commodore 64
mastercode assembler

SUNSHINE

commodore 64
mastercode assembler

SUNSHINE

First published 1983 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12-13 Little Newport Street,
London WC2R 3LD
Tel: 01-437 4343

Cover design by Graphic Design Ltd.
Illustration by Stuart Hughes.
Typeset and printed in England by The Leagrave Press Ltd.

Contents

Introduction

Mastercode is a substantial and complex program, providing a host of features of use to anyone interested in machine code programming on the Commodore 64. Despite its complexity Mastercode is designed with the user in mind and is far easier to use than many inferior assembler/disassemblers. All of Mastercode's many facilities are called up by means of clear menus and prompts which appear as the program runs. There are no obscure commands to enter to make full use of Mastercode and with an hour or so's practice you will be making the most of its power.

Even though Mastercode is simple to use, however, careful reading of this manual, before you begin to use the program, will pay dividends. As with all machine code programming, incorrect use of the assembler, or placing of code into reversed areas, can easily 'crash' your system. If that happens nothing will be harmed but you will probably lose any programs you have entered and have to load Mastercode again. Proper preparation will save you a great deal of time and trouble in the long run.

Needless to say, this brief handbook is not intended as an introduction to the subject of 6502/6510 machine code programming. The Mastercode program itself is ideal for those with little knowledge of machine code since its clear presentation makes the variety of tasks associated with entering machine code programs more simple to comprehend. Nevertheless, if you are starting out with machine code we would recommend that you purchase a good book on 6502 programming before attempting to use this or any other assembler.

Mastercode Features

Mastercode will provide you with the following facilities, all of which will be explained in this manual:

1) **The Machine Code Monitor,** including:
a) output of memory to screen or printer
b) modification of memory
c) execution of machine code programs
d) saving of machine code files onto tape or disc
e) loading of machine code files from tape or disc
f) step by step tracing of the execution of a machine code program, including display of register contents.

2) **The Disassembler,** including translation into assembly languages of the contents of any area of memory, whether the 64's ROM or a user program. Output may be sent either to the screen or to a printer.

3) **The File Editor,** including:
a) entry of numbered lines of assembly language instructions
b) listing, individually or in blocks, of previously entered lines
c) deletion, individually or in blocks, of existing lines
d) renumbering of existing lines
e) saving of assembly language files to tape or disc
f) loading of assembly language files from tape or disc

g) addition of a block of memory specified by the user to the user's assembly program.
4) **The Assembler,** allowing the translation of assembly language programs into machine code with full error checking, labelling and a range of assembler directives.

Loading Mastercode

Your Mastercode tape contains two full recordings of all the necessary data, one on each side. To load Mastercode into your 64, simply ensure that your cassette unit is properly connected and that the Mastercode tape is fully rewound, then press SHIFT/RUN and start the tape on command. Because of the size of the Mastercode program and the relative slowness of the 64's cassette loading system, it takes around 11 minutes to fully load the program. During this period the screen will be set first to pink and then, when the final tables of data begin to be loaded, to green. In this latter stage you will see the Mastercode title flashing briefly on and off – this is normal and indicates that loading is proceeding properly.

At the end of the Mastercode program the tape contains another short program for your use, RELOADER. It is not relevant to the initial loading of Mastercode and its use is explained under the section **Location of Code.** If you wish to save RELOADER for regular use, simple leave the tape in the position it was when Mastercode finished loading then, when you have finished with Mastercode, load RELOADER into the memory as a normal BASIC program and save on a tape or disc of your own.

Mastercode Output

When Mastercode is fully loaded it will begin by asking you what input/output devices are connected. The program is capable of dealing with three such devices: a) a cassette unit, b) a disc drive and c) a printer. All such external devices have a 'device number' when they are used by the 64 and Mastercode will treat them as device numbers 1, 8 and 4 respectively. Mastercode will not subsequently allow input from or output to any devices which are not specified as being present at this point.

The Main Menu

Having specified the devices present you will encounter the main program menu, which allows access individually to the functions of the Monitor and to enter the Disassembler, File Editor and Assembler. The functions of the Monitor will be explained individually first:

MAIN MENU 0: Exit to Basic
Entering this option from the main menu quits the Mastercode program. Having left the program all variables are lost and the only way to re-enter Mastercode is to reload the program from tape.

MAIN MENU OPTION 1: Memory Modify

On calling up this function you will be asked to specify a start address in hexadecimal or decimal (all start and end addresses in the Mastercode program may be input in decimal or hexadecimal, with hexadecimal numbers preceded by '$'. Mastercode will then display the address you have specified, the contents of the byte at that address (in hexadecimal) and the four command letters available, namely '+' '−', 'I' and 'E'. Entry of '+' or '−' will result in the byte after or before the current byte being displayed, while entry of 'E' will return you to the main menu. To modify the contents of the byte currently displayed, enter 'I' and then, in response to the prompt 'BYTE' enter the new value (again in decimal or hex) you wish to store at that location. The next byte of memory will then be displayed.

MAIN MENU OPTION 2: Memory Dump

On calling up this function you will be asked to specify your chosen start address and whether you wish to output to a printer (if connected). Mastercode will then display the contents of an area of memory in lines of eight bytes, each preceded by the address of the first byte in the line. On the right of the display will be printed a table showing the locations of any normal characters such as letters or digits. This will allow you to identify the location of any strings which may be held in the area of memory you are examining without having to translate the byte values. Characters with an ASCII value outside the range 32-90 are not printed but represented by a single dot in the table. At the end of each screenful of data you may either continue or return to the main menu.

Given below is a specimen memory dump using this facility:

Table 1: Specimen Memory Dump

```
A19E 54 4F 4F 20 4D 41 4E 59   TOO MANY
A1A6 20 46 49 4C 45 D3 46 49    FILE.FI
A1AE 4C 45 20 4F 50 45 CE 46   LE OPE.F
A1B6 49 4C 45 20 4E 4F 54 20   ILE NOT
A1BE 4F 50 45 CE 46 49 4C 45   OPE.FILE
A1C6 20 4E 4F 54 20 46 4F 55    NOT FOU
A1CE 4E C4 44 45 56 49 43 45   N.DEVICE
A1D6 20 4E 4F 54 20 50 52 45    NOT PRE
A1DE 53 45 4E D4 4E 4F 54 20   SEN.NOT
A1E6 49 4E 50 55 54 20 46 49   INPUT FI
A1EE 4C C5 4E 4F 54 20 4F 55   L.NOT OU
A1F6 54 50 55 54 20 46 49 4C   TPUT FIL
A1FE C5 4D 49 53 53 49 4E 47   .MISSING
A206 20 46 49 4C 45 20 4E 41    FILE NA
A20E 4D C5 49 4C 4C 45 47 41   M.ILLEGA
A216 4C 20 44 45 56 49 43 45   L DEVICE
A21E 20 4E 55 4D 42 45 D2 4E    NUMBE.N
A226 45 58 54 20 57 49 54 48   EXT WITH
```

The area displayed in this table is part of the ROM containing the BASIC error messages and illustrates the use of the display of ASCII characters.

MAIN MENU OPTION 3: Machine Code Execute

If you have entered a machine code program into the memory, either directly or using the Assembler, calling this function will allow you to run it. Such routines should end with the instruction 'RTS' in order to return execution to Mastercode. Care should be taken not to corrupt the 'return stack' during the course of the machine code program or the execution of the main Mastercode may be terminated. It is of course possible to use the stack but any values placed onto it should be removed before execution is returned to Mastercode – this is no more than good programming practice in leaving any machine code routine.

MAIN MENU OPTION 4: Load Machine Code

Using this function a file consisting of the contents of an area of memory previously saved using Mastercode (see next menu function) may be reloaded into memory in their original position. To accomplish this it will be necessary to remember the file name under which it was originally saved. Note that the original contents of the area of memory into which the file is loaded will be lost, so if they are important save them first. This function is particularly useful in developing long machine code programs in parts. Before each section of the program is developed, the program thus far can be reloaded into memory and the new section assembled so as to tag onto the end of it.

MAIN MENU OPTION 5: Save Machine Code

Using this function, any area of memory may be saved to tape or disc for later reloading by Mastercode. You will be asked to specify the output device you wish to use and will not be permitted to output to one which you did not say was present at the beginning of the program. You must also specify the start and finish addresses of the area of memory you wish to save. The main use of this program function will be the saving of machine code programs generated by the Assembler.

If you wish to load machine code files into memory later without using Mastercode, the following lines of BASIC may be used:

```
10 INPUT "FILE NAME"; IN$ (Note: the file name should be terminated with ',S,R'
if you are loading from a disc)
20 OPEN 2, (device number) ,0,IN$: INPUT#2, IN$
30 INPUT#2,SA,EA
40 FOR X = SA TO EA: INPUT#2, T:POKE X,T:NEXT: CLOSE2
```

MAIN MENU OPTION 9: Trace Machine Code

Using this function a machine code program may be 'stepped through' with each instruction being executed only when you press the 'F1' function key. The instruction itself is displayed in a disassembled form, together with the values of the CPU registers before the instruction is executed. Pressing 'F2' (SHIFT/F1) terminates the execution of the machine code routine. Note that faulty code, an incorrect start-point halfway through an instruction or tables of data in the

memory may result in the message 'INVALID OPCODE' being displayed for some instructions.

Trace is a powerful aid in debugging faulty machine code programs. It will even allow you to step through routines in the 64's ROM and examine their functioning. It is especially useful in the case of programs which 'lock up' in a loop. Trace can be used to detect this fault and to break out of the loop simply by pressing F2. Trace will also point out invalid instructions and refuse to execute them.

To use Trace, simply call it up, specify the address in memory at which execution is to begin and whether output is to be to the screen or a printer.

The Trace routine has one main limitation in that it cannot return sensible data for instructions which access addresses in page 1 of memory (100-1FF hex) since this area of memory, mostly taken up by the 'return stack', is used by the Mastercode program in executing the trace routine. The Trace routine uses its own simulated stack held in an array to allow other instructions which affect the stack such as calls to subroutines, pushes and pulls, to be simulated without corrupting the main stack.

Given below is a specimen output of the Trace facility operating on a short machine code routine:

Table 2: Trace of Machine Code Routine

```
A474  A976     LDA #$76
 REGISTER A = 76
 REGISTER X = 00
 REGISTER Y = 00
 REGISTER P = 30
 REGISTER S = FF


A476  A0A3     LDY #$A3
 REGISTER A = 76
 REGISTER X = 00
 REGISTER Y = A3
 REGISTER P = B0
 REGISTER S = FF
```

```
A478  201EAB  JSR $AB1E
 REGISTER A = 76
 REGISTER X = 00
 REGISTER Y = A3
 REGISTER P = B0
 REGISTER S = FD

AB1E  2087B4  JSR $B487
 REGISTER A = 76
 REGISTER X = 00
 REGISTER Y = A3
 REGISTER P = B0
 REGISTER S = FB

B487  A222    LDX #$22
 REGISTER A = 76
 REGISTER X = 22
 REGISTER Y = A3
 REGISTER P = 30
 REGISTER S = FB
```

Note that the steps in the trace follow the jump made in the third instruction to a new location in memory.

This concludes the options available using the Monitor section of the program. The remaining options available on the main menu permit you to enter other program sections.

The Disassembler

The Mastercode Disassembler is capable of providing assembly language translations of all 6502/6510 machine code instructions in the standard format laid down by Mostechnology (now part of the Commodore Semiconductor Group), the designers of the 6502 and 6510 CPU chip. The disassembly includes the address of the instruction, the contents of the bytes involved and the assembly language representation itself. Given below is a specimen output taken from the 64's Basic Interpreter:

Table 3: Specimen Disassembly

```
A474    A976    LDA #$76
A476    A0A3    LDY #$A3
A478    201EAB  JSR $AB1E
A47B    A980    LDA #$80
A47D    2090FF  JSR $FF90
A480    6C0203  JMP ($0302)
A483    2060A5  JSR $A560
A486    867A    STX $7A
A488    847B    STY $7B
A48A    207300  JSR $0073
A48D    AA      TAX
A48E    F0F0    BEQ $A480
A490    A2FF    LDX #$FF
A492    863A    STX $3A
A494    9006    BCC $A49C
A496    2079A5  JSR $A579
A499    4CE1A7  JMP $A7E1
A49C    206BA9  JSR $A96B
A49F    2079A5  JSR $A579
A4A2    840B    STY $0B
```

This is a disassembly from the same start address as Trace began working on in Table 2. Note that, unlike Trace, the Disassembler is not affected by the jump instructions, it simply ploughs through the area of memory in sequential order.

To enter the Disassembler, specify option 6 on the main menu. You will be asked to give the start address for Disassembly and to choose between output to screen or printer. Memory will be disassembled in blocks of 20 instructions, with a prompt to continue or quit at the end of each.

One point to note in relation to the assembler is the presence of '???' flags in the listings which it produces. These signify that a combination of bytes has been encountered in the memory which cannot be translated into a valid assembly language instruction. This can be for one of three reasons:

a) The code is actually invalid eg an improperly written program you have entered yourself.

b) You have asked the Disassembler to begin its disassembly partway through a valid instruction. In this case it may take several rejected bytes before it comes across the beginning of a valid instruction.

c) The Disassembler may have encountered a table (or individual bytes) of data in the memory. Such tables are purely there to store data and do not represent machine code instructions. Care should be taken in the interpretation of

disassembled instructions which appear interspersed with numbers of '???' indicators. These instructions are probably combinations of data bytes which only look like assembly language.

At the end of tables it is also possible that the Disassembler may be confused by the last one or two bytes of data combining with the first byte of the instruction which follows the table, to produce a spurious instruction. This is a problem common to all Disassemblers – they can only report what is actually there and if, purely by chance, it is a recognisable machine code instruction then it must be reported as such. When tables are encountered in the memory it is wise to stop the Disassembler and to start it again at the byte which appears to follow the table, then start it at the next byte, then the next. If the first start produces sensible code, while the following two do not then the first start probably represents the instruction which follows the table. If, however, two or more of the starts produce sensible (but different) instructions, then you will have to decide for yourself where the table ends and the machine code instructions recommence.

The main use of the Disassembler will be to examine the contents of your own programs, especially those which you have assembled in parts, to check that they are correct before you execute them. It can also be used to deepen your understanding of the internal workings of the 64 by examining the contents of ROM routines in the Interpreter and the Kernal.

The File Editor

While it is perfectly possible to enter machine code programs into memory directly using the Monitor, most programmers prefer to enter their programs in the form of 'source files' of assembly language and to leave it to an assembler to do the work of translating the program into valid machine code. The File Editor provides you with a convenient means of entering assembly language programs, editing them in a variety of ways and saving them to tape or disc for subsequent use. The File Editor is entered by specifying option 7 on the main menu and has its own separate menu of facilities which are described below.

FILE EDITOR MENU OPTION 0: Exit From File Editor
Having entered the File Editor you may return to the main menu at any time by entering this option on the menu.

FILE EDITOR MENU OPTION 1: Input Line(s)
The Assembler is set up to work on assembly language programs expressed in numbered lines similar to those of a BASIC program. To input a line to the File Editor, call up this option and, when the '?' prompt appears, enter a line number followed by the assembly language instruction you require. Line numbers may

be in the range 1-99999 and lines will be automatically inserted into the file in the correct position indicated by the line number. The File Editor will automatically insert a space between the line number and the main body of the file but will not space out the opcode mnemonic and the operand.

The total number of lines which may be entered in any one file is 1000, provided that this does not exceed the overall memory limit of 15K available to the user with Mastercode.

Individual lines may be deleted from the file by entering their line number only.

To quit this option press RETURN without entering a new line and execution will be returned to the File Editor menu.

Note that a line may be no longer than 80 characters and may not contain either commas or semi-colons.

FILE EDITOR MENU OPTION 2: List Lines

This option allows sections of a file to be listed in the same manner as a BASIC program.

On calling up this function you will be asked to specify the first and last lines to be listed. Pressing RETURN, without any lines specified, will list the whole of the file. The format for specifying a listing of part of the file can be any of the following:

-570 - would list all lines up to line 570
460- would list all lines from 460 onwards
300-500 would list all lines from 300 to 500 inclusive. In the event that the lines specified as the ends of the ranges are not present in the file the listing will be done from the line after the first one specified to the line before the last one specified.

If you are listing more than a single screenful of lines at one time then the scrolling of the lines may be slowed down by depressing the CTRL key.

At the end of the listing the program jumps to line entry mode, thus enabling you to enter lines with the listing displayed on the screen. Quitting the function is therefore exactly the same as quitting the previous function, that is to say by pressing RETURN without entering a line.

FILE EDITOR MENU OPTION 3: Delete Lines

This option works in the same way as List except that lines in the range specified are deleted from the file.

FILE EDITOR MENU OPTION 4: Renumber File

Calling this option will renumber all the existing lines of your file, starting with line 10 and proceeding in steps of 10.

FILE EDITOR MENU OPTION 5: Initialise File

Calling this option deletes the current contents of the file, thus allowing a new file to be entered. Note that when loading assembly language files back from tape or disc the reloaded file will be merged into the existing file unless the existing file is first deleted using this option.

FILE EDITOR MENU OPTIONS 6 AND 7: Saving and Loading Files

Assembly language source files may be saved to or loaded from tape or disc using these options. As with machine code files using the Monitor, you will be asked to specify the output/input device and a name for the file. It is wise to choose names which distinguish clearly between assembly language files and machine code files since Mastercode will not allow a machine code file to be loaded by means of the File Editor or an assembly language file by means of the Monitor. A simple way of distinguishing between the two types is to precede the chosen filename with 'M-' or 'S-' depending on the type of file.

The Save function is also used to output the file to a printer by specifying device 4 when asked for the output device. Given below is a typical short listing prepared using the file editor:

Table 4: Assembly Language File

```
MASTERCODE/S
*    10 LDA #$A9
*    20 LDX #$01
*    30 JSR $AB1E
*    40 LDA $0
*    50 AND #$FE
*    60 STA 0
*    70 RTS
END
```

FILE EDITOR MENU OPTION 8: Add Machine Code to file

The Mastercode assembler allows one or more individual bytes of machine code or tables to be entered directly into an assembly language program using the BYT directive (see Assembler Directives). This option makes use of that capability by allowing you to specify an area of memory which will then be placed into your assembly language program in BYT form, that is with each byte value specified rather than as assembly language instructions. This allows routines from the ROM or from a machine code program already resident in memory to be added to the current file.

Data is added to the current file in the form of numbered lines of 15 bytes each, with each line number incremented by five. Before specifying the line number at which insertion is to begin you should ensure that the lines inserted will not overwrite an existing line of the file. There is no need to specify a finish line when using this option since this will not affect the way in which lines are inserted.

Table 5: Lines inserted into file using Add Machine Code to File.

```
MASTERCODE/S
*    10 BYT $A9.$76.$A0.$A3.$20.$1E.$AB.$
A9.$80.$20.$90.$FF.$6C.$02.$03
*    15 BYT $20.$60.$A5.$86.$7A.$84.$7B.$
20.$73.$00.$AA.$F0.$F0.$A2.$FF
*    20 BYT $86.$3A.$90.$06.$20.$79.$A5.$
4C.$E1.$A7.$20.$6B.$A9.$20.$79
*    25 BYT $A5.$84.$0B
END
```

You may notice that these are the same bytes which were disassembled in **Table 3.**

FILE EDITOR MENU OPTION 9: Change Device Number
Calling this option allows you to change the current input/output device. You can only specify a device which was present when the program was first run.

In using the File Editor it should be remembered that though it allows the convenient entry of lines to an assembly language file, it does not in itself make any check that the lines entered are valid instructions. Its only check is that each line does commence with a line number. Note also that when inputting assembly language instructions, any *commas* in the format should be replaced with *full stops*.

The Assembler

The Mastercode Assembler accepts all the standard assembler mnemonics, *with the exception that commas should be replaced by full-stops.* Mastercode is a two-pass assembler, scanning the program once to ascertain the addresses of any labels and the value of any variables specified and the second time to actually assemble the program into machine code. All legal 6502 assembly language instructions (the same set as the 64's 6510 CPU) are accepted.

The assembler is entered by specifying option 8 from the main menu and, once entered, three main options are immediately available, as follows:

1a) ASSEMBLE TO MEMORY: the file entered by means of the File Editor is translated into machine code and placed *into the memory.* As mentioned in the commentary on the Monitor, programs may be conflated with previously assembled programs by loading an existing machine code program into memory (using the Monitor) and then starting assembly of the second program at the byte following the end of the first program, thus overcoming any problems you may have with the limitation of a single file to 1000 lines. Note that variables and labels from the first program must be redeclared for the second – they are not carried over.

b) ASSEMBLE WITHOUT PLACING IN MEMORY: the file is assembled, with a full listing of all addresses and their contents but memory is unchanged. This

option is ideal for use when checking programs for errors without disturbing the present contents of the memory.
2a) ERROR ONLY LISTING: only those instructions which contain errors will be printed, together with an indication of the nature of the error.

Table 6: ERROR ONLY Listing of File

```
ADD.  DATA      SOURCE CODE
                  20 LDA #%A9
==============================^
   INCORRECT NUMBER BASE ERROR
                  40 JSR #$AB1E
======================^
   ADDRESSING MODE NOT AVAILBLE WITH THI
S OPCODE ERROR
                  50 STA $10000
================================^
   DOUBLE BYTE OUT OF RANGE ERROR

 TOTAL ERRORS IN FILE --- 3
```

2b) FULL LISTING: the full listing of the program is printed including indications of any errors. Note that if there are two errors on the same line, only one will be indicated on any one assembly. Subsequent assemblies will flag any remaining errors once the first batch have been corrected.

Table 7: Full listing of the same file

```
ADD.  DATA      SOURCE CODE
00                10 PRT
                  20 LDA #%A9
==============================^
   INCORRECT NUMBER BASE ERROR
02    A201        30 LDX #$01
                  40 JSR #$AB1E
======================^
   ADDRESSING MODE NOT AVAILBLE WITH THI
S OPCODE ERROR
                  50 STA $10000
================================^
   DOUBLE BYTE OUT OF RANGE ERROR
07    8500        60 STA 0
09    60          70 RTS

 TOTAL ERRORS IN FILE --- 3
```

3) ASSEMBLE TO DEVICE: This option allows you to assemble the program in the form of a file on tape or disc. The advantage of this is that programs may be assembled as if to an area of memory which is in fact unavailable when using Mastercode. The file may then be reloaded into the memory as an independent program using the RELOADER program which follows Mastercode on your tape. For a fuller discussion of this see **Location of Code.**

Display of Assembled Program

The program being assembled is displayed twice, once line by line during Pass 1, when only the current line of assembly language is displayed and again during Pass 2 with the address at which the instruction will be assembled, the contents of the bytes involved and the assembly language instruction.

Assembler Directives

The Assembler provides 7 'directives' for the convenience of the user. These are not assembly language instructions which will be translated into machine code for inclusion in the eventual machine code program but instructions which can be included in a file and which modify the manner in which the Assembler processes the file. The seven assembler directives are:

1) ORG (address)

This directive indicates that the assembly language instruction following it is to be assembled at the address specified – subsequent instructions will follow on from that address. A single assembly language program may contain several ORG directives indicating sections of the program, or even individual bytes, to be placed in entirely different areas of memory. Files which do not contain an ORG directive or instructions which precede the first ORG directive in a program will be assembled beginning at address zero, crashing the system. For this reason it is wise to assemble the file first without placing it into memory so that mistakes may be rectified before anything irrecoverable is done.

Table 8: Assembled Program using 'ORG'

```
ADD.   DATA      SOURCE CODE
00                10 PRT
00                20 SYM
00                25 ORG $2A8
2A8    08         26 PHP
2A9    A900       30 PS LDA #$00
2AB    48         40 PHA
2AC    A900       50 AS LDA #$00
2AE    A200       60 XS LDX #$00
2B0    A000       70 YS LDY #$00
2B2    28         80 PLP
2B3    EA         90 NOP
2B4    EA        100 NOP
2B5    EA        110 NOP
2B6    08        120 PHP
2B7    8DAD02    130 STA AS+1
2BA    8EAF02    140 STX XS+1
2BD    8CB102    150 STY YS+1
2C0    68        160 PLA
2C1    8DAA02    170 STA PS+1
2C4    28        175 PLP
2C5    60        180 RTS
2C6              190 END

 TOTAL ERRORS IN FILE --- 0


PS                2A9
AS                2AC
XS                2AE
YS                2B0
 TOTAL NUMBER OF SYMBOLS --- 4
```

2) PRT
Following this directive output of the assembled program is diverted from the screen to the printer on Pass 2. PRT may be included at any point during the program so that half of the assembled listing can be sent to the screen and the remainder to a printer. You will note that all the assembled listings given in this handbook include a PRT directive, for obvious reasons.

3) SYM
This indicates that the 'symbol table' containing values of variables and addresses at which labelled lines are assembled is to be appended to the listing. For an example of the effect of SYM see under **Variables** and **Labels.**

4) END

Whenever encountered, this directive terminates assembly – it does not have to be placed at the end of a program. When END is used as the last line of a program, its address signifies the first free byte of memory which will follow the assembled program. **Table 8** (above) illustrates the use of this directive.

5) BYT

This directive allows a series of one byte values to be specified in a line, separated by full-stops. The values will be entered directly into memory. The format of BYT is illustrated by the example given below.

Table 9: Listing using 'BYT'

```
ADD.   DATA      SOURCE CODE
00               10 PRT
00     A510      20 LDA $10
02     85FE      30 STA $FE
04     60        40 RTS
05     203F05    50 BYT $20.&77.%101
08               60 END

 TOTAL ERRORS IN FILE --- 0
```

(For an explanation of the number of formats used in line 50 see NUMBER BASE below).

6) DBY

Similar to BYT except that the value specified may be up to two byte range (0-65535). The two bytes will be placed into memory with the high byte first.

Table 10: Listing using 'DBY'

```
ADD.   DATA      SOURCE CODE
00               10 PRT
00     A510      20 LDA $10
02     85FE      30 STA $FE
04     60        40 RTS
05     123407    50 DBY $1234.&3421
09               60 END

 TOTAL ERRORS IN FILE --- 0
```

7) WRD

As DBY except that the two bytes are placed into memory with the low byte first.

Table 11: Listing using 'WRD'

```
ADD.   DATA       SOURCE CODE
00                  10 PRT
00     A510         20 LDA $10
02     85FE         30 STA $FE
04     60           40 RTS
05     34120A       50 WRD $1234.&3412
09                  60 END

TOTAL ERRORS IN FILE --- 0
```

Labels

When writing an assembly language program it is usually not possible to foresee exactly where each instruction will be placed in memory, thus creating a problem when it comes to specifying the address to which jumps of various kinds are to be made. A properly written assembler will therefore provide the facility to 'label' individual instructions in order that in the rest of the program jumps may be specified to that label rather than an address having to be specified. On Pass 1 through the program the assembler will determine the address of each of the labelled lines and assign that address as the value of the label. On Pass 2, any jumps which are specified to a labelled line will have the value of the label inserted as the destination of the jump. Mastercode accepts labels up to six characters in length. The use of labels is illustrated by the listing given in **Table 12.**

Expressions

One of the most useful features of the Mastercode program is its ability to cope with simple expressions in assembly language programs. Such variables may be defined using +, −, * or /, though brackets are not permitted. Expressions may be used to directly load memory locations or to define variables.

Variables

The Mastercode Assembler allows variables to be declared and altered during the course of the assembly language program. Variables are assessed during Pass 1 and the correct values inserted into the program on Pass 2.

It is important to remember when defining variables that no variable may be used to set the value of another variable until it has been defined. Defining a variable after it has already been used in such an expression will result in the UNDEFINED LABEL ERROR when it is first used. No variable may be defined more than once, though if a variable has been invalidly defined (for instance by setting it equal to another variable which has not been defined), it may be used subsequently after any '=' in a line though its value will be zero. If, having been improperly defined, it is later REdefined, the same error which was given for the first, faulty, definition will be flagged until that original definition is removed from

the program. Any other attempt to define the same variable more than once will result in an error.

The use of variables and expressions is illustrated by the following listing:

Table 12: Assembly Listing including Variables and Expressions.

```
ADD.   DATA      SOURCE CODE
00               10 PRT
00               15 SYM
00               17 ORG $C000
C000   A900      20 LDA #KEY-KEY/256*256
C002   A2D0      30 LDX #KEY/256
C004   D003      40 BNE LABEL
C006   4C9EA1    50 JMP $A19E
C009   8D13C0    60 LABEL STA ASAVE
C00C   8E14C0    70 STX XSAVE
C00F   8C15C0    80 STY YSAVE
C012   60        90 RTS
C013            100 ASAVE
C013            110 XSAVE = ASAVE+1
C013            120 YSAVE = XSAVE+1
C013            130 KEY = $D000
C013            140 END

 TOTAL ERRORS IN FILE --- 0


LABEL           C009
ASAVE           C013
XSAVE           C014
YSAVE           C015
KEY             D000
 TOTAL NUMBER OF SYMBOLS --- 5
```

The Symbol Table

On Pass 1, as mentioned, labels and variables are assessed for use in final assembly of the program and stored in an array known as the 'symbol table'. The only limitation on this process is that a maximum of 100 labels or variables is permitted. If this maximum is passed during the assembly of a program an error message is generated and assembly terminates. Inclusion of the directive SYM at any point during the assembly language program will cause the assembler to print out the contents of the symbol table after the assembled listing. Note that the value of improperly defined labels and variables in the symbol table will always be zero.

A specimen symbol table is shown in **Table 12** above.

Comments in Programs

Mastercode allows comments to be entered into programs provided that each comment stands on a line on its own and that the comment line starts with a semi-colon.

Loading Registers with Characters

It is often necessary during the course of programs to load individual registers with the value of an ASCII character for the purposes of comparison with the contents of some other memory location. This can be done for all immediate mode instructions by including the character as the operand, preceded by a quotation mark, eg LDA #'F would load the accumulator with the ASCII value of 'F'

Error Messages

The Mastercode Assembler provides full error checking of assembly language programs and generates a variety of error messages to indicate faults in a program. When an error is flagged, either in an error only listing or a full listing, the offending line is displayed, together with the relevant error message and a rough indication of where in the line the error occurs. The error messages are as follows:

1) SINGLE BYTE OUT OF RANGE: this signifies that an attempt has been made to load a single byte or an 8 bit register with a value outside the range 0-255.

2) DOUBLE BYTE OUT OF RANGE: an attempt has been made to load a sixteen bit location (2 bytes) with a value outside the range 0-65535.

3) INVALID OPERAND OR OPCODE: either the opcode mnemonic or the format of the operand do not conform to a known opcode or operand.

4) INVALID OPERATOR: an attempt has been made to use a mathematical operator which Mastercode cannot deal with (see **Variables**).

5) INDEX IS NOT X OR Y: the format of the instruction suggests that an indexed addressing mode is required but neither the X nor Y register has been specified as the index register.

6) LABEL IS NOT ALPHANUMERIC: an attempt has been made to define a label name which contains characters which are not letters or numeric digits.

7) INCORRECT NUMBER BASE: the format of a number entered into the program does not conform to the necessary format of the number base specified (see **Number Base**).

8) LABEL DEFINED TWICE: an attempt has been made to use the same label for two different lines.

9) BRANCH OUT OF RANGE: branch instructions may only refer to addresses which are in the range +127 to −128 relative to the address of the first byte following the branch instruction. This error is given if that limit is exceeded.

10) UNDEFINED LABEL: reference has been made to a label in the program but no line with that label has been discovered.

11) OUT OF SYMBOL SPACE: the maximum number of labels and variables (100) has been exceeded.

12) DIVISION BY ZERO: an attempt has been made to obtain the product of an expression which involves division by zero.
13) ADDRESSING MODE NOT AVAILABLE WITH THIS OPCODE: both the opcode and the operand may be legal but they are not found together in the 6502/6510 instruction set.

Number Bases

Values may be entered into an assembly language program in any one of four bases, namely hexadecimal (base=16), decimal (base=10), octal (base=8) and binary (base=2). The formats are as follows:
Hexadecimal: number preceded by '$' eg $FF =255 decimal.
Decimal: number without identifier, eg 255.
Octal: number preceded by '&' eg &377 = 255 decimal.
Binary: number preceded by '%' eg %11111111 = 255 decimal.

Invalid inputs, for instance $12AZ, which do not conform to the number base specified will be flagged with a suitable error message. The exception to this is the case of hexadecimal numbers which begin with one of the characters A–F but omit the '$' identifier. These will be interpreted as labels and an undefined label error will be generated.

Given below is a specimen listing of a program employing numbers of different bases.

Table 13: Listing using different Number Bases.

```
ADD.   DATA      SOURCE CODE
00               10 PRT
00     AD0001    20 LDA $100
03     AE0001    30 LDX %100000000
06     AC0001    40 LDY 256
09     8D0001    50 STA &400
0C     60        60 RTS
0D               70 END

 TOTAL ERRORS IN FILE --- 0

 TOTAL NUMBER OF SYMBOLS --- 0
```

Location of Code for Assembled Programs

Mastercode is designed to make use of the 'invisible' 4K area of RAM beginning at C000 hex for the assembly of programs. This area provides more space than most machine code programmers will ever require for their programs. If you do wish to enter programs of more than 4K in length or wish to relocate your programs for any reason then you have the facility to assemble the program to the area commencing at C000 for testing. Then you can assemble it to tape or disc but with the ORG set at another location, with subsequent sections of the

overall program being assembled to tape or disc in such a way that, when reloaded into the memory, the sections will form a single continuous program. The method for assembling to tape or disc was described under the commentary on the assembler. To reload assembled programs into the desired area of memory use the RELOADER program located on the tape following the Mastercode program. Running this program will automatically load the assembled file back into memory at the location specified when it was assembled to tape or disc.

Given below are a series of listings which illustrate the methods involved in patching together a series of program sections so that they will eventually constitute a single program.

Table 14: Listings illustrating program assembled in parts.

```
ADD.   DATA      SOURCE CODE
00                 10 PRT
00                 20 ORG $C000
C000   A9A9        30 LDA #$A9
C002   A200        40 LDX #$00
C004               50 END

 TOTAL ERRORS IN FILE --- 0

 TOTAL NUMBER OF SYMBOLS --- 0
```

```
ADD.   DATA      SOURCE CODE
00                 10 PRT
00                 20 ORG $C004
C004   201EAB      30 LABEL JSR $AB1E
C007   85FC        40 STA $FC
C009   86FD        50 STX $FD
C00B               60 END

 TOTAL ERRORS IN FILE --- 0

LABEL              C004
 TOTAL NUMBER OF SYMBOLS --- 1
```

```
ADD.  DATA      SOURCE CODE
00              10 PRT
00              20 ORG $C00B
C00B            30 LABEL = $C004
C00B  CA        40 DEX
C00C  D0F6      50 BNE LABEL
C00E  60        60 RTS
C00F            70 END

 TOTAL ERRORS IN FILE --- 0


LABEL           C004
 TOTAL NUMBER OF SYMBOLS --- 1
```

Note that each section of the program begins at the address of the END statement of the previous section and that labels referring to previous sections are declared as variables, since they will no longer be contained in the symbol table.

Programs which are small enough to fit into the 4K area beginning at C000 hex may be saved using the straightforward Machine Code Save function of the Monitor and reloaded with the simple BASIC program given under the commentary for that section. Having reloaded the machine code program into memory it is run with a straightforward SYS command to the address at which the program's instructions begin. If you store data at the beginning of your machine code program remember that the ORG of the problem will not be the same as the address which will be specified in the SYS command.

Using Machine Code with BASIC

Machine code programs written with the aid of Mastercode can be used in BASIC programs in one of two ways:

(i) If the program is a straightforward machine code file saved from memory use the lines given above under Machine Code Save to reload the code.

(ii) If the machine code program was 'assembled to device' then the Reloader program at the end of the Mastercode tape must be used to place it in memory.

If you place your code in the memory area beginning at C000 hex, all you then need do is to SYS the correct memory address. If, for some reason, you wish to load the code into the area of memory which is normally used by the BASIC system then it will be necessary to first lower the top of memory pointer by POKEing a new value into the system variable at locations 55-56 decimal. Make sure you leave enough room for your BASIC program and any variables and strings it may require and then assemble your machine code program so that it will fall above the new top of memory, using the BASIC program to reload the

assembled code into the correct position.

Remember that the simple loader given in the commentary on Machine Code Save will only load a file back into the same area of memory as that from which it was saved. RELOADER, on the other hand will only load a file assembled to tape or disc using the assembler.

Warning: The Mastercode will stop with an error if you attempt to RUN the assembler when no lines of assembly language have been entered.

Table 15: Available Instructions and Formats.

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
00	BRK	
01 X	ORA	(ADDR.X)
05 X	ORA	ADDR
06 X	ASL	ADDR
08	PHP	
09 X	ORA	#DATA
0A	ASL	A
0D XX	ORA	ADDR2
0E XX	ASL	ADDR2
10 X	BPL	DIS
11 X	ORA	(ADDR).Y
15 X	ORA	ADDR.X
16 X	ASL	ADDR.X
18	CLC	
19 XX	ORA	ADDR2.Y
1D XX	ORA	ADDR2.X
1E XX	ASL	ADDR2.X
20 XX	JSR	LABEL
21 X	AND	(ADDR.X)
24 X	BIT	ADDR

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
25 X	AND	ADDR
26 X	ROL	ADDR
28	PLP	
29 X	AND	#DATA
2A	ROL	A
2C XX	BIT	ADDR2
2D XX	AND	ADDR2
2E XX	ROL	ADDR2
30 X	BMI	DIS
31 X	AND	(ADDR).Y
35 X	AND	ADDR.X
36 X	ROL	ADDR.X
38	SEC	
39 XX	AND	ADDR2.Y
3D XX	AND	ADDR2.X
3E XX	ROL	ADDR2.X
40	RTI	
41 X	EOR	(ADDR.X)
45 X	EOR	ADDR
46 X	LSR	ADDR
48	PHA	
49 X	EOR	#DATA
4A	LSR	A
4C XX	JMP	LABEL
4D XX	EOR	ADDR2

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
4E XX	LSR	ADDR2
50 X	BVC	DIS
51 X	EOR	(ADDR).Y
55 X	EOR	ADDR.X
56 X	LSR	ADDR.X
58	CLI	
59 XX	EOR	ADDR2.Y
5D XX	EOR	ADDR2.X
5E XX	LSR	ADDR2.X
60	RTS	
61 X	ADC	(ADDR.X)
65 X	ADC	ADDR
66 X	ROR	ADDR
68	PLA	
69 X	ADC	#DATA
6A	ROR	A
6C XX	JMP	(LABEL)
6D XX	ADC	ADDR2
6E XX	ROR	ADDR2
70 X	BVS	DIS
71 X	ADC	(ADDR).Y
75 X	ADC	ADDR.X
76 X	ROR	ADDR.X
78	SEI	
79 XX	ADC	ADDR2.Y

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
7D XX	ADC	ADDR2.X
7E XX	ROR	ADDR2.X
81 X	STA	(ADDR.X)
84 X	STY	ADDR
85 X	STA	ADDR
86 X	STX	ADDR
88	DEY	
8A	TXA	
8C XX	STY	ADDR2
8D XX	STA	ADDR2
8E XX	STX	ADDR2
90 X	BCC	DIS
91 X	STA	(ADDR).Y
94 X	STY	ADDR.X
95 X	STA	ADDR.X
96 X	STX	ADDY.Y
98	TYA	
99 XX	STA	ADDR2.Y
9A	TXS	
9D XX	STA	ADDR2.X
A0 X	LDY	#DATA
A1 X	LDA	(ADDR.X)
A2 X	LDX	#DATA
A4 X	LDY	ADDR
A5 X	LDA	ADDR

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
A6 X	LDX	ADDR
A8	TAY	
A9 X	LDA	#DATA
AA	TAX	
AC XX	LDY	ADDR2
AD XX	LDA	ADDR2
AE XX	LDX	ADDR2
B0 X	BCS	DIS
B1 X	LDA	(ADDR).Y
B4 X	LDY	ADDR.X
B5 X	LDA	ADDR.X
B6 X	LDX	ADDR.Y
B8	CLV	
B9 XX	LDA	ADDR2.Y
BA	TSX	
BC XX	LDY	ADDR2.X
BD XX	LDA	ADDR2.X
BE XX	LDX	ADDR2.Y
C0 X	CPY	#DATA
C1 X	CMP	(ADDR.X)
C4 X	CPY	ADDR
C5 X	CMP	ADDR
C6 X	DEC	ADDR
C8	INY	
C9 X	CMP	#DATA

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
CA	DEX	
CC XX	CPY	ADDR2
CD XX	CMP	ADDR2
CE XX	DEC	ADDR2
D0 X	BNE	DIS
D1 X	CMP	(ADDR).Y
D5 X	CMP	ADDR.X
D6 X	DEC	ADDR.X
D8	CLD	
D9 XX	CMP	ADDR2.Y
DD XX	CMP	ADDR2.X
DE XX	DEC	ADDR2.X
E0 X	CPX	#DATA
E1 X	SBC	(ADDR.X)
E4 X	CPX	ADDR
E5 X	SBC	ADDR
E6 X	INC	ADDR
E8	INX	
E9 X	SBC	#DATA
EA	NOP	
EC XX	CPX	ADDR2
ED XX	SBC	ADDR2
EE XX	INC	ADDR2
F0 X	BEQ	DIS
F1 X	SBC	(ADDR).Y

OBJECT CODE	INSTRUCTION	OPERAND FORMAT
F5 X	SBC	ADDR.X
F6 X	INC	ADDR.X
F8	SED	
F9 XX	SBC	ADDR2.Y
FD XX	SBC	ADDR2.X
FE XX	INC	ADDR2.X

Notes:
'X' or 'XX' in the first column refers to the number of bytes following the opcode.
'ADDR' refers to a single byte address.
'ADDR2' refers to a two byte address.
'DIS' refers to displacement.
'DATA' refers to a data byte.

Printed in UK

SUNSHINE

commodore 64
mastercode assembler