

Please see page 5 for MacroTeA starting instructions or page 22 for Toolkit starting instructions.



B. For PET/CBM 4016, 4032, with 12" screens (Fat Forties), 8032, SP9000, 8096.

- 1) Remove the ROM/EPROM/ROM assembly (Socket-2-ME, Go-4-IT, etc.) from UD12 on the main logic board and install in UD4 of the MacroTeA board. Note the notch direction.
- 2) Remove the ROM/EPROM/ROM assembly from UD11 on the main logic board and install in UD3 on the MacroTeA board - be careful of the notch direction.
- 3) Remove the ROM 901499-01 or 901474-03/901474-04 from UD7 and install in UD5 on the MacroTeA board.
- 4) Place the MacroTeA board in the right rear quarter of the main logic board. The MacroTeA ROM/EPROM notches should be to the right and the power regulator heat sink to the rear.
- 5) Plug the red/white, 2 wire cable assembly into pins 21, 22, 23, 24 of the inner row of J9 on the main logic board. See detail - Figure 1.
- 6) Plug the 16 wire gray ribbon cable assembly onto the rear 8 pins of both rows of J4. See detail - Figure 1. Note the red edge stripe should be to the rear. You will cover but not plug into pins 16 and 25.
- 7) Plug the 24 pin cable socket assembly into UD7 on the main logic board. Pin 1 should be to the left front of the socket.
- 8) Plug the 5 wire 7 pin power cable into J11 at the left rear of the main logic board. A black wire should be to the rear and a red wire to the front. The connector is keyed.
- 9) Carefully check all cable plug-ins against the instructions and Figure 1 details.





10) Close your PET/CBM reconnecting any cables originally unplugged.

11) Turn on your PET/CBM, listen for the chirp and observe the turn on message.

If the "Chirp" occurs, but the screen is full of random characters, you probably have the 20 wire (step 6) cable forward one position on J4. Move it to the rear one or two positions.

If no chirp occurs and the screen never comes on or is filled with characters, you probably have the connection to UD7 wrong. Check the pin 1 position.

If you smell smoke, etc., you probably connected the power cable backwards.

12) Once the PET/CBM is running type:
POKE45056,199:SYS4 <return>

you should see:

B*

PC IRQ SR AC XR YR SP
.; 0005 E455 30 00 5E 04 F8

Then type: M 9000-900F <return>

you should see:

.M 9000 900F
.: 9000 20 EF E9 A9 5D 8D 7E 7F
.: 9008 A9 E9 8D 7F 7F A2 FF 9A

Then type: M A000-A00F <return>

you should see:

.M A000 A00F
.: A000 D0 0F 20 28 93 28 90 03
.: A008 4C C1 93 B1 3D D0 EA D8

Then type: M E900-E90F <return>

you should see:

.M E900 E90F
.: E900 20 41 54 20 4C 49 4E 45
.: E908 00 0D 0A 0A 0A 4C 41 42

13) If you observe above a printed readout of \$A000-\$A00F when you type 9000-900F, move the 20 pin gray ribbon cable (step 6) one position to the rear and repeat step 12.

14) If you observe the above printed readout of \$9000-\$900F when you type A000 A00F, move the 20 pin gray ribbon cable (step 6) one position to the front and repeat step 12.

1. The purpose of this document is to provide a comprehensive overview of the current status of the project and to identify the key areas for improvement. The document is organized into several sections, each of which addresses a specific aspect of the project. The first section, "Introduction," provides a brief overview of the project and its objectives. The second section, "Current Status," provides a detailed overview of the project's progress to date. The third section, "Key Areas for Improvement," identifies the areas of the project that require the most attention and resources. The fourth section, "Recommendations," provides a list of specific actions that should be taken to address the identified areas for improvement. The fifth section, "Conclusion," provides a summary of the document's findings and a final statement of the project's status.

2. The project has made significant progress since the last report. The initial phase of the project, which involved the collection and analysis of data, has been completed. The second phase, which involved the development of a prototype, is currently in progress. The third phase, which involved the testing and evaluation of the prototype, is also in progress. The project is on track to meet its deadline, and the results of the testing and evaluation are expected to be positive. The key areas for improvement identified in the previous report are being addressed, and the recommendations are being implemented. The project is expected to be completed by the end of the year.

3. The project has made significant progress since the last report. The initial phase of the project, which involved the collection and analysis of data, has been completed. The second phase, which involved the development of a prototype, is currently in progress. The third phase, which involved the testing and evaluation of the prototype, is also in progress. The project is on track to meet its deadline, and the results of the testing and evaluation are expected to be positive. The key areas for improvement identified in the previous report are being addressed, and the recommendations are being implemented. The project is expected to be completed by the end of the year.

4. The project has made significant progress since the last report. The initial phase of the project, which involved the collection and analysis of data, has been completed. The second phase, which involved the development of a prototype, is currently in progress. The third phase, which involved the testing and evaluation of the prototype, is also in progress. The project is on track to meet its deadline, and the results of the testing and evaluation are expected to be positive. The key areas for improvement identified in the previous report are being addressed, and the recommendations are being implemented. The project is expected to be completed by the end of the year.

5. The project has made significant progress since the last report. The initial phase of the project, which involved the collection and analysis of data, has been completed. The second phase, which involved the development of a prototype, is currently in progress. The third phase, which involved the testing and evaluation of the prototype, is also in progress. The project is on track to meet its deadline, and the results of the testing and evaluation are expected to be positive. The key areas for improvement identified in the previous report are being addressed, and the recommendations are being implemented. The project is expected to be completed by the end of the year.

6. The project has made significant progress since the last report. The initial phase of the project, which involved the collection and analysis of data, has been completed. The second phase, which involved the development of a prototype, is currently in progress. The third phase, which involved the testing and evaluation of the prototype, is also in progress. The project is on track to meet its deadline, and the results of the testing and evaluation are expected to be positive. The key areas for improvement identified in the previous report are being addressed, and the recommendations are being implemented. The project is expected to be completed by the end of the year.



15) To test that the ROM/EPROMs, you removed from UD11 and UD12 and installed in UD3 and UD4 type: X <return> to exit the monitor to BASIC. Then type: POKE 46106,251:SYS4 <return> Then type: M 9000 900F <return> and see:

ROM TYPE

```
.M 9000 900F
.: 9000 4C 0C 90 4C 45 92 4C 29      Command-0 CO-40N
.: 9008 94 4C 87 94 A9 4C 85 70
```

```
.M 9000 900F
.: 9000 4C 0C 90 4C 3D 92 4C DD      Command-0 CO-80N
.: 9008 93 4C 3B 94 A9 4C 85 70
```

```
.M 9000 900F
.: 9000 3E 41 5D 55 3D 00 3F 48      VISICALC
.: 9008 48 48 3F 00 41 7F 49 49
```

these are ROMS/EPROM you moved from UD12 to UD4.

Now type: M A000 A00F

and see:

```
.M A000 A00F
.: A000 4C 75 A2 A9 0A 8D C2 03      Toolkit TK40N
.: A008 A9 00 8D C3 03 85 83 85
```

```
.M A000 A00F
.: A000 01 0A 64 64 0A 01 0D 8D      WordPro 4+ (5054)
.: A008 13 93 5F DF 5C 03 83 12
```

these are ROMs you moved from UD11 to UD3.

16) Once MacroTeA is running satisfactorily, you might consider screwing down the front rightfoot with the screw in the right side center of the logic board.

17) Now go to page 5 of the manual to start MacroTeA or page 22 to turn on the Command-0 or the Toolkit.

CONFIDENTIAL - SECURITY INFORMATION
This document contains information that is exempt from public release under the provisions of the Freedom of Information Act, 5 U.S.C. 552, and is to be controlled, stored, handled, transmitted, and disposed of in accordance with the provisions of Executive Order 11652, as amended, and the provisions of the Department of Defense Information Security Manual, 100-20, as amended. This information is to be controlled, stored, handled, transmitted, and disposed of in accordance with the provisions of Executive Order 11652, as amended, and the provisions of the Department of Defense Information Security Manual, 100-20, as amended.

1. NAME: [REDACTED]
2. DATE: [REDACTED]
3. TIME: [REDACTED]
4. LOCATION: [REDACTED]
5. SUBJECT: [REDACTED]

6. ACTION: [REDACTED]
7. STATUS: [REDACTED]
8. COMMENTS: [REDACTED]

9. APPROVAL: [REDACTED]
10. DISTRIBUTION: [REDACTED]
11. STORAGE: [REDACTED]
12. DISPOSAL: [REDACTED]

13. OTHER: [REDACTED]
14. [REDACTED]
15. [REDACTED]
16. [REDACTED]
17. [REDACTED]
18. [REDACTED]
19. [REDACTED]
20. [REDACTED]