MACHINE LANGUAGE DEVELOPMENT SYSTEM

EDITOR • ASSEMBLER • MONITOR

100 Graydon Hall Drive, Suite 2301, Don Mills, Ontario, Canada M3A 3A9

Notice

Micol Systems reserves the right to make improvements in the product described in this manual at any time and without notice.

Limit of Liability

While every precaution has been made as to the validity of the software and its accompanying manual, Micol Systems and Micol Systems Canada cannot assume any responsibility or liability for any damage or loss caused by our software. If you have any questions or problems, please write to our address in Canada. We will respond as soon as possible.

SECOND EDITION
FIRST PRINTING-1984

Commodore 64 and VIC 20 are trademarks of Commodore Business Machines, Inc.

TABLE OF CONTENTS

PREFACE

Welcome to the world of Micol Systems Assembly Language Programming on the VIC 20 or Commodore 64. With this package you have the ability to create the fastest and most versatile software your computer is able to execute.

You will be amazed at the speed differences between programs you will write with this package and BASIC programs which perform the same function. A forty fold speed increase or better should be the norm. Hundreds of times, although unusual, can be achieved.

Few things in life come for free, and assembly language programming is no exception. While assembly language programs execute far faster than BASIC programs, they are usually more difficult to write. This package will be a great aid in minimizing this difficulty. The monitor, for example, is indispensable for assembly language debugging and offers features found only on much more expensive computers. You will find your investment well worth the money.

This manual is not intended as an assembly language tutorial, but rather as a tutorial on the software contents on the disk or cassette you received with this manual. It is highly recommended you obtain a good 6502 assembly language manual. Even advanced programmers need such a manual now and again for reference. In addition, the VIC 20 or Commodore 64 Programmer's Reference Manual will be indispensable for more advanced programming.

If your drive is not properly aligned, it may have trouble reading the disk. Try several times. If the original disk or cassette should prove defective so as to be unusable, return it to us with an explanation of what is wrong. We will mail another. This offer will be honored up to 30 days from the original date of sale.

We at Micol Systems wish to make this machine language development system the best of its kind. If you have any questions, problems or discover any bugs, please write to us at our address in Canada. We will reply to correspondence as soon as possible.

DESCRIPTION OF MANUAL

This manual is written as a general instruction book for all the versions being sold, from the 13K VIC RAM version to the Commodore 64 versions for both disk and cassette. Publication costs make this necessary, but as the purchaser has the option of upgrading to a higher memory or Commodore 64 version at a minimal charge, ($10.00 U.S.), this is also desirable. We hope this does not cause any difficulty to the reader.

This manual is broken up into five chapters, one for each piece of software on the disk or cassette.

Locations of system software specified are for the Commodore 64 (disk or cassette) and VIC 20 37K disk version. See Appendix A for complete information.

All of the software you have received is machine language code. This is easy to determine because a '.B' has been appended to the file name. With the exception of the MENU and copy program (disk version only) they cannot be RUN, but you will have to use the SYS command from the keyboard (does not apply to Commodore 64). If you go through the MENU routine, you will not have to worry about that, as the MENU routine takes care of program locations automatically.

It is important you make a backup of the master disk or cassette and keep the master safe as a backup (see Ch. V). Those of you with the cassette version will probably want to backup each file to a separate cassette in order to speed loading time.

WE WISH YOU GOOD PROGRAMMING

OVERVIEW

Contained on the disk or cassette are six files. Below is a brief description of each of the files. Following, in its own chapter, will be a detailed description of the first five.

MENU.B MENU routine for the system. This simplifies the use of these programs by giving the user the ability to access the editor, assembler or monitor by the press of one key. Must be used on the Commodore 64, as most of the software loads over the BASIC interpreter.

TED.B This is the Micol Systems Text Editor. This program gives you all the features you need for creating and editing assembly language text files.

ASSM.B This is the Micol Systems Assembler. This processes the program created using TED.B into machine code, the fastest code on your micro computer.

VICTOR.B This is the Micol Systems Monitor. It contains all the features you need for loading, saving, manipulating or debugging the programs created by ASSM.B.

COPY.B This program allows you to make backups, either from disk to disk, or cassette to cassette.

EXAMPLE An example program to study (use the editor), assemble and execute.

NOTE I. It is assumed that in the cassette version, the user will automatically follow the instructions "PRESS PLAY ON TAPE" or "PRESS PLAY AND RECORD ON TAPE". No further mention will be made of this in the description of the software.
NOTE II. It is assumed that the disk drive unit is attached to device #8. This system is designed for a one drive system.
NOTE III. On syntax. When the syntax of commands is being discussed, the "<" and ">" enclose a description, not the actual letters typed. Anything enclosed in parentheses is optional.

WARNING: The wedge program should not be in memory while this software is running, it may cause the system to hang.

I MENU

I. MENU

File: MENU.B

Locations: $12ØØ (VIC) $8ØØ (CBM64)

Function: This is the control program. Through the MENU all system software can be accessed without worrying about where in memory it sits.

How to execute:

Disk version: Type LOAD"MENU.B",8,2<CR>
RUN<CR>

Cassette version: Be certain tape is rewound.
Type LOAD"MENU.B",1,1<CR>
RUN<CR>

Thereafter the MENU will automatically load and execute:

1. From the editor by exiting.
2. From the assembler after a program has finished assembling (if 'N' is typed).
3. With the Q(uit) command from the monitor.

Type of program: Machine Language.

General description: Upon a successful load and execution you will see this display:

1. Edit
2. Assemble
3. Monitor
4. Load
5. Exit
6. Set colors

By pressing one of the digits (without a carriage return) you will cause the loading of the system software, or a piece of software of your own choosing.

Detailed description:

Press "1"

By doing so you will cause the system text editor to load and execute.

Press "2"

By doing so you will cause the assembler to load and execute.

Press "3"

By doing so you will cause the system monitor to load and execute.

Press "4"

Disk version: By pressing "4" you can load a binary file from disk. (See last paragraph of this page).

Cassette version: Only files which the assembler creates can be loaded by this method as the system understands the files created by the assembler to be data files. See also B(LOAD) of the monitor.

If you, for some reason, should wish to write your own loader, the assembler creates cassette files in the following format:

```
        Starting address    End address
          /LSB / MSB   /   LSB / MSB / CODE - - /
```

You cannot look for the end of file marker as it will undoubtedly often occur in your code.

By pressing "4" you will then receive the prompt "LOAD?" and you should type in the same name you gave the assembler. The MENU appends a ".B", searches for the file and loads it when found. You will then be asked if you wish to execute the code. Type "N" and you will be back in the MENU. Type "Y" and the system will begin execution at the first byte of your code.

Press "5"

By pressing "5" you will exit to the BASIC level, MENU.B will still be in memory. If you did not wish to exit, simply run it again.

Configuration note: You may configure the MENU for the type of printer interface you have attached to your computer. It is at present set up for the Commodore printer (device 4). If this is satisfactory, or you do not have a printer, then you may skip this section.

If you are still here and have a serial printer, then follow these steps:

LOAD the MENU and RUN it.

Press 3 to load and execute the monitor.

CBM 64

Type C 81Ø 2,W,X,Y,Z[1] <CR>
S MENU 8ØØ C68<CR> (Disk version)
S MENU 8ØØ B36<CR> (Cassette version)

VIC 20

Type C 121Ø 2,W,X,Y,Z[1] <CR>
S MENU 12ØØ 1473<CR> (Disk version)
S MENU 12ØØ 14B7<CR> (Cassette version)

NOTE: If your MENU is configured for device 2 (RS 232 port) handshaking will be taken care of by this software, unless Z above is Ø.

NOTE: The MENU must be run at least once in order for the printer to function correctly.

---

1. First entry device number (2 for serial interface, 4 for parallel), second entry baud rate if applicable (Ø if not). Third entry is character printer expects as the top of form character ($D if none). Fourth entry is number of lines printed before the top of form is sent (used only by the assembler, be certain the form length is correctly set on your printer, if applicable). Fifth entry is a handshaking flag (Ø=NO, 1=YES). If you have the printer output sent through the RS 232 port and do not want the handshaking taken care of by this software, Z should be a Ø. W,X,Y and Z must be numbers.

Press "6"

By pressing "6" you can alter the colors you will be using during editing and assembling. Press the numbers between the parentheses as the prompt directs you. First you select what you want changed; characters, border or background. Next you select the color, all Commodore colors are possible. No carriage returns are needed.

You will then be taken back to change another color. If finished, press a "4".

Many color combinations are illegible on a t.v. screen, so you will have to experiment.

NOTE: Neither white background nor white characters are recommended as these colors have special meaning to the replace function of the editor.

NOTE: If you should get an "ILLEGAL QUANTITY ERROR" when trying to LOAD the MENU after an assemble, simply move the cursor back over the LOAD"MENU.B",8,1 line and hit <RETURN>. It will load this time.

II EDITOR

II. TEXT EDITOR

File: TED.B

Location: $AØØØ

Function: Full function line orientated text editor. By using TED.B, the programmer creates the text file program which the assembler will later process into machine code.

Type of program: Machine Language.

How to execute: Type 1 from the MENU or load TED.B, SYS the appropriate location (VIC 20 only), 'E' at the end of the assemble or 'E' from the monitor (CBM64 only)

General description: This text editor offers the user the following functions:

Add line(s) to file

Copy line(s)

Delete line(s) from file

Load a text file from cassette or disk

Save a text file to cassette or disk

Look at the directory (disk version only)

Find a string in a file

Replace a string with another string in a file

Edit (a) line(s)

Obtain information about memory usage

Convert a decimal number to hexadecimal or vice-versa

List all or part of the file

Start afresh with a clear buffer

Merge program files

Where applicable, the general format for instructions is very similar to that used by the BASIC editor. It is command (<line number1>-<line number2>).

A space after the command is required. We will use list as an example.

Command	Action
LIST	list entire file
LIST 5-1Ø	list lines 5 thru 1Ø
LIST -1Ø	list lines 1 thru 1Ø
LIST 1Ø-	list lines from 1Ø to the last line

A space must follow the command and the line number. This does not apply to such commands as DIR, SAVE, and LOAD. Please see detailed description below.

Pressing S will halt the listing.

Pressing C will terminate the listing.

Detailed description: Upon executing TED.B you will see some information. By hitting return you will get to the top level of the text editor. At the top of the screen you will see a help line, these are the commands which you have at your disposal.

On the following pages, in alphabetical order, is a detailed description of the commands available to you. Most commands have an abbreviation.

NOTE: (CBM64 Disk Only) Commands ADD, DIR, EDIT, FIND, LIST and REPLACE will not give the help line after completion, but will leave its results on the screen with the flashing cursor waiting for a command. You can enter a command or hit <return> to clear the screen and get the help line.

Command: ADD (<ln1>-<ln2>)

Function: Used to add line(s) of text to a file (begins adding after ln1).

Abbreviation: A

Example: ADD 1Ø
A 22-4Ø

By typing just ADD, you will begin adding after the last line. Assume the file has 100 lines, if you type either ADD<CR> or A<CR> you will begin adding at line 1Ø1. A 1Ø1 will be displayed at the left of the screen.

You have all of the editing features under BASIC edit such as left arrow, right arrow, up arrow, down arrow, insert, delete and, of course, typing a character.

You have a maximum of 86 characters on the VIC and 78 on the Commodore. The editor will not let you go above and beyond that. For instance, if you are at character Ø, a left arrow will do nothing. If you are at character 86 you cannot add anymore. Please see EDIT (pg. 12) for a more detailed description.

Hitting RETURN without adding any characters in a line terminates the add.

Error condition: If the buffer should become full, you will receive a "BUFFER FULL" message and the last line typed will not be added.

Command: ASSM

Function: Will load and execute the assembler without going through the MENU.

Abbreviation: AS

Description: By typing AS or ASSM you will be asked if you wish to load the assembler. If you type an 'N', you will be taken back to the top of the editor. If you type 'Y', the system will try to load, then execute the assembler. Be certain the assembler is available, but be more certain you have saved the latest version of your text file; you will lose it if you have not.

This command's main function is for those of you with only a cassette. It saves you having to load the MENU an extra time. Those of you with a disk drive may, however, find it useful.

Command: CON <$HEXnumber>

Function: Convert number

Abbreviation: <none>

Example: CON $1ØØØ
CON 4Ø96

Used to determine the hexadecimal value of a decimal number, or to determine the decimal value of a hexadecimal number. Can be very useful for interfacing between BASIC and assembly code. If the number is preceeded by a dollar sign, it is a hexadecimal number, otherwise it is assumed to be decimal.

User: CON $FFFF
Computer: $FFFF=65535

Hit RETURN to proceed

By hitting RETURN you will be at the top level of the editor again.

Error conditions: No error checking is done. The user must be careful he/she is typing in only legal numbers.

Command: COPY (<ln1>-<ln2>)

Function: Copy lines

Abbreviation: CO

With this function you can copy line(s) from one part of the file to another. Does not delete any lines.

Example: CO 5-1Ø

```
Computer: TO?
User:     2Ø
```

Lines 5 through 1Ø will be copied after line 19.

NOTE: Line(s) copied before line specified.

NOTE: No lines are deleted. If you wish to move lines, you must use a copy and delete.

Error conditions: Empty response. Will return to top level of editor.

Error conditions: If the editor was unable to copy lines because of an illegal input, you will receive no error message. It is the user's responsibility to make certain the COPY was performed.

Command: DEL (<ln1>-<ln2>)

Function: Delete lines

Abbreviation: D

Example: DEL 1Ø-2Ø Deletes lines 1Ø through 2Ø
D 3Ø- Deletes lines 3Ø to the last line

Used to remove the specified lines from your buffer area. The user must be very careful when using this command, as there is no protection against removing lines which need to be retained, and, once removed, cannot be recovered (unless, of course, the file had been previously saved).

User: DEL 1Ø
Computer: LINES 1Ø-1Ø DELETED
HIT RETURN TO PROCEED

By hitting <RETURN> you will be at the top level of the editor.

Error conditions: If the user specifies a line number greater than the number of the last line, lines will only be deleted to the last line.

Command: DIR (disk version only)

Function: List directory

Abbreviation: <none>

Example: DIR will display the directory of the disk

Type: DIR

The screen will clear, then display the file name followed by its file type, ignoring all but one consecutive space and file sizes. If the directory is large enough, the screen will fill then halt with the disk continuing to turn. By hitting any key you can continue the display. After the directory is finished, hitting RETURN will return you to the top of the editor.

Error conditions: None.

Command: EDIT

Function: Used to alter a line, i.e. add characters, delete characters, etc.

Abbreviation: E

Example: EDIT 10-2Ø
E 22

With this function, you have all of the editing features as in BASIC.

Type: EDIT

Result: Entire file will be displayed, one line at a time, from line one until the last line. The line number will be displayed to the left with the cursor over the first character of the line. Hitting RETURN will take you to the next line if one has been specified for editing. The double quote (") is used to terminate the editing process prematurely, with any previous alterations to the line being ignored.

← left arrow moves the cursor to the left as far as character #1.

→ right arrow moves the cursor to the right one character at a time, up to one past the last character entered.

↓ down arrow moves the cursor down one line to a maximum position of one past the last character.

↑ up arrow moves the cursor up one line to a minimum position of character #1. During EDIT only, up arrow, if at the top of a line, can be used to go to the next previous line. The screen will clear, placing the previous line at the top of the screen, the cursor at the first character.

" double quote cannot be entered as a character, but tells the editor the editing process is finished leaving the last line being edited unchanged.

<DEL> deletes the character previous to where the cursor is sitting up to and including the first character.

<INST> inserts spaces into a line just after where the cursor is sitting to a maximum of 86 characters for the VIC 20 or 78 characters for the Commodore 64.

<ANY DISPLAYABLE CHARACTER> will overwrite any character over which the cursor is sitting or add characters past the line up to a maximum of 86 characters for the VIC 20 or 78 for the Commodore 64.

<RETURN> will display the next line to be edited below the line which was previously edited, displaying the line number to the left and placing the cursor over the first character of the line. If the line you are editing is the last line, or the last line specified, you will be returned to the top level of the editor.

NOTE: Edit can be used as a good method to scrool through a file, seeing one line at a time.

NOTE: Any changes made during edit are not accepted until RETURN has been pressed.

Error conditions: None.

Command: EXIT

Function: To exit the editor and load and execute the MENU

Abbreviation: EX

Example:

```
User:      EX
Computer:  EXIT(Y/N)?
User:      Y
```

Computer will try to load the MENU from disk or cassette (whichever is applicable). Before using this command, be certain the MENU is available.

By typing "N" after the computer response EXIT(Y/N)? the user will return to the top level of the editor.

Command: FIND (<ln1>-<ln2>)

Function: Used to find the occurrence of any string within a file.

Abbreviation: F

Example:

User: F 1Ø-5Ø
Computer: FIND?
User: ANY STRING
Computer: 5Ø THIS IS ANY STRING WITHIN A LINE

Type: FIND

The computer will ask you for the string you want found. In this case it will search the entire file, line by line. Where it finds a match, it displays the line, with the matched string displayed in reverse.

Hitting an "S" during the process will interrupt the search.
Hitting a "C" will terminate the process.

NOTE: No delimiters are placed around "ANY STRING", any character which can be displayed (including space) can be searched for.

If you do not want to use this command, hit "RETURN" when the computer asks "FIND?"

Error condition: If no match is found, nothing will be displayed. The flashing cursor will tell you the search is finished. You may now enter a new command.

Command: LIST (<ln1>-<ln2>)

Function: Used to display the file on the screen.

Abbreviation: L

Example: L -1Ø

Will list lines 1 through 1Ø on the screen.
On the left is the line number followed by the line.
The line is displayed in assembly language format, i.e.
label field, op code field, as an assemble listing
would do.

Pressing an "S" will interrupt the listing.
Pressing a "C" will terminate the listing.
Pressing RETURN will return you to the top level of the
editor.

Command: LOAD <file name>

Function: Will load a previously saved file into the edit buffer for further editing.

Abbreviation: LO

Example: LOAD FILE1

NOTE: No quotation marks around file name and no device number following. If you have the cassette version, the file will be loaded from cassette, if you have the disk version, the file will be loaded from disk.

NOTE: Be careful here, the old file will be destroyed.

Type: LOAD <the file you want loaded>

Disk version: Will search the disk for the file. If no file exists, you will receive a "FILE NOT FOUND" message. Hitting return will take you to the top of the editor.

Cassette version: Will continue to search until it finds the file. You must be careful to type the file name in correctly, as there is no error recovery.

After the input is accepted, you will see the screen clear and the message "LOADING <file>" until the file is finished being loaded.

After it is loaded, you may edit it as if you had entered it in directly, as you will be brought automatically to the top level of the editor.

Error condition: If no file name is specified after the load command, the user will be returned to the top level of the editor, the old file will be unchanged.

Command: MEM

Function: Display memory and file information.

Abbreviation: M

With this function you can obtain information about the file you are editing. You will receive information about number of bytes used, number of bytes remaining, number of lines defined, and, if applicable, name of file you are editing.

Type: MEM

Computer: Clears screen
Displays number of bytes used
Displays number of bytes unused
Displays number of lines defined
If file has been loaded or saved, displays file name.

Error conditions: None.

Command: MERGE <file name> (CBM 64 Disk Only)

Function: Same as LOAD except the old file is not destroyed. Instead, the new file is loaded on top of the old file. The default file name is that of the old file. You cannot merge a file unless a file has been previously loaded.

Abbreviation: <None>

Example: MERGE FILE2<CR>

Error condition: If the text buffer should overflow, you will receive a "BUFFER OVERFLOW" message and a NEW will be executed.

Command: NEW

Function: Initialize work buffer.

Abbreviation: <None>

Example:

```
User:     NEW
Computer: CLEAR BUFFER(Y/N)?
User:     Y
```

All information in file is erased.

NOTE: Be careful with this command. It could be very easy to undo a lot of work.

Error conditions: "N" typed after computer response, user returned to top level of editor with file left intact. Only "Y" or "N" will be accepted input.

Command: OS (C64 Disk Version Only)

Function: Send string to the operating system.

Abbreviation: <None>

Example:

```
User:     OS<CR>
Computer: What string to OS(I,V,SØ:<Filename>)?
User:     RØ:FILE1=FILE2<CR>
```

FILE2 on the disk will be renamed to FILE1.

Any string which can be included with a PRINT#15 statement from BASIC can be used here, but be careful of the NEW command (NØ:); you will destroy the information on the disk.

Command: PRINT

Function: Sends all or part of a listing to the printer.

Abbreviation: P

Example: P 1Ø-2Ø

Will send lines 1Ø through 2Ø to the printer in assembly language format.

NOTE: If the editor has been configured[1] for a serial interface, handshaking will be taken care of by the editor. You will probably notice some characters at the top of the screen changing. This is nothing to worry about, but may rather be of interest to you, as you can see the process as the handshaking is taking place.

NOTE: You may have to hit <RETURN> after the print out in order to clear the command from the screen.

---

1. See section on configuring the MENU in the description of the MENU.B

Command: REP (<ln1>-<ln2>)

Function: Replace a string with another specified string.

Abbreviation: R

Example:

```
User:     R
Computer: REPLACE?
User:     THIS STRING
Computer: WITH?
User:     THAT STRING
Computer: AUTO(Y/N)?
User:     Y
```

NOTE: No delimiters needed around specified strings, spaces can also be specified.

Computer will search the entire or partial file, replacing every instance of 'THIS STRING' with 'THAT STRING' displaying each string as it is changed.

If auto is not stipulated, when found, the line containing the string will be displayed in red (white for CBM 64) with the search string in reverse. The computer next waits for a Y or N response. If Y, the line will be written over in the original color with the change made. If N, no action. In either case, the search will continue.

Pressing an "S" will interrupt the search.
Pressing a "C" will halt it.

Error conditions: If no string found, nothing will be displayed. If a string should be expanded past about 1ØØ characters, the search will be stopped and any further attempts to replace on this line will terminate the search.

Command: SAVE

Function: Saves a text file to tape or disk.

Abbreviation: S

Example:

```
User:     S
Computer: SAVE AS <FILE NAME>(Y/N)?
```

Computer saves program under old file name, old information on disk or cassette destroyed.

This command has nothing following it on the command line. The user simply types SAVE or S.

If this file had been either previously loaded or saved, the computer will ask if you want it saved under the same name. A "Y" here will automatically save the file and return you to the top level.

An "N" typed here will give the response "SAVE?". The computer then waits for a file name <CR>. The file will be saved as this file name, destroying the old file, if any, and taking you to the top level of the editor.

Error conditions: If the computer needs a file name from you (PROMPT "SAVE?") by hitting only return, you will be returned to the top level of the editor with no action taken.

III ASSEMBLER

III. THE ASSEMBLER

File: ASSM.B

Location: $AØØØ (CBM 64) $598Ø (VIC 20)

Function: Full featured assembler.

Type of Program: Machine Code

General description: ASSM.B is a full assembler, which reads as input a text file created under the text editor, and writes as output a binary file which can be loaded and executed by the computer. By full assembler is meant that the assembler has the capacity to accept and use labels.

In addition, the assembler can send listings to the screen or a printer, chain files, to get a symbol table to dump to screen or printer and much more (please see section on pseudo operation codes for features).

The Micol Systems' Assembler is a two pass assembler. During pass 1 the source code is read from disk or tape and the symbol table is built. That means all labels are given an address and stored in the computer's memory. During pass 1 "1"s[1] are printed on the screen. During pass 2 code is generated. During this pass you will see either "2"s[1] on the screen or the text lines and hexadecimal code displayed. After pass 2 is finished, the symbol table will be listed (if LST or PRI is in effect).

During pass 2 the processing may be interrupted by pressing the 'S' key. Pressing any key again will continue the assembly. Pressing a 'C' will terminate the assembly (useful if you only want a symbol table dump, but be certain LST or PRI is in effect).

---

1. Represents 2Ø lines processed.

As was stated earlier, the assembler accepts text files created by the text editor. The assembler reads the text file line by line into a buffer. There are two types of lines the assembler can correctly accept:

Comment lines

65Ø2/651Ø assembly code lines

Comment lines

Any line which begins with a semi-colon(;) in column one will be assumed to be a comment. No code is generated from such a line.

65Ø2/651Ø Assembly code line

Consists of a maximum of four fields:

1. Label
2. Op Code
3. Address
4. Comment

1. Label field

Must begin in column one.
Must begin with a letter of the alphabet.
May then contain alphanumeric characters.
May be any length.
Labels are optional.
Must be a space between it and the op code field.
An op code must follow the label otherwise an error arises.
Labels should not be an A,X,Y,op code or pseudo op.

2. Op code field

Must follow the lable with a space between these fields. If no label is used, the op code should begin in column two. All op codes are three characters.
The Micol Assembler accepts all legitimate 65Ø2/651Ø op codes. These are the operation codes for the 65Ø2/651Ø which cause the CPU to perform an action. Examples are LDA, DEY, SEC, PLA. Please see any good manual on 65Ø2 assembly language.

Pseudo op codes

These are used as instructions to the assembler. Efficient use of these can make the coder's job much easier.

The pseudo operation codes recognized by the Micol Assembler are:

BYT

Example: LABEL BYT 1,1,$D,$FF,NUMBER

Causes the assembler to generate byte values for each of the numbers appearing after the op code. The only limit to the amount of numbers is the length of the line. The numbers may be expressed in decimal, hexadecimal, octal, binary notation, or a LABEL whose value is less than $1ØØ.

Error condition: If any value is greater than $FF (one byte maximum value), an error condition will be flagged.

NOTE: There can be no spaces before or after the commas. Anything after a space will be considered a comment.

CHN

Example: CHN FILENAME

Causes the assembler to read FILENAME as if it were physically sitting after the code (i.e. chaining).

Disk version: This command is mostly transparent to the user. It is important that all the files be contained on the same disk. It is further recommended that the initialize command (I) be used on the disk before assembling when using several disks, as we have had files written over by other files. This is apparently due to a bug in the operation system when two disks are used which have the same I.D. For this purpose it is highly recommended that a backup be kept of the source disk.

The assembler creates only one object file (if the ORG pseudo op is used). That file has the name of the first file (the name you typed in when the assembler asked you the file name) with a .B appended (can be overridden by typing a ','<filename> following the source code name).

Cassette version: Requires a little more work. With the datasette, only one file may be open at a time. Ideally, we need at least two, one for reading the source file, the other for writing the object file (if the ORG pseudo op is in effect). We, unfortunately, cannot do that. What we can do is this:

(i) During pass one we read each file as it is chained. The computer will pause and let you know which file it wants to read. Put in the proper tape and hit return, but be certain the tape sits before the beginning of the file.

(ii) At the beginning of pass 2 (the screen will clear and you will see "PASS 2" at the top) the computer will pause and ask you for the first file. This would be a good time to rewind all of your tapes. Put in the first (rewound) tape, and hit return. The assembler will then assemble the file, writing the object code to a buffer area. If the ORG pseudo op is in effect, at the end of each file you will be asked for the object tape. Place it in the datasette. It will open a file with a ".B" appended to the name of the source code. You will get one object file for every source file read. Only the MENU or "B" from the monitor can load one of these files, but "S" of the monitor can be used to save it as a file, which can be loaded from the keyboard. If the PRG pseudo op is in effect, the code will not be saved to tape but will be written to memory, so you will not need an object tape.

NOTE: You will have to load all of the chained object files created by the assembler using the MENU or monitor to have a complete program. You will have to note all the source file names, as you will need them all to have a complete program.

Error condition: Cassette version only. See Appendix B. During the assembly process, your code will be written to one of these buffers. If the buffer size is exceeded, the assembly will be aborted. If you believe your code will exceed this buffer, then you could chain your files and send the binary code to tape.

CHN should be the last line in the code. CHN terminates a file.

EJT

Used to cause a page eject (top of form). Useful only if your printer has a top of form character. See information about configuring the MENU in Chapter I.

ELS

NOTE: This command requires no operand.

This pseudo op should only be used if the IFF pseudo op is still in effect. This statement will evaluate to the opposite of the IFF statement and code will or will not accordingly be generated. For example, if the IFF statement evaluates to greater than Ø, the code will be normally assembled until the ELS statement. From there to the STP or end of code, the assembler will generate no code.

An ELS statement not coupled with an IFF will turn the code generation off.

EQU

Example: LABEL EQU $1234

Assign the address to the right of the EQU to the label to the left. It equates them.

NOTE: Although it is not an error, an EQU statement without a label is useless.

Error condition: It is recommended you place the EQU statements at the beginning of your program. They may be placed anywhere in the code; however, if one with a value of less than $1ØØ is declared after it is first referenced, subsequent addresses of labels will probably be wrong (during pass one, the assembler assumes 3 byte operations for as yet unknown addresses).

Error condition: If a label exists as part of the address field, it must have been previously defined, otherwise you will get an unknown LABEL error during pass 1.

EXP <label> <parameter1>,<parameter2>,etc.[1]

Used when you want a previously defined macro expanded at the line specified. The parameters will be included in the order they are defined.

See the description of the MAC pseudo op for an example and more information.

NOTE: Total number of characters involved in a macro expansion should not exceed the buffer area. The assemble will be aborted with the appropriate message if exceeded. (See Appendix C).

IFF <address>

Used in conditional assembles. There must be an operand field in this statement. If, during assembly time, the operand has a value of Ø, the statements following it to the STP statement, ELS statement or end of code (whichever is first) will not be processed. If the operand is not equal to zero, the code will be processed as if the IFF statement were not there. You must be certain the operand can be properly evaluated at the time the statement is encountered during pass one, otherwise probable errors result.

This statement is useful if you wish different code generated at different times.

```
Example:        COND EQU Ø
                     <code>
                     IFF COND
                     <code>
                     ELS
                     <code>
                     STP
```

By changing the EQU statement, you can cause the second set of code to be generated instead of the first.

1. All versions except 13K VIC

LST

Causes an assembled listing to be sent to the CRT. Default is LST. Implies an NPR pseudo op.

MAC <label>[1]

In order to define a macro for later expansion, one uses the MAC pseudo op followed by a label. This label is the one which will be used when the macro is expanded later in the code. It is important the macro be defined before it is used, otherwise it is an error.

Parameters may be specified within the macro definition by placing a question mark followed by a digit or letter of the alphabet, a '1' representing a one, a '9' representing a nine, an 'A' representing the tenth parameter and a 'Z' representing the thirty-sixth parameter as if it were a base 37 system.

A TMC pseudo op terminates the macro and must not be left off, else the macro storage area will certainly overflow.

Parameters can be extremely useful. The parameter is marked within the macro definition by use of the question mark (?) followed by some digit or letter. Later, with the EXP pseudo op, the parameters will be defined. The parameter can have a different value for each EXP statement.

Within the macro definition the parameters may be marked anywhere as anything, even labels. Then, when the macro is expanded, they will, like magic, be changed to their definition. If the parameter is undefined, it will simply not be expanded with an error message. We invite you to experiment with this feature. It can make your programming much more enjoyable.

---

1. All versions except 13K VIC

Macros may be nested by using the EXP <label> statement within a macro definition. In theory, at least, there is no limit to the amount of nesting.

```
Example:    MAC EXAMPLE
            LDA ?1
            STA ?3          comment will be included
            JSR ?2          when used
            TMC
            EXP EXAMPLE FIRST,SECOND,THIRD
```

This will expand to:

```
            LDA FIRST
            STA THIRD       comment will be included
            JSR SECOND      when used
```

NOTE: When the macro is defined, it is written to a buffer area described in Appendix C. If this buffer should be exceeded, the assembly will be aborted with the appropriate message.

Macros are physically expanded within the source code buffer. If that buffer should be exceeded, the assemble will abort with an appropriate message. In practice this should never happen.

During a normal assemble, the system reads the source code into a sizable buffer area. If the buffer is too small, the system will simply process the source code until it is finished, then finish reading the rest of the code into the buffer. There is a small area (2K on the CBM64) allowed for macro expansion. If this is exceeded the system will also be aborted as above.

NLT

Causes the listing not to be sent to the screen. Speeds up the assembly process considerably. Only "2"s for pass 2 will be displayed on the CRT.

NPR

Causes an assembled listing not to be sent to the printer. Useful to send only that part to the printer which you need to see.

ORG <address>

Used to set the program counter of the assembler and send the binary code to disk or tape. The next statement assembled will have the address specified by the ORG statement.

There can be only one ORG statement in a program, and there should be only one. If more than one, the others will be ignored by the assembler. If a PRG pseudo op has been previously defined, this statement will also be ignored.

PRG <address>

Same as the ORG statement but assembles to memory instead of to disk or tape.

The object code is written to a buffer area (see Appendix B), then, when the assemble is complete, the code is moved to the area specified by address. That means there should be no conflict between your generated code and the system software. Simply do not specify an address such that your code will overwrite the assembler or the buffer area ($AØØØ-$FFFF).

If the buffer should overflow, the assemble will abort with an appropriate message.

NOTE: For code to be generated, there must be either an ORG or PRG declared within the program.

PRI

Send an assembled listing to the printer. If the last statement of the program is a PRI, will only dump the symbol table and possible error messages to the printer. Implies an NLT pseudo op.

NOTE: See section about configuring the MENU in chapter one.

RES <number>

Reserves number bytes. Actually generates number NOPs ($EA) so be careful, large numbers will generate large amounts of code. Number may be in any notation, even a label.

Example: Label RES 5

NOTE: Very useful for defining variable locations.

STP

Used to terminate an IFF statement.

NOTE: There are no nested conditional statements. This statement will terminate all unresolved conditional statements in effect.

This statement should be used if the IFF statement is used.

STR 'ANY STRING'

Causes the assembler to generate one ASCII byte for each character between the quotes. This command is extremely useful for sending output to the screen.

Example:

```
CHROUT    EQU $FFD2
          PRG $2ØØØ
          JMP PROG  NOTE: BEGIN PROGRAM EXECUTION
OUTPUT    STR 'THIS IS A STRING'
          BYT $D
PROG      LDY #Ø
PROG1     LDA OUTPUT,Y
          JSR CHROUT
          INY
          CMP #$D
          BNE PROG1
          JMP PROG
```

This program, when assembled and executed, will print 'THIS IS A STRING' continuously until you press RUN/ STOP and RESTORE simultaneously.

NOTE: The string following STR must be enclosed in single quotation marks (').

TMC[1]

Used to terminate a macro definition. Do not leave this statement out, it is extremely important. Without it, the file from the MAC statement to the end will be considered a macro.

WOR <values>

Generates two bytes of code based on value in 65Ø2/651Ø addressing format (LSB,MSB). If value is less than $1ØØ then MSB will be set to zero. Value may be in hex, binary, octal, decimal, or a LABEL.

Useful in creating jump tables.

Example:

```
LABEL WOR $FFFD,$FF,$1234,Ø
```

The assembler will generate the following hex code:

```
FD FF FF Ø 34 12 Ø Ø
```

NOTE: There may not be a space before or after a comma. Anything following a space will be considered a comment.

NOTE: The assembler looks for an END<CR> or EEND<CR> which the editor creates when a file is saved. Unless you wish to terminate a file prematurely, avoid these terms.

NOTE: The LST,NLT,NPR and PRI pseudo ops may be implemented from the keyboard by typing L,N,Q or P respectively, but do not abuse it.

---

1. All versions except VIC 13K

3. Address field

There must be at least one space between the op code and the address field. The 65Ø2/651Ø uses eleven possible modes of addressing. The addressing mode and syntax accepted by the assembler are:

Accumulator addressing

These are instructions which operate only on the accumulator. There are only four of these, they are:

```
ASL A
LSR A
ROL A
ROR A
```

Immediate addressing

Takes the byte following the operation and performs its operation on it.

Example:

```
LDA #$AØ
LDX #<LABEL
```

NOTE: The '#' is used to specify immediate addressing.

Absolute addressing

Uses the two bytes following the operation as its address.

NOTE: The address must always be in memory in 65Ø2/651Ø addressing format, that is: least significant byte, most significant byte. However, the assembler will take care of the order, so this should be transparent to the user.

```
LDA $FFØØ
LDY LABEL     Label is not at zero page
JMP $FFD2
```

Zero page

The address specified by the byte following the operation is assumed to be between $Ø and $FF (i.e. zero page).

Example:

```
LDA $FØ
LDA P1    Where P1 has an address less than 256
```

NOTE: Note the difference between zero page and immediate addressing. After the instruction LDA #$FØ is executed, the accumulator will contain an $FØ. After the instruction LDA $FØ is executed, the accumulator and address $FØ will have the same value.

Indexed zero page addressing

Adds the contents of the specified register and the byte following the operation code to get the address upon which to operate.

Examples:

```
LDA $FØ,X
LDX $FØ,Y
```

Indexed absolute addressing

Same as above, but uses the two bytes following the operation code (in 65Ø2/651Ø addressing mode).

Examples:

```
LDA $ABCD,X
LDX $ABCD,Y
```

Implied addressing

Always a one byte instruction.

Examples:

```
DEY
PHA
PLA
INX
```

Relative addressing

Used exclusively with conditional branches. Uses the byte following the operation code as a branch offset. If bit 7 of the address byte is set, the branch will be before the operation, if not set, afterwards. The address byte is added to the program counter when the program counter is pointing to the next instruction.

NOTE: If relative branching is used exclusively with labels, relative and absolute addressing appear to be the same.

Examples:

```
SEC
BCS LABEL   and
JMP LABEL
```

will perform the identical operation if LABEL lies not more than about 125 bytes from the instruction. If greater, the assembler will flag it as an error.

Examples:

```
BEQ LABEL
BCS *+5  *stands for program counter at beginning
           of instruction. Can be used to skip
           around a JMP statement.
```

Indexed indirect addressing

Takes the value of the byte following the operation code (i.e. must be zero page) and adds the contents of the X register to it. Using that value as an address, takes the value of that byte and the subsequent byte, and uses that value as the address upon which to perform the operation.

Example:

```
LDA ($FØ,X)
```

Let us say X has a value of two and at location $F2 and $F3 are $11 and $CØ respectively. The CPU will add 2 to $FØ obtaining $F2. It will then fetch the $11 and $CØ obtaining $CØ11 as the address. In this case only, the statement is equivalent to an LDA $CØ11.

Indirect indexed addressing

Using the value of the byte following the operation code as an address (i.e. must be zero page), fetches the value from that specified address and the following byte and adds the Y register to them in order to obtain the address on which to perform the operation.

Example:

```
LDA ($FØ),Y
```

NOTE: There is a subtle but important difference between this addressing mode and the previous one. In the previous mode, the contents of the register were added before the first address was obtained, in this case after.

```
LDA ($FØ),Y  and
LDA ($FØ,X)
```

perform the same operation only if X and Y both contain Ø.

Indirect

PURE INDIRECT ADDRESSING The same as the previous two instructions, except the registers are not used. There is only one instruction using this addressing mode, it is:

```
JMP (ADDR)
```

4. Comment field

Following any completed line of code you may place an optional comment. It will have no influence on the code which is generated. There must be at least one space between the previous field and the comment.

Example:

```
LABEL DEY  THIS IS A COMMENT
      LDA ($FF),Y THIS IS ANOTHER COMMENT
      PHP
```

Values with the addressing field

May consist of:

Zero page label
Absolute label
Zero page address
Absolute address
Special characters consisting of:

* program counter at beginning of instruction
> least significant byte designator
< most significant byte designator
$ denotes the following number is hexadecimal
@ denotes the following number is octal
% denotes the following number is binary
no symbol defaults to decimal
+ add the following number to the previous result
- subtract the following number from the previous result
specifies immediate addressing
' instructs the assembler to take the ASCII code of the next character.

Examples:

```
BEQ *+5
LDA #>ADDR
LDY LABEL+1
JMP $FFD2
AND #%111ØØØØØ
LDA #'A'
```

NOTE: No spaces allowed between characters within this field. A space will denote the next character(s) as being a comment.

NOTE: No precedence is observed. All operations done from left to right.

Error Messages

None of us is perfect and you will undoubtedly make mistakes. These errors will be flagged as they are found. In addition, a summary of the errors will appear after the symbol table dump (VIC 20 13K version excepted).

The error messages differ somewhat between the VIC 20 and Commodore 64 versions, so these may not be an exact wording.

"ADDR ERROR AT <line number> (in file <filename>)"
The address field of the specified line is in error.

"DD ERROR AT <line number> (in file <filename>)"
You have declared the same label a second time at the specified line number (appears during pass one).

"OP CODE ERROR AT <line number> (in file <filename>)"
That, which was used on the op code field, was not a recognized 65Ø2/651Ø op code or Micol Assembler pseudo op code.

"?LABEL ERROR AT <line number> (in file <filename>)"
You have referenced a label which has never been declared.

"BR OVERFLOW ERROR AT <line number> (in file <filename>)"
Can have two causes:
a) The label to which you were branching has not been declared.
b) You have tried to branch more bytes than is allowed (about 126 bytes).

"S.T. OVERFLOW AT <line number>"
You have declared more labels than there is memory to store it.

Fatal error: "ABORT,HIT KEY,BUFFER OVERFLOW AT<line number>" If the source code buffer during a macro expansion, the macro buffer during a macro definition or the memory buffer as the result of a PRG statement or ORG statement in the cassette version should overflow, the assembly process will simply be aborted with the appropriate message, "HIT ANY KEY TO CONTINUE"

At the finish of the assembly process, after the symbol table dump, the error messages will be displayed again. Pressing an "S" will interrupt the list for you to read. Pressing a "C" will terminate it.

NOTE: The system stops counting or storing errors if greater than 125. Line number is the same as the text editor's line numbers.

NOTE: (VIC 20 13K version only) Because of memory considerations, neither file names nor a summary of errors will be displayed. You will have to note them as they appear.

Symbol table dump

At the conclusion of the assemble, all labels and their addresses will be displayed (if an LST or PRI is in effect). If a label has not been accessed (has no use in the code) it will be in reverse on the screen and have a ">" pointing to its address if on the printer.

The symbol table together with the monitor will probably be your most useful debugging tools.

How to assemble a program

1. You must first have created and saved your program source file using the text editor.

2. Exit normally to the MENU or directly to the assembler after saving your text file.

3. If in the MENU, press 2 to load and execute the assembler.

4. The computer will ask you what file you want assembled. Type the file name of the text file you have just saved with an optional ,<object file name>. Disk version only: the comma is required if you wish to specify a separate file name.

5. The assembler will process your file, generating 65Ø2/ 651Ø machine code to a file if disk, or a buffer, if cassette or the PRG is in effect, saving the code to the name specified above with a ".B" appended (assuming the ORG pseudo op is used in the code).

6. When finished, you will get the messages giving you information about total number of errors followed by number of bytes of code generated and number of lines processed. If you have used the ORG pseudo op and have the disk version, the object file will be closed here. If you have the cassette version and have specified an ORG pseudo op, you will be asked to insert an object tape: any suitable cassette will do, then hit any key (the computer will instruct you).

7. If you have specified the PRG pseudo op in your code, the object code will be moved from its buffer to the area you specified. You will then be asked if you wish to execute the code. Press 'Y' only if there were no errors and the program execution begins at the PRG statement, else type an 'N'.

8. You will then get the symbol table dump followed by the message "ASSM ANOTHER FILE(Y/N/E/M)?". Type 'Y' if you wish to assemble another file, type 'N' if you want the MENU loaded, type 'E' if you wish to go directly to the editor, type 'M' if you want the monitor loaded and executed, but be certain one of those files is available before pressing the appropriate key.

9a. Disk version: If the binary code was written to disk and you wish to load your recently assembled program, you can do it either through the MENU (type "4") or the monitor using the "L" command, or from the keyboard by typing: LOAD "<filename>.B",8,2.

9b. Cassette version: The object files created by the assembler cannot be loaded from the keyboard but can be done by using either the MENU or the monitor. Please see "4" of the MENU or the command B(LOAD) of the monitor for details.

NOTE Cassette version only: If the source code was too large for the source code buffer or you have chained your files, the assembler will have to read your files a second time for pass 2. You will be informed at the beginning of pass 2.

IV MONITOR

IV. THE MONITOR

File: VICTOR.B

Location: $AØØØ

Function: Allows full manipulation of machine code. VICTOR.B contains a disassembler, a hexadecimal dumper, allows you to change code, execute code, move code, save code to disk or cassette, load code from disk or cassette, trace code through execution, set break points and more.

Type of code: Machine language.

How to execute: Type a "3" form the MENU, or load the monitor and SYS the appropriate address (VIC 20 versions only), or press "M" after the assemble is finished.

General description: The monitor allows you all the features you need in your viewing, altering, saving, loading and debugging of machine code writing and can dramatically shorten the time needed to debug your programs.

Syntax:

<ONE LETTER COMMAND>(<SPACE(S)>)(<HEX ADDRESS> or <FILENAME>)<SPACE(S)>
(<HEX ADDRESS>)<SPACE(S)>(<HEX ADDRESS>) <etc.>
<CR>

The numbering system understood by the monitor is hexadecimal (base 16); all input and output is in hexadecimal.

Here is a general overview of the commands and/or features of the monitor.

A - ALTER block moves code from one area of memory to another, altering any absolute addresses which are within the field being moved (relocates).

B - BLOAD Cassette version only: loads the object file created by the assembler.

C - CHANGE change specific areas of memory.

D - DISASSEMBLE turns the machine code into 65Ø2/651Ø assembly like code.

E - EDITOR CBM 64 only: exit to editor.

I - INSPECT dumps memory as characters.

K - CLEAR clears the screen.

L - LOAD loads machine code from either cassette or disk.

M - MOVE same as A, but makes no alterations.

P - PRINT sends the output from the subsequent command(s) to the printer.

Q - QUIT loads and executes the MENU.

S - SAVE saves machine code to either cassette or disk.

T - TRACE follows the program execution step by step showing disassembled code, contents of registers, values of flags and counters.

V - VIEW dumps a screen full of memory in hexadecimal notation.

W - WALK same as T, but only one line at a time done.

X - EXECUTE execute the code from where specified.

You can also set break points (Ø) within your code. When executed, you will get a dump of all registers, flags and counters. A very useful debugging technique. (Use the "C" command to set the break points, then use "X" to execute your code.)

Error conditions: The monitor gives no error messages but does check for errors. If there is an error in input, the monitor simply sends you to its top level with no action taken. If in doubt, check that the required action has been taken.

NOTE: You cannot set the decimal mode (op code SED) prior to using the monitor, it protects itself against it. Also, during a WALK or a TRACE, the decimal mode and break interrupt status flags cannot be altered.

Description of Commands

Command: A(LTER)

Function: Block moves memory from one area of RAM to another, changing all absolute addresses which lie within the field being moved to new relative addresses.

Example: A2ØØØ 28ØØ 1ØØØ<CR>

Let us say you have these lines of (disassembled) code sitting between locations 2ØØØ and 28ØØ

```
LDA $2ØØ5
JSR $25ØØ
JSR $FFD2
```

After the command above is executed, sitting relative to location $1ØØØ as the code was to location $2ØØØ will be:

```
LDA $1ØØ5
JSR $15ØØ
JSR $FFD2
```

NOTE: Only those addresses within $2ØØØ and $28ØØ were modified.

Warning: This command assumes that all the code sitting between the specified locations is executable code. Some literals, for example, it encounters which it misinterprets as op codes, will have their address fields also altered. By moving literal and other fields afterwards with the move command, making certain that there are no 3 byte operations within the literal field and that the beginning of the executable code begins at the correct location (use the disassembler), you should be able to move any program.

Syntax:

A(<SPACE(S)>)<BEGINNING HEX ADDRESS><SPACE(S)><END HEX ADDRESS>
<SPACE(S)><DESTINATION HEX ADDRESS>

NOTE: This is a block move command. That means the destination address (the third address specified) cannot lie within the field being moved.

Command: B(LOAD) (Cassette version only)

Function: Load recently assembled binary files.

Syntax: B(<SPACE(S)>)<filename>

Example: B FILE<CR>

The command is used to load the object code generated by the assembler which cannot be loaded from the keyboard or the L command of the monitor, but can be loaded by the "4" command of the MENU.

By typing the example above, the monitor will append a ".B" to the filename, search the tape and, when found, load the binary file to the address specified by the file header.

Error recovery: If you type a bad file name, try the RUN/STOP key. It usually works, but may cause the system to crash.

NOTE: This command will not load files saved by the S command.

Command: C(HANGE)

Function: To allow the user to change value(s) sitting in specified memory locations.

Syntax: C<ADDRESS><SPACE(S)><HEX VALUE><SPACE(S)> <HEX VALUE><SPACE(S)>(<HEX VALUE>) etc. to 88 characters (8Ø Commodore).

Example: C 5ØØ 1 2 3 4 5 6 7 8 9 A B<CR>

Memory locations 5ØØ through 5ØA will contain the above numbers.

NOTE: (CBM 64 only) You can set the values for Reg A, Reg X, Reg Y, and status register before a WALK or TRACE by typing:

C AØØ3 <Reg A> <Reg X> <Reg Y> <Status Register><CR>

Example: CA ØØ 31 23 Ø<CR>

Command: D(ISASSEMBLE)

Function: To display memory contents as an assembly listing.

Syntax: D(<SPACE(S)>)<ADDRESS>

Example: D 5ØD<CR>

Upon typing the above, the screen will be filled with a listing in the syntax of the Micol Assembler Text File, except that no labels will be displayed, only addresses. The address of the line will be followed by the assembly line. Pressing RETURN will continue the disassemble.

Command: E(DITOR) (CBM 64 Only)

Function: To exit the monitor, load and execute the text editor.

Syntax: E

Example: E<CR>

By typing the above, the monitor will try to load and execute the text editor. Be certain it is on the disk before pressing <CR>.

Command: I(NSPECT)

Function: Dumps a screen full of memory. If the code is between $2Ø and $7F, the character representation will be displayed. Very useful for looking for text within memory.

Syntax: I(<SPACE>)<ADDRESS>

Example: I 5ØØ<CR>

Will display memory beginning at hex location 5ØØ. By pressing RETURN you will continue the display.

Command: K(LEAR)

Function: Clear the screen.

Syntax: K

Example: K<CR>

The screen will be blanked and the cursor will be homed.

Command: L(OAD)

Function: Load a binary file from disk or cassette (whichever applicable).

Cassette version note: The file to be loaded must be program files and not data files which the assembler generates (see MENU and also the monitor commands B(load) and S(ave) for saving these files).

Syntax: L(<SPACE(S)>)<FILENAME>(,<ADDRESS>)

If ",<ADDRESS>" is specified, the file will ignore the file header and load to the specified address. This does not work with the cassette versions.

Example: L FILE<CR>

Will search the disk or cassette (whichever applicable) for FILE.B and load the file to the location it was originally saved at.

NOTE: Both the SAVE command of the monitor and the assembler append a ".B" to any file name you give it; that means you cannot load a binary file that does not end with this.

When you load a file, specify it without the ".B"

Error condition: Disk. If the file cannot be found, the red light will flash on the drive with nothing loaded. Cassette. The system will continue to search until the file is found.

NOTE: (VIC 20 cassette version only). For code with address above $AØØØ, this command or SAVE will not work. B(load) will work, and, therefore, you will have to keep these files as the assembler generated them.

Command: M(OVE)

Function: Block copy memory to another area of memory.

Syntax: M(<SPACE(S)>)<STARTING ADDRESS><SPACE(S)>
<FINAL ADDRESS><SPACE(S)><DESTINATION ADDRESS>

Example: M5ØØ 6ØØ 1ØØØ<CR>

Will move bytes 5ØØ through 6ØØ to location 1ØØØ through 11ØØ.

NOTE: This is a block move command. The destination address may not be contained within the starting and final addresses.

Command: P(RINT)

Function: Send screen output to the printer.

Syntax: P

Example: P<CR>

Will send all screen output from the next command to the printer. Output returned automatically to screen when any new command is issued (i.e. <RETURN> will not return output to the screen).

NOTE: Please see section on configuring the MENU in Chapter I.

Command: Q(UIT)

Function: Exit the monitor. Will try to load the MENU.

Disk version: The MENU must be on the disk in the drive.

Cassette version: You must have a tape with the MENU ready to be loaded.

Syntax: Q

Example: Q<CR>

Computer will try to load and execute the MENU.

Command: S(AVE)

Function: To save memory to disk or cassette (whichever is applicable).

Syntax: S(<SPACES>)<FILE><SPACE(S)><STARTING ADDRESS>
<SPACE(S)><FINAL ADDRESS>

Example: S MYFILE 1ØØØ 2ØØØ<CR>

Will save memory locations 1ØØØ through 2ØØØ to disk or cassette (whatever applicable) as file MYFILE.B

NOTE: A ".B" will be appended to the file name to distinguish it from text files. You should know this, but for the most part its use should be transparent to the user.

Command: T(RACE)

Function: To trace the execution of a machine language program step by step giving information about contents of registers, flags set or unset, and stack pointer.

Syntax: T(<SPACES>)<ADDRESS>

Example: T 1ØØØ<CR>

Displayed will be:

ADDRESS
DISASSEMBLED LINE
CONTENTS OF REGISTERS
FLAG SETTING
STACK POINTER

This information will scroll across the screen. Pressing "S" will interrupt the trace (pressing any key will start it again). Pressing "C" will stop it.

Command V(IEW)

Function: Will hex dump a screen full of memory. The address will be written followed by a colon, followed by the memory dump on each line.

Syntax: V(<SPACES>)<ADDRESS>

Example: V 1ØØØ<CR>

Starting at hex location 1ØØØ, a screen full of memory will be displayed. Hit RETURN to continue the display.

Command: W(ALK)

Function: To step through a program execution. Same as trace, but does one line at a time.

Syntax: W(<SPACES>)<ADDRESS>

Example: W 1ØØØ<CR>

Beginning at memory location 1ØØØ, you can step through the program execution line by line. Hitting RETURN will walk the next line. Pressing "C" will stop the trace.

Command: E(X)ECUTE

Function: To cause the execution of a machine language program from a specified location.

Syntax: X(<SPACES>)<ADDRESS>

Example: X 8ØD<CR>

The computer will begin execution from the address specified.

Error conditions: You cannot start execution from address Ø, this is used as an error check. Also, the system will probably hang if you try to execute unexecutable code.

V COPY

V. THE COPY PROGRAM

COPY.B (Cassette version)

Function: To make backups of system software.

How to use:

Place tape containing file COPY.B into your datasette
Type LOAD "COPY.B",1,1
When loaded type SYS8192<CR>
Follow computer instructions.

You will have to swap tapes between those that contain the system software and the tapes you want them copied to. Copies are made in the order in which they are contained on your master cassette.

COPY.B (Disk version)

Function: To copy text, BASIC and binary files from one disk to another.

How to use:

1. Place a disk containing COPY.B in your drive.
2. Type LOAD "COPY.B",8,2
3. After it is loaded type RUN<CR>
4. The computer will respond with COPY?
5. You type the file name and hit RETURN
6. The computer responds INSERT MAIN DISK and waits.
7. Insert the disk which contains the file you want copied and hit RETURN
8. The computer reads the file into its memory. If the file is too large you will be informed.
9. The computer prints INSERT SAVE DISK and waits.
10. Insert the disk to which you want the file copied and hit RETURN
11. The computer will save this file under the same name on the new disk.
12. The computer responds with COPY? If you wish to finish simply hit RETURN, otherwise start the process again from line 5.

APPENDIX A

Addresses of Software

CBM 64 Disk and Cassette

File name	Location	Approx. length in bytes	X(SYS)
MENU.B	$8ØØ	1.ØK	$8ØD(2Ø61)OR RUN
TED.B	$AØØØ	7.5K	None
ASSM.B	$AØØØ	9.5K	None
VICTOR.B	$AØØØ	5.ØK	None
COPY.B(Disk)	$8ØØ	8ØØ	$8ØD(2Ø61)OR RUN
COPY.B (Cassette)	$2ØØØ	8ØØ	$2ØØØ(8192)

VIC 20 37K Disk Version Only

File name	Location	Approx. length in bytes	X(SYS)
MENU.B	$12ØØ	8ØØ	$8ØD(2Ø61)OR RUN
TED.B	$AØØØ	7.2K	$AØØØ(4Ø96Ø)
ASSM.B	$598Ø	9.5K	$598Ø(22912)
VICTOR.B	$AØØØ	4.5K	$AØØØ(4Ø96Ø)
COPY.B	$12ØØ	8ØØ	$8ØD(2Ø61)OR RUN

VIC 20 37K Cassette, 29K Disk and Cassette

File name	Location	Approx. length in bytes	X(SYS)
MENU.B	$12ØØ	8ØØ	$12ØD(2Ø61)OR RUN
TED.B	$63ØØ	7.2K	$63ØØ(25344)
ASSM.B	$598Ø	9.5K	$598Ø(22912)
VICTOR.B	$6B8Ø	4.5K	$6B8Ø(2752Ø)
COPY.B(Disk)	$12ØØ	8ØØ	$12ØD(4621)OR RUN
COPY.B	$2ØØØ	8ØØ	$2ØØØ(8192)

VIC 20 21K Disk and Cassette

File name	Location	Approx. length in bytes	X(SYS)
MENU.B	$12ØØ	8ØØ	$12ØD(2Ø61)OR RUN
TED.B	$43ØØ	7.2K	$43ØØ(17152)
ASSM.B	$398Ø	9.5K	$398Ø(1472Ø)
VICTOR.B	$4B8Ø	4.5K	$4B8Ø(19328)
COPY.B(Disk)	$12ØØ	8ØØ	$12ØD(4621)OR RUN
COPY.B (Cassette)	$2ØØØ	8ØØ	$2ØØØ(8192)

VIC 20 13K Disk and Cassette

File name	Location	Approx. length in bytes	X(SYS)
MENU.B	$12ØØ	8ØØ	$12ØD(2Ø61)OR RUN
TED.B	$23ØØ	7.2	$23ØØ(896Ø)
ASSM.B	$1E6Ø	8.5	$1E6Ø(7776)
VICTOR.B	$2B8Ø	4.5	$2B8Ø(11136)
COPY.B(Disk)	$12ØØ	8ØØ	$12ØØ(4621)OR RUN
COPY.B (Cassette)	$2ØØØ	8ØØ	$2ØØØ(8192)

APPENDIX B

Object Code Buffer Sizes

Buffer sizes available with the disk version if using the PRG pseudo op and the cassette version for all code generation. Assemble is aborted if the appropriate buffer size is exceeded.

Computer	Buffer size in bytes (approx.)	Address
Commodore 64	8K	$EØØØ-FFFF(Hidden RAM)
VIC 20 37K	6K	Top of RAM($A8ØØ-$BFFF)
VIC 20 29K	4K	Below assembler(See Appendix A)
VIC 20 21K	1.5K	Below assembler(See Appendix A)
VIC 20 13K	5ØØ	Top of RAM(NOTE: Co-resident with RS232 buffer and may cause some print output distortions to serial printers)

APPENDIX C

Macro Buffer Sizes

When you define a macro (pseudo ops MAC and TMC), the computer has these buffer sizes at its disposal to store the definitions. If those are exceeded, the assemble will be aborted.

Computer	Buffer size in bytes(approx.)
Commodore 64	2K
VIC 20 37K	2K
VIC 20 29K	1K
VIC 20 21K	.5K
VIC 20 13K	Not implemented

NOTE: 13K VICs must be careful with their memory usage, as there is not a lot of memory for buffers and symbol table storage. Large files however can still be assembled if the binary code is written to disk or tape, programs are chained (CHN pseudo op) and labels are small and used only when necessary. Other systems will not have this problem.

GLOSSARY

65Ø2 ADDRESSING FORMAT	Two byte addresses specified in least significant byte, most significant byte order.
65Ø2 CHIP	CPU used by the VIC 20
651Ø CHIP	CPU used by the Commodore 64. Both 65Ø2 and 651Ø are virtually identical.
ABSOLUTE ADDRESSING	Refers to specified addresses greater than $FF.
ACCUMULATOR	One of the three registers in the 65Ø2 or 651Ø. Most important of the three, virtually all mathematical calculations are done using the accumulator. See Registers.
ALPHANUMERIC	Usually used to describe strings that can consist of both letters of the alphabet and digits.
ASCII CODE	Standardized code used to represent characters. A decimal 32 in ASCII code is printed as a space. Commodore uses a modified version of ASCII.
ASSEMBLER	A program which can take as input an assembly language text file and translate it into the binary code the computer can execute. It usually gives additional useful information.

ASSEMBLY CODE	A formatted text file an assembler can process into binary code.
ASSEMBLY LANGUAGE	The lowest level of the computer languages. At this level the programmer is directly programming the CPU. An assembler is used to process programs written in assembly language into the binary code the computer understands.
BINARY	A numbering system consisting of only zeros and ones. It has a radix of two (i.e. base 2).
BINARY FILES	Machine language programs saved on either disk or tape.
BIT	Acronym of BInary digiT. A switch in the computer's memory which represents either a Ø or a 1.
BRANCHING	Causes the program to begin execution at another memory location. On this system is always conditional and uses relative addressing.
BREAK POINTS	Used in machine language debugging. When executed will cause the system to dump all registers, flags and counters, and halt execution. The binary code here is a Ø.
BYTE	A collection of bits wired together. In most, but not all cases, a byte consists of 8 bits. Can represent one character.

CHAINING	The process of linking separate text files by the assembler. The assembler can successfully assemble separate text files, as if they were a whole.
CPU	Stands for Central Processing Unit, the "brain" of the computer. When writing in machine language, you are actually programming the CPU directly.
CURSOR	A special character, often blinking, used to show the user where, on the screen, he is entering characters.
COMPILER	A program which accepts as input a text file and creates as output binary code the computer understands. It is different from an assembler in that the text file it reads in is at a higher level of sophistication than what the assembler reads in.
DECIMAL	A numbering system based upon the digits Ø,1,2,3,4,5,6,7,8,9. It has radix 1Ø. The numbering system we use in everyday life.
DIRECT ADDRESSING	Consists of either zero page addressing or absolute addressing.
DISASSEMBLE	A program which takes the binary numbers stored in the computer and translates them into assembly like code.
EDITOR	Same as text editor. A program which allows the user to create, modify and save text files.

FLAG	A memory location which can be set so that later a determination can be made based on that value.
FILE	A collection of data stored in some memory device. This can be the computer's memory, disk or tape. On magnetic media there is usually a file name associated with the file.
HEXADECIMAL	A number system based on the digits Ø,1,2,3,4,5,6,7,8,9,A,B, C,D,E,F. Has radix 16 (i.e. base 16).
IMMEDIATE ADDRESSING	Addressing mode in which the byte following the op code contains the value to be used. "LDA #$FØ" will cause the accumulator to load an $FØ.
INDEXING	Addressing mode in which the contents of the specified register are added to the specified address to get the address which will be used.
INDIRECT ADDRESSING	Addressing mode in which the specified address contains the address which will be used.
INTERPRETER	A program which reads in as data a program, determining what needs to be done and doing it. The BASIC in the ROM of your computer is an interpreter. Remarkable for its convenience and slowness.

JUMP	Causes the program to begin execution at the specified location. Varies from a branch on your system in that it is unconditional, and uses absolute rather than relative addressing.
LABEL	Used in assembly language and some higher level languages to allow the programmer to reference a part of a program. In assembly language, the label will stand for an address in memory.
LOAD	The act of bringing information from some long term storage device, such as disk or cassette, to the computer's memory.
MACHINE CODE	Almost synonymous with assembly code. Usually refers to the binary code which the computer directly executes.
MACRO	In assembly language a segment of text which can later be referenced and thereby inserted as part of the code. Usually accepts parameters.
MEMORY LOCATION	The same as a byte. Can be thought of as a little box in the computer containing a piece of information.
MICOL SYSTEMS	A dynamic software house founded almost simultaneously in Southern California and Ontario, Canada. Dedicated to quality systems software, MICOL is an acronym of MIcro COmputer Language.

MNEMONIC	A collection of characters which help you remember something. "JMP", for example, stands for a hex 4C in machine code and is a mnemonic for it.
MONITOR PROGRAM	Program which interfaces a human with the machine code in his computer.
OCTAL	Numbering system based on the digits Ø1,2,3,4,5,6,7. It has radix 8 (i.e.base 8).
OP CODE	Short for operation code. The second field in a 65Ø2/651Ø assembly line which instructs the CPU what action to take.
OPERAND	The address field following the op code.
PASS 1	In an assembler, the phase in which all addresses are resolved.
PASS 2	In an assembler, the phase in which the code is generated.
PROGRAM	A collection of instructions designed to perform specific actions.
PSEUDO OP CODES	Instructions which resemble operation codes, but are usually designed to instruct the assembler what action to take, or code to generate.
RADIX	The base value of a numbering system. The radix of the decimal system is 1Ø.

REGISTERS

Memory locations within the CPU having special features not found in memory. Without registers your computer would be worthless. In the 65Ø2 and 651Ø CPUs there are three registers, they are:

A accumulator - virtually all mathematics are performed here.

X register - mainly used for indexing.

Y register - mainly used for indexing.

SAVE

The act of storing all or part of a computer's memory to some long term storage device such as disk or tape.

STATUS FLAGS

Bits within the CPU which are set or unset by certain conditions. These are the status flags in the status register: zero, sign, decimal, interrupt, overflow, break and carry. All "decisions" are based upon the status (Ø or 1) of these flags.

STRING

A string of characters. The 'STR' pseudo op is used by the MICOL Assembler to declare strings, e.g. 'THIS IS A STRING'.

ZERO PAGE

The area of memory between locations Ø and 255.

INDEX

```
" 13,17
# 36,43
@ 43
% 43
> 43
< 43
* 39,43
$ 43
+ 43
- 43
' 34,43
```

PRINTED IN CANADA

IMPORTANT

The diskette supplied with this package is meant only as a means of supplying you the software and is not meant as a work diskette. You will need to transfer the disk files to another diskette before program development can begin. The steps to accomplish this are:

1. Initialize a diskette suitable for your work in assembly language (any quality single sided single density or single sided double density soft sectored diskette will do).

2. Using the copy program explained in Chapter V, copy all the files on the master diskette to your work diskette. The files which need to be copied are:

   MENU.B
   ASSM.B
   TED.B
   VICTOR.B

3. Only use the work diskette for your program development and keep the master safe as a backup in case you should crash the work diskette.

Those of you with the cassette version should copy your files using the copy program explained in Chapter V. The copy program sits almost at the very end of the tape. The files are stored in this order: MENU.B, VICTOR.B, TED.B, ASSM.B, COPY.B. After they are copied, put the master cassette in a safe place as a backup.

It is recommended you read the manual before you begin your program development. You may want to read pages 46-47 'How to assemble a program' as an overview before reading from the beginning of the manual.

Also contained on the diskette is an example program called EXAMPLE. You may want to study it before you begin your programming. It can be a great help in acquainting you with this system in particular and assembly language in general.