

C-128 Midnight Assembly System

by

Matthew Montchalin

Copyright 1987

Copyright 1988

All Rights Reserved

Mountain Wizardry Software

Warranty

Mountain Wizardry Software will for the first year after purchase replace any original master diskettes found to be defective, flawed, or corrupted.

Disclaimer

No program of this size and complexity can be regarded as perfect or complete. *Mountain Wizardry Software* reserves the right to update, upgrade, and improve the program. *Mountain Wizardry Software* will not be held liable for any errors or damages consequent or incidental to the use of this program.

Foreword

MAS-128 is Mountain Wizardry Software's Symbolic Assembler for the C-128; it is derived directly from the *Midnight Assembly System* for the C-64. Both versions are authored by programmer Matthew Montchalin. While many of the varied features of *MAS-64* were coresident in memory, and *that* is a programming feat in itself, *all* are now memory co-resident in *MAS-128*; indeed, numerous other features and commands have been added. A notable feature of *MAS-128*, thanks to the memory architecture of the C-128, is its ability to recognize the bank that it is loaded into and accordingly coordinate its activities in either RAM 0 or RAM 1 (respectively memory configurations 3F and 7F); object code generation defaults into the other bank. This allows considerable flexibility in the assembling of text into object code.

Acknowledgements

Special acknowledgement is made towards R.Eric Lee of *Busy Bee Software*, without the use of whose native C-128 mode word processor these doc's would never have seen the light of day. More so, I am deeply indebted to my folks and friends for their patience, toleration, and support towards this most unusual obsession of obsessions, *Assembly Language Programming*.

TABLE OF CONTENTS

Introduction	
Getting Started	
Software on Diskette	
Disk Support	
Screen Editor	
Quick Keys	
Immediate Commands	
Pseudo-Opcodes	
User Expansion	
User Opcodes	
File Conversion	
S-128 Disk Utility	
Breaker-128	
Swirling Sprites	
Music Demo	
Appendix I	<i>MAS-128 Memory Map</i>
Appendix II	<i>MAS-128 ASCII</i>
Appendix III	<i>MAS-128 Text Structure</i>
Appendix IV	<i>6510/8502 Overview</i>
Appendix V	<i>Memory Management Unit</i>

Introduction

This manual describes the essentials in operating the *Midnight Assembly System for the C-128*. Programmers familiar with the C-64 version may wish to skip this introduction.

It is assumed that you are familiar with rudimentary Assembly Language Programming. If you are not, there are many informative manuals and publications readily available today; as a matter of fact, it would be a good idea even for an advanced programmer to obtain some source of reference material. Many Users' Groups also routinely offer seminars on Assembly Language Programming. The beginner is emphatically urged to persevere and march on at his own pace; excellence can only be achieved through patience and dedication. It does not take long before the rough and uncertain attempts of the initiate are replaced by the skillful nuances of the master.

If you are familiar with assembly language programming for other microprocessors but are a bit rusty with regards to the 8502 (which is functionally identical to the 6502), you should refer to the appendix on the overview of the 8502 microprocessor. Conventions on assembly language syntax should be ascertained through reference to a programming manual.

Symbolic assemblers distinguish themselves from *objective* assemblers chiefly in their degree of flexibility and versatility. The programmer is permitted to refer to addresses or values not by a definite number (though he may if he so wishes) but by arbitrary names or *symbolic labels*. Indeed, he can assemble code not just by the line but *by masses of lines*, where the definition of a symbolic label more often than not takes its value from its position in the code relative to the Program Counter. It is especially the virtue of the *symbolic* assembler to determine on its own the value of a symbolic label for the implementation of jumps, branches, and memory references.

GETTING STARTED

First and foremost, be *sure* to backup your master disk! MAS-128 is in *no* wise copy-protected (but *is* copyrighted), so any difficulties in loading are probably due to disk alignment problems on your end. In any case, if you simply *can't* get it loaded, and if you are just *certain* that it got corrupted somehow or in some way was defective, or that you perhaps had rendered it defective, simply return the original diskette and *Mountain Wizardry Software* will replace it as per the conditions of your warranty.

Those programmers that are equipped with 1700/1750 REU's will find it quite easy to back up their systems software using the built-in RAMDOS commands for *downloading* and *uploading* the files on disk. If you are not equipped with a Ram Expansion Unit, you may wish to use your favorite archival utility, or the S-128 archival program included on the system disk. MAS-128 is not copy-protected, so you should not replicate any disk errors or vagaries that may be found on your disk.

Now that you've backed up your disk, sit back and familiarize yourself with the documentation by flipping through these pages and reading whatever may interest you. Advanced programmers may wish to skip the introductory sections, scan the immediate commands, and then read through the appendices.

LOADING MAS-128

The first file on your system disk, MAS-128, is a pure binary object file that can execute in either RAM 0 or RAM 1 at address \$1700. If only to ensure loadability with any Commodore compatible disk drive, no fast boots were prefixed to the body of MAS-128. Those who want to prefix a fast-booting header on MAS-128 are urged to do so. Best results are achieved by loading the file directly via a monitor program. However, to load up MAS-128 from CBM BASIC, simply enter a command such as one of these given below:

```
l"mas-128 v.05",08           :from CBM Monitor (BEST RESULTS)
bload"mas-128",p5888        :rem RAM 0
bload"mas-128",u9,b1,p5888  :rem RAM 1
```

and then execute a Cold Start by specifying the appropriate bank configuration and then entering:

```
g 1700                       : execute address $1700
```

or from Commodore BASIC:

```
bank 0                       : rem configuration $3F
sys 5888                      : rem jmp $1700
```

The credits should now reveal "MAS-128 by Matthew Montchalin (C) 1987 Mountain Wizardry Software". If you see anything else you should reject your copy as unfortunately suspect and obtain a new one if only to ensure that none of it is in

the least corrupted. The Midnight Assembly System is exclusively the intellectual property of Mountain Wizardry Software. This intellectual property has thus far been implemented exclusively for the C-64 and the C-128. Mountain Wizardry Software reserves the right to manifest its property for other computer systems in the future.

CUSTOMIZING YOUR COPY

You should now customize your copy of MAS-128 by adjusting MAS's internal BRK vector to point the way to your own machine language monitor program. Now, when you enter the command BRK, you will exit MAS-128 via this vector. [Similarly, you may wish to adjust the internal BAS vector also.] Even though the C-128 *may* have its *own* built-in monitor program in ROM [*such as that found installed at the factory at \$B000-BFFF*], MAS-128 makes *no* presumptions regarding the operating system in ROM. Besides, the aforementioned monitor program in ROM is without a doubt an exceedingly *poor* one to be saddled with and certainly not worth familiarizing yourself with.

The ROM addressing space \$B000-BFFF is far too valuable a piece of real estate to waste on a piece of shoddy programming like the C-128 internal monitor. You should feel free to replace any or all ROM's with an EPROM of your *own* design. Your entry to the monitor program should initialize the MMU, CIA's, VIC or 8563 chips, as required.

CUSTOMIZING THE QUICK KEYS

You are also seriously urged to provide your *own* definitions for the MAS-128 Quick Keys. To achieve this you should scan through the main body of MAS-128 until you come to the area containing the MAS-128 predefinitions for the Quick Keys. The easiest way to do this is by using MAS-128's DUMP and BNK commands and referring to the MAS-128 memory map. Then, having located the area, you should rewrite the Quick Key Definitions as you feel best, keeping in mind the extended control code set of MAS-128 ASCII.

Having done all this, you should save a new copy of MAS-128 as adjusted for your own personal use. Please note that you may not distribute your customized copy of MAS-128; doing so would be a violation of the author's proprietary rights over his product.

CUSTOMIZING THE VIDEO LAYOUT

The programmer may wish to customize several aspects of MAS-128's 8563 initialization routine. The default cold start provides you with a White Cursor on a Black Screen. The screen editor ordinarily defaults to a layout of 80 columns by 26 rows, but can handle a logical screen up to 80 columns wide and some 32 rows deep. You can adjust the number of rows by increasing the constant *ROWS* to a maximum of 32, and simultaneously editing the other constants, *Fine Vertical Adjust*, *Screen Memory*, *Character Definitions*, and *Color Attribute Memory*. [*Please see Appendix I.*] Most RGB 80 column monitors cannot accept video signals pumped up to more than 30

rows; however, you should still be able to operate at 27 or 28 rows just fine.

Before editing any of the values contained within the MAS-128 initialization routine, you should first try them out by employing the *Video* command in the immediate mode.

CUSTOMIZING THE TAB TABLE

Your copy of MAS-128 contains its own internal tab table. When you save MAS-128 back to disk, the current tab configuration will be saved back with it. The default tab spacing is set at every 11th column. You will want to adjust the respective tabs by moving your cursor to the desired column and then pressing either CTRL K or CTRL L to set or clear the various tabstops.

CUSTOMIZING THE COLDSTART DATE

The system date is stored in the initialization routine for MAS-128, and also, subsequent to coldstart, in zero page. When you enter a date at cold start, the internal date constant is rewritten. You may then wish to save a *new* copy of MAS-128 back to disk with the *new* date. You will certainly want to adjust your coldstart date on a regular basis, say, once every month or so. Note that the date is represented by 16 bits: the month is represented by four bits, the day by five bits, and the year by seven bits. Of the four bytes prefixed to every MAS structured file, two are used to denote the file date and two are used to denote the file size.

SOFTWARE ON DISKETTE

The following files are included on the master diskette:

MAS-128 *Midnight Assembly System* for the C-128, loading in at address \$1700. MAS-128 should always be cold started at its loading address. Warm Start may be executed at address \$1703 if the MAS privileged RAM areas are still intact. Please consult Appendix I for details regarding the MAS-128 memory map.

BREAKER Breaker-128 Source Code. This program is similar to that which was issued with MAS-64, but has been adjusted a bit to deal with the C-128 configuration register at address \$FF00. Not a few programmers resort to use of unimplemented opcodes in order to achieve a measure of obscurity in their code. This program provides a similar degree of obscurity but does not risk potential system crash from failure of the unimplemented opcodes.

S-128 An updated version of the C-64 Disk Zapper, a disk utility that was written a few years ago by the author of MAS-128. This program works with all Commodore-compatible disk drives. It may now be copied freely by registered owners of either MAS-64 or MAS-128. The source code is included in the MAS-128 documentation for your inspection.

SWIRLING A delightful demonstration of the positioning and moving of multicolor **SPRITES** sprites from assembly language. The action of the sprites is constantly changing and mesmerizing, certainly an eye-catcher. This program makes ample use of the Breaker-128 program, above. After assembling in *either* bank of RAM, execute it by JSR'ing \$2000. You can restart this program by pressing the RESTORE key. In order to break out of the program, press the RESET button.

MUSIC Source Code for a strikingly pleasing, yet simple, song. Reviewing this program will yield a number of insights into the mastering of the C-128's SID chip. As a matter of fact, MAS-128 employs the nucleus of this program for use as its *own* SID driver.

NOTES A series of PET ASCII files are included with the most recent comments regarding this version of MAS-128, and impending updates and developments. You may wish to generate hard copies of these comment files for leisurely inspection, or you may display them to the 80 column screen by using the *@display "file name"* command.

MAS SYMS This sequential file contains the actual symbolic labels generated during assembly of MAS-128 itself. The enterprising student may wish to inspect these labels and refer to them in order to understand MAS-128 more clearly. It will also be convenient to load these symbolic labels in order to design MAS user support files, such as that given below-

PRINTER This is the source code for a MAS user support file. It requires a serial printer interface to communicate with non-commodore printers. It may be called in off disk when the user wishes to send a TEXT file to the Printer. You can use this file to transmit any ascii files or, for that matter, *binary* files from disk to your printer via the serial bus.

DISK SUPPORT

MAS-128 communicates with the disk drive according to standard CBM TALK and LISTEN serial protocol. Eight methods of disk access are implemented through a pseudo @-wedge:

@append	"0:filename"	Append file from diskette to the end of the current text stored in the Text Buffer
@display	"0:filename"	Display PETASCII file on screen
@execute	"0:filename"	Execute specially prepared MAS-128 object file
@load	"0:filename"	Load an MAS-128 source text file into the text buffer from disk
@merge	"0:filename"	Load an MAS-128 text file from disk, merging it into the current text in RAM
@read	"0:filename"	Read an MAS-128 text file from disk, listing it non-destructively thru the 8563 video chip
@^	"0:filename"	Save an MAS-128 text file to disk
@view	"0:filename"	View an object file non-destructively through screen memory.
@ "command"		Send a command to disk drive
@ <Expression >		Specify which serial device MAS-128 will address
@# <Expression >		Redefine which serial device the drive will identify itself as

MAS-128 implements ten other forms of disk support in addition to the @-wedge, namely, CON, DIR, DIS, GET, HEX, LOD, PUT, SAV, SEE, and TXT. Some of these commands admit more than one argument; at the very least, they require a final argument enclosed in quotes to serve as the file name. These commands are described in the section on Immediate Commands (q.v.).

It should be noted that after the "@" character is entered, only the first letter of the command need be entered- That is, the command @rear view mirror "filename" will be taken as the *same* command as @read "filename". Also, the device number may be included in the command by immediately following the "@" character. Specifying the device number is purely optional.

EXAMPLES

@7,load	"filename"	Load source file from device 7
@22,^3	"filename"	Save source file to device 22
@24,#14		Redefine device 24 as device 14

The user should take care to distinguish between *drive* numbers and *device* numbers. Simply, a *device* has just one power cord and, usually, one master microprocessor to control all its components, such as stepper motors, read/write heads, etc. A *drive* is merely *one* of the dedicated components in a device. A drive is the part that is employed chiefly for spinning the diskette, stepping the r/w head from track to track, and for reading and writing data on and off a diskette. Now, some devices have *two* drives and are said to be *dual*. Indeed, some devices may be equipped with a great number of drives; for example, it is not hard to envision a single device controlled by one master microprocessor, 256 slave microprocessors working asynchronously or in parallel, each having its own memory and controlling its own drive. If you can picture it, it would have one power supply, 256 disk drive slots, and one serial cable attachment.

Recently Commodore has tried to resolve this confusion by introducing the term *unit* to replace *device*. In fact, the BASIC that comes with the C-128 implements the use of the term *unit* for this very purpose. You can now use such structures as *bload "file name",u8* or *bload "file name" on u8*, and so on. It is a particularly bad habit to think of these intelligent serial peripherals according to Commodore BASIC terminology. Regardless of Commodore's redefinition of terms, we will in these discussions, however, stick with the more familiar term *device*.

CBM serial protocol lets you daisy chain a maximum of 32 devices together, numbered 0 to 31. Serial devices numbered less than 4 may be employed freely if you are willing to refrain from CBM BASIC which interprets these numbers as channels dedicated to other purposes. This point bears emphasis: although Commodore BASIC issues syntax errors when you try to access serial devices 3 and less, this is simply a heavy-handed way of restricting your use of serial device numbers! If you circumvent Commodore BASIC with the TALK and LISTEN routines, there's nothing stopping you from accessing serial devices with these numbers. Admittedly, there may be few devices, or none at all, on the *consumer* market which identify themselves by numbers 3 and less, but you should be aware of their potential existence.

In all practicality, however, you should refrain from defining a disk drive device either as device 0 or device 31 since the TALK and LISTEN commands used for opening and closing channels can get screwed up with the commands of device addressing.

In communicating with your serial device you should be aware of the terms *primary* and *secondary* addresses. The *primary* address is simply the number of the device, such as device 8, 9, 10, and so on. The *secondary* address is a number that serves as a command for the device addressed.

Those who have equipped themselves with the 1700/1750 RAM Expansion Units will be glad to know that we have included an extensive RAMDOS with MAS-128. In order to access our 1750 RAMDOS, you will have to address serial device zero, which number for our purposes has been dedicated for expansion RAM specifications. support may soon be forthcoming for the 64K Battery Backed *Quick Brown Box*; when available it will be accessed as serial device 1.

RAMDOS COMMANDS

FUNCTION

@ "c0:file2=0:file1"	Create file2 from file1. Note that you can change a file's filetype by using this command. For example, if you issue the command @ "c0:abcd,seq=0:mnop,prg" you will produce an exact duplicate of <i>mnop</i> but with a sequential filetype designator instead of a program file type designator.
@ "d0:8"	Download all files from Serial Device 8.
@ "i0"	Initializes RAMDOS (refreshes RAMDOS Block Allocation Map and closes read & write channels) Early versions of the RAMDISK operating system fail to close input/output channels after certain rare operations. To straighten things out you may wish to issue this command.
@ "l0:filename"	Lock file. Locked files may <i>not</i> be scratched, so use this command with care! If you want to get rid of a locked filetype you will have to issue the reset command <i>new0</i> (see below).
@ "n0"	New (Kills all files regardless of Lock Status and then performs RAMDOS reset and initialization.) You should be <i>sure</i> to issue this command at the start of your day. The regular MAS-128 initialization routine does <i>not</i> make a call to this RAMDOS reset routine.
@ "r0:file2=0:file1"	Rename or change file's name from "file1" to "file2"
@ "s0:file2"	Scratch (Kill) file2. Please note that <i>Locked</i> files cannot be scratched.
@ "x0:file1=0:file2"	Exchange file1 with file2. This command can facilitate sorting of the files in the directory. It does not copy files or rename them; it only does a swap on the actual file name entries in the directory
@ "u:9"	Upload all files from 1700/1750 Expansion RAM to Serial Device 9. Note: Before uploading, you should <i>definitely</i> have a blank pre-formatted disk in drive 0 of device 9.
@ "v0"	Validate (Reconstruct RAMDOS Block Allocation Map) This validates <i>really</i> fast so don't hold back on trying this one out. Unlike external disk drives, the internal RAMDOS validate command only takes a <i>fraction</i> of a second to go through the whole REU!

DOWNLOADING

The wholesale transferral of files from a diskette to the 1700/1750 REU is said to be *downloading*. To effect this transferral, you need simply access device 0 (the REU) and then issue the download command to MAS-128's internal RAMDOS. The 512K 1750 REU affords you 2000 blocks of storage and over 350 directory entries; the 128K 1700 REU affords you, proportionately, something less than half as much storage and directory space.

EXAMPLE

@0,"d:7" ; download from serial device 7

At times, however, you may wish to be more selective in the files you wish to download from the serial device to the REU. This increased selectivity is made possible by the extended RAMDOS download command where you can specify a number of *trailing attributes*. Essentially, these *trailing attributes* let you specify in sequence the *serial* device number, the *disk size*, and the *origin* of the directory in terms of track and sector. Please note that in the examples below, all trailing attributes are delimited by *periods* and certainly not spaces, commas, or colons.

Ordinarily, you needn't specify disk size or directory origin because MAS-128 has been provided with enough intelligence to figure it out for you. Nevertheless, it is conceivable that you may wish to define and redefine these variables at your own option for the express purpose of downloading segments of a disk directory.

EXAMPLES

@0,"d:12" ; download from serial device 12

@0,"d:8.37" ; download from serial device 8
which has tracks from 1 to 37,

@0,"d:9.80.40.5" ; download from serial device 9
which has tracks from 1 to 80,
with the directory starting
at track 40, sector 5

TECHNICAL NOTE: The download procedure is brought about by driving the disk drive zero page job queue for direct track and sector access! This lets you read in many disks that could otherwise balk at any number of DOS errors and so preclude normal reads. If you feel your diskette is in any way suspect, you are seriously urged to *upload* the contents of the REU to a brand new formatted diskette.

PARTIAL DOWNLOADING

The commands detailed above provide you with the means of transferring complete disks of information to & from the 1700/1750 REU. Sometimes, however, you may wish to initiate a transfer involving only certain select files. The extended download syntax given below lets you do just that:

EXAMPLES

- @0,"d:9=:file name" ; download from serial device 9
a single file by the name of
"file name"
- @0,"d:11=:sprite * " ; download from serial device 11
just those files whose first
five letters begin with "sprite"
- @0,"d:8=:a * t" ; download from serial device 8
any file that begins with an "a"
and ends with a "t"
- @0,"d:12=: * _sou" ; download from serial device 12
any file that ends in "_sou"
- @0,"d:8=:???" ; download from serial device 8
any file whose name is just three
characters long.
- @0,"d:8=:???sprite?3" ; regardless of the second to last
character, and/or the first three
letters, download any file which
otherwise would be identified by
"sprite"

FULL SCREEN EDITOR

MAS-128 employs full 80 column screen editing in a manner much the same way MAS-64 edits in 40 columns. The quote mode familiar to Commodore programmers is present also but may be canceled by pressing (simultaneously) ESC OFF.

Cursor Movement

The cursor is moved by pressing any of the four keys with arrows found between the *NO SCROLL* key and the keys labeled F1-F7. The user accustomed to the C-64 may also press either of the two keys just below the RETURN key to move *down* or *right*, and in combination with a SHIFT key move *up* or *left*. Special note about backspacing (cursoring left): if you cursor left from TAB 0 the cursor will move to the last character of the line above.

Auto-Insert

The Auto-Insert editing mode is disabled on cold start. You can enable it by pressing ESC A (simultaneously). When you enter characters on a text line in Auto-Insert mode, an insert will be performed automatically before the character is printed on the screen. To disable the Auto-Insert mode, press ESC OFF.

Editing

Editing is accomplished not only by cursoring around but by pressing the DEL key to "eat" the character to the left of the cursor or by pressing SHIFT DEL to open up a space to the right of the cursor. A variation of the familiar delete function is achieved by pressing CONTROL DELETE.

These standard editing keys function as they do on a C-64. Entire lines of text in the source buffer may be moved around by using the immediate commands MOVE, SWAP, and COPY. Text lines may be deleted by using the DELETE command. Searching and Replacing is accomplished through use of the EDIT command.

Entering Text

MAS-128 interprets the RETURN key somewhat differently than does MAS-64. With the press of the RETURN, *regardless* of the state of the SHIFT KEY, the text line (if present) is entered into the Text Buffer. However, *after* the cursor is advanced to the next line, TAB is cleared to zero only if the SHIFT key is *not* pressed; otherwise, TAB is set to point at the last non-space character of the new row!

Like MAS-64, MAS-128 is also a line-oriented assembler. Text Lines are entered into the Text Buffer *only* if they are preceded by a series of decimal characters, i.e., a line number expressed in decimal form, and if the number evaluates out to a 16 bit integer between 0 and 65535. All Text Lines are ordered one after another according to strict numerical sequence. This allows for simple text structuring, manipulation, editing, and listing.

Scrolling Text

Some commands will automatically scroll your text; for instance, if you are LISTing or READing source files. Trying to move the cursor past the top or bottom rows of the screen will scroll any resident text into visible 8563 screen memory if at least some other decimal characters are directly "in sight" of the left hand of the screen and not blocked by nonnumeric characters. (Try it!) You can also press the keys to the left or right of the 40/80 key to bring about scrolling; the current position of the cursor will remain unaffected. This latter method affords you some easy scrolling when you come to find that the cursor need not be moved to the other end of the screen just to scroll a few lines of text into sight.

Quick Key Phrases

At the right of the C-128 keyboard, below the Function Keys F1-F8, you will find a numeric keypad. The MAS-128 key scan recognizes these 14 keys as being rather special. If you simultaneously press a SHIFT Key while pressing one of the keys on the keypad, you will print out 14 of the 16 pre-defined Quick Key Phrases. You may or may not agree with the definitions that the Author has provided for these keys. You may wish to customize these key-definitions for your own personal use. In order to customize your copy of MAS-128 so that these keys will always say what you want them to, it is necessary to locate and edit the actual addresses of these Quick Key Strings, as found buried within MAS-128 itself. To do so, you should use the BNK and DUMP commands to locate the address within MAS, and then drop to a monitor for the actual rewriting of the strings.

Besides the 14 Quick Keys mentioned above, there are two more, and as a matter of fact, they are not accessed through the numeric keypad.

The 15th Quick Key Phrase is not at all expressed via *any* combination of the SHIFT and Keypad Keys. In order to express the 15th Quick Key Phrase, you have to press ESC * simultaneously instead. This Quick Key Phrase can be redefined quite easily, as needed, by pressing ESC = simultaneously.

The 16th Quick Key has been dedicated for use with the HELP key. You should not alter its definition; it is intended for reading the HELP file on the system disk.

Redefining Quick Keys

Aside from permanently *customizing* your copy of MAS-128, you can also redefine the Quick Keys on a temporary basis by using the KEY command. In order to replace a Quick Key's definition with one of your own, you have to supply two arguments. The first argument specifies which of the 16 Quick Keys you mean to define; the second argument must be a string of characters enclosed in quotation marks. Each Quick Key Definition is limited to 32 bytes; moreover, a Quick Key Definition is terminated by the first null byte in its character sequence. Remember, you should not try to delimit the arguments with periods, commas, colons, or parentheses! The proper delimiter is the simple unshifted space character.

Expression Evaluation

The expression evaluator for *MAS-128* is capable of evaluating a mix of hexadecimal, binary, and decimal constants and symbolic label arguments with 16 bit precision. You should be sure to prefix all hexadecimal constants with the dollar sign "\$" and binary constants with the percent key "%". The results of an operation are regarded as zero page if all the arguments had zero for a high byte, if decimal constants consisted of 3 characters or less, if binary constants consisted of 8 characters or less, if hexadecimal constants consisted of 2 characters or less, *and* if the result of the operation had a zero for a high byte. In other words, if any argument in the operation was absolute or if the high byte of the result was absolute, then the final value would be taken as an absolute value.

Only the first 5 decimal characters are regarded as decimal, the first 4 hexadecimal characters are regarded as hexadecimal, and the first 16 binary characters as binary. If the next character is not a SPACE (end of operation terminator), or a plus, minus, times, divides, or shift left/right character, an **ILLEGAL OPERATOR** error message will be issued.

The following characters should be used as operators for mathematical operations:

- + Addition
- Subtraction
- * Multiplication
- / Division
- < Shift Left
- > Shift Right

Please note that with immediate addressing, it is always the lower 8 bits of the evaluated result that is taken as the operand. For example, if you want to load the accumulator with the *high* byte of the immediate value # $\$1234$, simply divide it by 256 or shift it right eight times. For example, here are three ways of effectively entering the same thing:

```
LDA # $\$1234/256$ 
LDA # $\$1234/16 > > >$ 
LDA # $\$12$ 
```

You should feel free to utilize the *MAS-128* expression evaluator as you find most convenient. Any given expression may involve a mix of variables and binary, decimal, or hexadecimal constants. A variable is also further defined as *zero page* or *absolute*. If at any time *any* part of a variable's makeup has been defined as *absolute*, then the entire variable is said to be absolute. For example, if the label *Somewhere* were defined to equal \$00 then it would be regarded as zero page. If it were defined to equal \$00+\$0000 then it would be taken as *absolute*. Similarly, if a label called *Another_place* were defined as the sum of \$01 plus \$ff it would be taken as absolute also.

Easy conversion of numeric types may be achieved by simply entering the equals sign and the expression to be evaluated. The equivalent values of the computed result will be displayed on the line below. For example, if the user entered:

= \$3341

then the equivalent values would be computed and displayed below:

; \$3341 = %0011001101000001 = 13121 = "3a"
hex binary dcm asc

The semicolon precedes the display of equivalent values in order to render inert any stray carriage returns that may land on the line. The semicolon denotes that the rest of the line is not to be acted upon by the *MAS* parser.

A special argument is the equal sign itself. It denotes the current value of the Program Counter. The value of the PC is constantly changing through the course of the assembly and as such is ideal for references to memory which are relative to the PC. On assembly, the PC defaults to \$200 of the opposite bank of RAM (ordinarily RAM 1 or Configuration \$7F). The effective address of the PC depends also on the current state of the reverse generation counter. Ordinarily, the reverse generation counter equals zero. (See the Appendix on the MAS-128 Memory Map)

IMMEDIATE COMMANDS

Some commands require arguments that are either *file names* or *expressions*. If a command allows of several arguments, they should *not* be delimited by commas, unless *specifically* otherwise indicated; rather, they should be clearly separated with lowercase spaces. Use of uppercase spaces are *not* interpreted as field delimiters! File names, on the other hand, *may* have embedded spaces, lowercase or uppercase, but *must* be delimited by quotes. It should be emphasized moreover that, although the immediate commands are presented here as three uppercase characters, they may also be entered in lowercase form.

ASM Initiate a full two pass assembly of any source code in text RAM (or any text file if a file name is supplied). The first pass yields the symbol table. The second pass generates the object code. Before assembling a file, you should always be sure to specify *origin* and *output mode*. If you fail to specify origin (using the *org* pseudop) or output mode (using the *out* pseudop), assembly will default to an origin of \$0200 and the output mode will be assumed to be *non-destructive* or non-generating. Thus, if MAS-128 resides in Bank 0 (configuration \$3F), source will be taken to start from \$0200 in RAM 1 (configuration \$7F) *under Common RAM* and no actual bytes will be stored in RAM.

EXAMPLES

ASM ; Proceed to assemble any text that is in the TEXT buffer

ASM "file name" ; Get specified file into TEXT buffer and assemble it

AUT The user may specify an auto-numbering increment with this command. If the auto-number increment is greater than zero, then every time the user enters a line of text into the text buffer, a line number equal equal to that just entered plus the auto-number increment will be printed out on the screen line just below. If you want to turn off auto-numbering, simply enter this command *without* an argument- The auto-numbering increment will be reset to zero.

EXAMPLES

AUT 10 ; Turn on Auto-number Mode

AUT 0 ; Turn off Auto-number Mode

BAS Entering this command causes an indirect jump off of Vector \$XXXX, aiming for the C-128 Cold Start address \$E000 with all ROMS banked in. The user may wish to set this vector to point to a subroutine of his own design. BAS and BRK are commands which the User should customize for use with his own individual system. Unless you customize your copy of MAS-128, entering either BAS or BRK will both yield a cold start of the system by banking in all ROM's and JMP'ing to \$E000. Neither BAS nor BRK require an argument.

EXAMPLE

BAS ; Return to C-128 ROM operating system

BNK Specify the memory configuration that object code is generated into, as well as the configurations accessed by such commands as JSR, LOD, SAV, and PAGE. The user should take care not to confuse the terms "memory configuration" with that of "BANK" as employed by CBM BASIC programmers. When using this command, be sure to supply an operand that reflects the accurate value to be stored in \$FF00. That is, Machine Language programmers should refer to BNK \$3F where BASIC programmers will refer to BANK 1.

EXAMPLES

```
BNK $3E ; select RAM 0 plus I/O in the range D000-DFFF
BNK $3F ; select all RAM 0
BNK $7E ; select RAM 1 plus I/O in the range D000-DFFF
BNK $7F ; select all RAM 1
```

BRK This command is similar to the preceding command, BAS, but jumps off of MAS-128's internal BRK Vector [*See Memory Map*]. To enter the C-128 monitor program, the user should now hold down the STOP key after entering the BRK command. To restart MAS-128 after exiting, simply JMP \$2000. You should not re-enter MAS-128 via its Warm Start Address \$2003 unless you are absolutely sure that MAS-128's Zero Page and Common Routines and Buffers are untampered. Both BAS and BRK are commands which the User is urged to customize for his own particular system.

CMD This command has no arguments; it merely lists out the names of all legal MAS commands. It is probable that future versions of MAS will have more commands than those listed in the documentation. Information on these undocumented commands will be found on the accompanying master disk. Some commands will be reserved for the User to implement, such as NOP and USR.

COL This command suppresses the color discrimination facility of the assembler and text editor. In order to re-enable color discrimination you should provide this command with a non-zero argument. Programmers with monochrome monitors should customize their copies of MAS-128 so that the assembler comes up *colorless* on cold start.

COP Copy a single line, or a range of lines, as specified by line number, from one place in text to another. The user should also compare this command with the MOV, SWAP, and DEL commands. For example, the first command given below will copy the range of lines from 10 to 74 and insert them after line 211. The second example denotes the copying of a single line. Technical note: after being copied, the moved lines will be renumbered briefly to reduce text structure ambiguities.

EXAMPLES

```
COPY 10 74 211 ; copy range 10 to 74 over to 211
COPY 140 300 ; copy line 140 over to 300
```

DAT All files written are dated by month, day, and year. On Cold Start, the user is provided with the opportunity to set the date; the user may further redefine the date subsequently with this command. Illegal dates may be freely entered; it should be emphasized, however, that it is to the user's advantage to enter correct dates if only to keep track of the development of his source code. The Date is unique to 127 years of 1986. MAS Source and Object files are unique in that they are stamped with the current date when written or updated. Your copy of MAS-128 has its own peculiar date time-stamped internally for a default cold start. [See Appendix I.] You may wish to periodically edit this two-byte constant inside MAS-128 every couple months.

EXAMPLE

DATE: September 27, 1988 ; Matthew Montchalin's Birthday

DEL With this command the user may delete a single line, or a series of lines, of text. Once deleted, a line can not be recovered; it is permanently lost. Individual lines may also be deleted as a consequence of screen editing; simply enter an empty line having only the line number of the target line, and that line will be deleted. If you wish to delete the entire source file, you should use the NEW command instead of entering the command *del 0 65535*.

EXAMPLES

DELETE 1434 ; delete line 1434

DELETE 810 1340 ; delete lines 810 through 1340 inclusive

DIR This command is the one to use to read your disk directory. In the event of a DEVICE NOT PRESENT error message, you should redefine the device address (please refer to the section on Disk Support). The disk directory may also be read by entering the command @read "\$". These two methods of reading the directory each have their own advantages; by typing DIR you can specify file types and wildcards for selective file matching and searching. On the other hand, if you type @read "\$", the directory will be read as though it were a pure binary file- Selective file searching is disallowed, but if it's any consolation, unusual file types and even deleted file types *will* be reported faithfully.

EXAMPLES

DIR ; Display Directory

DIR "0" ; Display Directory of Drive 0

DIR "1" ; Display Directory of Drive 1

DIR "ab*" ; Show directory of all files whose names start with the characters *ab*

DIR "?" ; Show directory of all files whose names consist of just one character.

DIR "**,s" ; Show directory of all sequential files

@read "\$0" ; read directory of drive 0

@read "\$1" ; read directory of drive 1

DIS Disassemble an object file off diskette and generate source code straight into the text buffer. This command permits disassembly of both CBM object files and MAS object files at any arbitrary load address. MAS will assume you are disassembling a CBM object file if you fail to specify otherwise. Please note that the arbitrary designation *CBM* denotes those files stored on a Commodore Disk Drive, and whose initial pair of bytes are used for determining the LOAD ORIGIN. MAS-128 object files on the other hand use the first few bytes for describing date and file size. Furthermore, MAS-128 *executable* object files also fill out another 14 bytes to constitute a 16 byte identification header.

EXAMPLES

DIS \$8000 CBM "object file" ; disassemble Commodore binary object file as though it loaded at address \$8000

DIS \$CF00 MAS "another file" ; disassemble MAS binary object file as though it loaded in at address \$CF00

DIS "some file" ; disassemble a binary object file presuming it to be structured like any standard (Commodore) program.

DUMP Display memory in pure ASCII format. This Command requires an argument which will be taken as the base address for the memory dump. This command is intended primarily for reviewing with a quick glance any object code that has just been generated. To single-step through the Dump, simply press the ESC key. To abort, press STOP.

EXAMPLE

DUMP \$8000 ; display memory beginning with range \$8000

EDIT Search and replace any series of characters in the text buffer. When this command is entered, it requires an argument as the target of its search. The target of a search may have "wild cards", i.e., *any characters that are underlined or flashing*. To enter such "wildcard" characters, press CONTROL B or CONTROL O; to quit entry, press ESC B or ESC O (Refer to the Section on MAS ASCII). Note that the underscore character should not be confused with the hardware underlining of characters. After this argument is found, a control message is issued ("WITH") and subsequent input is required which will denote the modification or substitution. Note that the arguments should not be delimited by any special characters, such as quotes or other arbitrary delimiters; MAS-128 will recognize the arguments as being delimited simply by the first and last standard lowercase space characters of the line.

EXAMPLE

EDIT ora Sprite_1 ; search text for the words "ora Sprite_1"

WITH eor Sprite_2 ; and change 'em to read "eor Sprite_2"

EXT External Access. This command lets you generate code directly into another C-64 or C-128. In order to make use of this command you will have to construct an interface between the Master (which hosts MAS-128) and the Slave (which receives generated code). This is *not* an imposing task; you need only obtain a minimum of four wires and possess a small modicum of skill with a soldering iron. A couple card connectors would be ideal, but at worst, soldering the wires directly to the appropriate port pins is also feasible. Besides these four wires, you might also want to solder on some grounds to the appropriate ground pins of the ports designated below. [See *Accompanying Diagram.*]

Theory of Operation

Firstly, what the interface does is, join the Master and Slave together on a one to one basis by connecting just four lines: Cassette Write, Cassette Sense, User Port CNT2 Pin 6, and User Port SP2 Pin 7. The Cassette Lines are used for polling the status of transmission. Entire 8 bit bytes are transmitted serially across SP2 according to a synchronizing signal available at CNT2.

Protocol of I/O

As MAS-128 is called upon to store a single byte in the second pass of assembly, the External Output Driver takes over and delivers four bytes across SP2 to the Slave system, in this order: Object Byte, Address High Byte, Address Low Byte, and a concluding Checksum Byte. The Checksum Byte is arrived at by performing a Logical Exclusive OR of the previous three bytes. After transmitting these four bytes, the Master Cassette Write Line is set to 0 or 1 according to the state of Bit 7 of the Checksum Byte. If the Slave detects an error, either from a checksum mismatch or a disagreement on what state the Master's Cassette Write Line should be, it will request a repeat in transmission by setting the Cassette Sense Line high; otherwise, the Slave will set the Cassette Sense Line to a zero to signify that all is okay with the transmission so far.

Wedging the Driver

The Immediate Command *EXT* does require an argument- If the argument is non-zero, the normal MAS-128 object output routine will be wedged so that the desired output byte is delivered to the External Output Driver. To return MAS-128 back to normal, you will have to use an argument that evaluates out to zero; this will unwedge the External Output Driver. Future versions of MAS-128 may provide for the optional addition of a second argument.

Setting Timer A

When you supply the an argument to the *EXT* command, you should be aware that the evaluated result is used to set Timer A of CIA #2. Data is not transmitted through SP2 until the timer counts down and underflows. At this moment, data is synched out SP2 by a synchronizing signal available at CNT2. Ordinarily, a value of 4, 5, or 6 should provide you with a reasonable degree of speed and accuracy. If for some reason your transmission rate is error plagued, you should consider using a higher figure like 10 or

20. Also, you may wish to shield the cable with a ground. The greater the figure used for an argument, the slower the transmission rate. If transmission is unacceptably slow, or if you have some other sort of transmission trouble, please contact Mountain Wizardry Software for technical details regarding alternative transmission protocols.

EXAMPLES

EXT 4 ; Set up CIA #2 for fast transmission

EXT 16 ; Set up CIA #2 for slower transmission

Before using the EXT command you are presumed to have *already* installed code in the Slave to receive data transmissions. The installed code should implement a speed that matches that specified by the EXT argument. For instance, if you are generating code at 2 MHZ and delivering the code to a C-64 running at 1 MHZ, you should take this discrepancy into account when specifying the EXT argument. You should fine tune the Receiver Program found on your system disk by trying out different timer settings against that specified in the EXT argument so that the Timers decrement evenly and in tune.

Similarly, if MAS-128 is interrupted in the middle of a data transmission (as by pressing CTRL RESTORE], you should interrupt the Slave Syste and restart the Receiver program also.

FAST Speed up from 1 to 2 MHz. On Cold Start, the system will be running at a default of 1 MHz. If a 40 column monitor is connected, a message will be displayed to the user ("Sorry- 80 column RGB only"). In 2 MHz mode, the VIC is disengaged and the 40 column screen will be blanked. You may wish to flip back and forth between these two speeds depending on your needs. This command takes no arguments.

FIND Search for any arbitrary series of characters in the text buffer. Any flashing or underlined characters (entered by pressing CTRL O or CTRL B respectively) are reckoned as "wildcard" characters. After all of the text file has been searched through, the number of successful matches is displayed. Compare the *EDIT* command.

EXAMPLE

FIND eor sprite_color

JSR Call a subroutine in the memory configuration specified by the BANK command. By supplying arguments to this command, you can predefine the registers of the microprocessor before making the call. On return from the call, the register contents are displayed on the next line.

EXAMPLE

BANK \$3F ; All RAM 0, No ROM, No I/O
SET 127 AC ; Define symbol AC as \$7F
SET \$4D XR ; Define symbol XR as 77
SET %110 YR ; Define symbol YR as 6

JSR \$8022 AC XR YR ; call address \$8022 in Bank previously specified by the BANK command.

KEY Redefine Key String. MAS-128 supports 15 different key strings; each can be entered by simultaneously pressing a shift key and a key on the numeric keypad to the right of the keyboard (see Figure 1). To redefine any of these keys, you merely specify the number of the keystring and then provide an argument in normal ASCII form, delimited by the *first* and *last* quotes of the line. The length of any key string is limited to 15 characters.

EXAMPLES

KEY 1 "jsr \$"

KEY 2 "jmp \$"

KEY 3 "asl asl asl asl "

KEY 4 "lsr lsr lsr lsr "

KEY 5 "txa asl tax "

LINE Control output of leading decimal line number when listing text or reading text from disk. To suppress the line numbers, enter LINE 1; but to permit full expression of line numbers, enter LINE 0.

LIST So long as your source code is already present in the text buffer, you can list it to the screen with this command. (*If you want to list it to the disk drive, please refer to the TEXT command.*) Two arguments may be supplied to this command to specify the range of text to be listed. If these operands are not supplied, they will default to 0 and 65535 so that *all* of the text will be listed. You should note especially that the arguments *must* be delimited only by lowercase space characters.

The listing may be *frozen* by pressing the ESC key, *slowed down greatly* by pressing the back-arrow key, *slowed down slightly* by pressing the CONTROL key, and *aborted* by pressing the STOP key.

EXAMPLE

LIST 830 1160 ; List text lines from 830 to 1160 inclusive

LOD This command is *not* intended for use in loading source text into the text buffer; instead, it lets the user load for later inspection any file from disk into the bank of memory specified by the BNK command. Two file structures are supported, namely CBM and MAS. The file structure will default to that specified by the last LOAD operation. If a file structure has not yet been specified, then a Commodore file structure will be assumed.

EXAMPLES

LOD \$2000 mas "sprite object" ; load binary sprite data as
data to address \$2000

LOD \$C000 cbm "commerc_program" ; load obscure binary
code to address \$C000

LOD cbm "*" ; load commodore object
file to default origin

LOD mas "*" ; load mas binary object
file to default origin

MEM Display memory parameters for MAS text and symbol storage buffers. The Text Buffer is *always* in the Bank of Origin (refer to the WHR command if you can't remember which Bank you loaded MAS-128 into); the Symbol Buffer is set to the opposite bank. These buffers are both set to the ranges \$8000-F000 on cold start. The addressing ranges for these buffers may be redefined by supplying arguments to the command.

EXAMPLE

MEM \$A000 \$DFFF \$8000 \$9FFF ; set TEXT Buffer to \$A000-DFFF
and SYM Buffer to \$9000-9FFF

NEW Specify that the Text Buffer is effectively empty. All text previously held in the text buffer is regarded as erased with use of this command. However, any symbol table that may reside in the symbol buffer is *not* affected by this command; to erase a symbol table, it is necessary to assemble a *new* or empty file. This command ignores any supplied arguments.

NUM This command lets the user *renumber* all text in the text buffer by any increment up to 255, starting at a specific line. It's always a good idea to frequently renumber your text, *especially* after appending or merging files. (When you *MOVE* or *COPY* ranges of lines, some renumbering can also occur.) This command does not require an argument; if you do not supply an argument, renumbering will proceed according to the last specified renumbering increment. The default increment on cold start is 10. Do not specify a null increment.

EXAMPLES

NUM 10 50 ; Renumber text by ten's starting with line 50

NUM 100 ; Renumber source text by hundreds.

NUM ; Renumber source according to last renumber
increment.

PAS If you don't wish to bother with the first pass of assembly, when the symbol the symbol table is generated, you can skip to the second pass with this command. This is particularly useful in long reassemblies where only a single variable need be redefined for each of several different passes.

EXAMPLES

PAS ; Proceed to second pass for text (already loaded)

PAS "0:file name" ; Get specified file into memory
and then proceed to second pass.

RES On occasion you may deem it necessary to execute a Cold Start *especially if you feel that system variables have somehow been trashed or poisoned* by errant user routines or co-resident programs. This command executes a simple Cold Start. If the RES command fails to make the mark, something is gravely wrong and you may wish to resort to using an NMI keypress combination, given below:

<u>KEYS PRESSED</u>	<u>EFFECT</u>
ESC + RESTORE	Cold Reset
CTRL + RESTORE	Warm Reset
STOP + RESTORE	Forced Break to Monitor

SAV This command is not intended for saving source code from the text buffer; you should refer to the section on disk support for that- Rather, this lets the user save object code from RAM in either CBM or MAS format. Programmers who wish to use MAS object files should use this command. For example, you may wish to convert your Sprites and Character Sets from CBM binary format over to MAS binary format. You would do so by first LODing CBM structured binary files to RAM and then SAVing them as MAS structured binary files.

EXAMPLES

SAV \$4000 \$6FFF cbm "file name" ; save range of memory
in CBM structure

SAV start limit mas "file name" ; save range of memory
in MAS structure

SET Sometimes you may wish to define or redefine the value of a symbolic label without resorting to a full-fledged reassembly. With this command a symbol table can be enlarged manually simply by setting the values of symbols one at a time.

EXAMPLES

SET 127 sprite_variable ; impart constant value of \$7F
to the symbolic label called
"sprite_variable"

SET work+5/\$100 test ; assign the result of a complex
evaluation to the symbolic
label called "test"

SLOW Reduce speed back to the default 1 MHz. If the 2 MHz mode is too fast for you, you can always use this command to slow it down by one half. *Note that pressing CONTROL RESTORE will drop you back to 1 MHz also.* When the processor is running at 1 MHz, the VIC chip is enabled; if you have also connected a 40 column monitor to your system, you will find that you can observe object code stored at RAM \$0400 in memory bank 7F. This may constitute some reason for occasionally dropping down to 1 MHz.

SWAP Interchange any pair of text lines currently in the Text Buffer.

SYM Display Table of Symbolic Labels. Symbolic labels are reckoned unique not simply by their spelling, but by case, color, and address. It should be emphasized that a symbolic label spelled with upper-case characters is *not* confused with another label spelled with lowercase characters, i.e., *ABCD* is not confused with *abcd*, nor is it confused with another word with identical spelling but of a different color. File corruptions (or files which are not MAS source code) can be readily ascertained through the display of odd colors, flashing characters, and beeps (CTRL G's) on listing.

This command permits use of an argument whose lowest three bits are significant.

EXAMPLES

SYM %000	; list all of symbol table
SYM %001	; list zero page symbols only
SYM %010	; list absolute symbols only
SYM %011	; list zpg symbols <i>of same color as argument</i>
SYM %100	; list abs symbols <i>of same color as argument</i>

TAB Set tabulation point by entering a number. Tabs can also be set by pressing CTRL K and cleared by pressing CTRL L. Tabs are useful in adjusting the format for listing source code either to screen with the LIST command or to the disk drive with the TXT command. You can also press the TAB key to advance the cursor to the next TAB point.

EXAMPLE

TAB 10	; set tab to the 10th character from the left of the screen
---------------	--

TXT Convert source code into text file. Any text currently resident in the text buffer is translated and listed to disk. The command requires an ASCII statement as an argument which will be taken for use as the name of the new disk file. This facilitates the conversion of MAS-128 source code to ASCII text files which may be modified by other assemblers. Please compare this command, used in immediate mode, with the *TXT* pseudop, used in assembly mode.

EXAMPLE

TXT "0:file name" ; convert source code to ascii text
file and assign it the name that's
specified in the ASCII statement

VID Redefine 8563 video controller variables. This command can take up to five arguments, namely, screen color, number of rows, starting screen address, starting attribute address, and vertical fine adjust. Most 80 column RGB monitors can support quite well a selection of 26, 27, or 28 rows, and this says something about Commodore's choice of defaulting to 25 rows. *The enterprising programmer may wish to modify the values found in the screen initiation routine of his copy of MAS-128 so that his favorite colors or screen parameters will be used on Cold Start.*

EXAMPLES

VIDEO 1 ; select white screen

VIDEO 0 25 \$0000 \$0800 \$20 ; select default values

VIDEO 6 27 \$0000 \$0C00 \$1D ; select blue screen color, 27 rows,
screen start address at \$0000,
attribute start address at \$0C00,
and vertical fine adjustment of \$1D.

NOTE- It's a good idea to clear the screen after editing any of the 8563 internal variables, especially after changing the number of rows. Flashing characters at the bottom of the screen are natural, usual, and to be expected after changing the number of rows or moving the screen around. This should not constitute reason for alarm.

WHR Display which memory configuration was current when MAS-128 was cold-started. In essence this indicates the memory bank MAS-128 found itself in on cold start. If you mean to employ MAS-128 from memory bank 7F, you should take care not to generate code in the area of common ram used by MAS-128 for text and symbol manipulations. If in such a case you do generate code into common RAM, you should trigger a Reset by tapping ESC Restore.

Pseudo Operation Codes

Pseudo-operation codes, or *pseudops*, are commands embedded in the source code that serve as directives that are to be acted upon by the assembler during assembly and which can represent series of operations or instructions, perhaps to redirect or modify output, to read data from disk, to perform operations on code, or to store data or code in ways peculiar to the purposes of the programmer.

Some (if not most) of these pseudops require an argument or two (or more) to serve as *pseudo-operands*. When entering a series of arguments to a pseudop, you should take care that each is delimited *only by lowercase space characters*. Commas, colons, semicolons, apostrophes, or uppercase space characters should *never* be used as field delimiters. Uppercase spaces may occasionally be entered inadvertently with the intention of separating words, but they will *not* be interpreted as proper delimiters and will cause the two words to be considered *joined and one*. The accidental entry of such uppercase spaces or improper delimiters will almost always result in an assembler error. However, such characters are not without their virtues; they may be employed freely, if deliberately, in ASCII strings or in comment data.

In the examples of source code below, the word "skip" as a symbolic label is meant to be taken merely as an example of an arbitrary name for the destination of a relative branch. The programmer need not supply this word with the pseudop in order to make use of the pseudop.

LO@ Store low byte of evaluated expression. Requires one argument.

HI@ Store high byte of evaluated expression. Requires one argument.

LH@ Store low byte then high byte of evaluated expression. Requires one argument.

HL@ Store high byte then low byte of evaluated expression. Requires one argument.

DN@ Decrement a 16 bit variable in memory. The variable should be in low-byte high-byte order. You should try to avoid using this pseudop with the indirect addressing modes since 16 bit integers may be stored in highly irregular ways when referenced indirectly. Requires one argument.

EXAMPLE 1A

dn@ variable

duplicates the following instruction sequence

```
lda variable
bne skip
dec variable + 1
skip dec variable
```

EXAMPLE 1B

dn@ table,x

duplicates the following instruction sequence

```
lda table + 1,x
bne skip
dec table + 1,x
skip dec table,x
```

UP@ Increment a 16 bit variable in memory. The variable should be in low-byte high-byte order. Requires one argument.

EXAMPLE 1A

up@ variable

duplicates the following instruction sequence:

```
inc variable
bne skip
inc variable + 1
skip
```

EXAMPLE 1B

up@ table,x

duplicates the following instruction sequence:

```
inc table,x
bne skip
inc table + 1,x
skip
```

ASC Store a series of ASCII characters in memory. *Compare also the SCR pseudop.* Requires one ASCII statement as an argument. **IMPORTANT-** Only the *first* and *last* quotation marks of the text line are regarded as the delimiters to the ASCII string. Therefore, *everything* between those two quotes is taken as the body of the string-- escape codes, carriage returns, color codes, and even other (internal) quotes and graphics characters.

BEP Evaluate an expression and use it to generate a musical note. This command permits use of several arguments. Its intended use is to flag the user regarding a particular point during assembly, perhaps when diskettes need to be swapped. [Now, to make a single sound, you must remember, your C-128's audio-output should be plugged into the audio-input of your RGB 80 column monitor.]

EXAMPLES

bep 1 2 3 4 5 6

bep 6 5 4 3 2 1

bep 10 11 12 10 11 12 10 11 12

dsk msg "insert next disk now"

NOTE- Each byte is interpreted according to the following scheme when generating music:

Bits 0-3 Which of sixteen frequencies

Bits 4-5 Which of four durations

Bits 6-7 Which of three voices

Exception- If both bits 6 and 7 are set, then no note, just a longer pause.

BIN Store binary data in memory. Requires at least one argument. The argument(s) should *not* be preceded by the percent key, but should consist simply of the digits "0" and "1".

EXAMPLES

bin 00010100 ; store a constant \$14 in memory

bin 1010 ; store a constant \$0a in memory

bin 1 10 100 1000 10000 ; store constants 1, 2, 4, 8, 16

bin 01 10 01 00 10 00 00 ; store constants 1, 2, 1, 0, 2, 0, 0

CHK Check one argument against the other. Requires two arguments.

EXAMPLE 1A

chk job work

duplicates the following instructions

```
lda job
cmp work
```

EXAMPLE 1B

chk (job),y work,x

duplicates the following instructions

```
lda (job),y
cmp work,x
```

EXAMPLE 1C

```
chk zero_page,x absolute,y
```

duplicates the following instructions

```
lda zero_page,x  
cmp absolute,y
```

EXCEPTION: If the first of the two arguments is IMMEDIATE addressing, and is reckoned as ABSOLUTE, a 16 bit comparison is yielded:

```
chk #1234 abcd
```

duplicates the following instructions

```
lda #12 ; high byte  
cmp abcd + 1 ; high byte  
bne skip  
lda #34 ; low byte  
cmp abcd ; low byte  
skip
```

DAT Display the current date. This reflects the date that you entered when starting up MAS-128. You should note that this pseudo-opcode *outputs* the date internal to all MAS files. To set the date for any given file, please refer to the immediate command of the same name, DATE.

DCM Store an 8 bit byte of an evaluated decimal expression. Requires at least one argument.

EXAMPLES

```
DCM 1 3 7 11 13 17 19 23 ; store first 8 primes
```

```
DCM 8192/256 ; store the high byte of the result  
of the evaluated expression
```

DEV Specify which of 32 different serial devices MAS-128 will communicate with. This command lets you switch device numbers in mid-assembly. Requires one argument, whose lower 5 bits are significant. Ordinarily most disk drives are device 8 but some users have a couple extra drives. Spare drives tend to have device numbers (*primary addresses*) 9 or 10. A special case is made for device 0, however. When you specify device 0, you access the 1700/1750 RAMDOS that is internal to MAS-128.

DSK CMD Send a command to disk drive. Requires an ASCII statement as an argument. Note that this pseudo-operation is comprised of two pseudops, namely, *dsk* and *cmd*. They should always be entered with a clear visible space between them.

EXAMPLE

DSK CMD "initialize1"

DSK MSG Prompt user with message. Actually, all this does is simply flash the specified ASCII string, waiting for a key press. It's intended use is for swapping diskettes but may be employed for other purposes requiring a delay. The main difference between this pseudopair and the simple msg command is that the string is flashed and the system waits until it detects a keypress.

EXAMPLE

```
100      DEV 9
110      DSK MSG "Remove Source Disk from Device 9, Drive 1"
120      DSK MSG "Insert Object Disk into device 9, Drive 1"
130      DSK CMD "initialize1"
140      OBJ "Sprite Set 1"
150      OBJ "Sprite Set 2"
160      OBJ "Sprite Set 3"
170      OBJ "Sprite Set 4"
180      OBJ "Sprite Set 5"
190      OBJ "Sprite Set 6"
200
210      BEP 17 48 65 13
220
230      DSK MSG "Return Source Disk to Device 9, Drive 1"
240      DSK CMD "initialize1"
```

DST Define Storage area. Advance PC pointer according to the evaluated expression. The memory so skipped remains undefined and unaltered. You might wish to use this pseudopair to reserve storage area. If you require *constant* data in a particular storage area, you might want to skip using this command and instead save the constant data on disk as a *mas* object, bringing it in as needed. Requires one argument.

END Quit Phase and go to next phase of assembly. If encountered in pass 1 of assembly then MAS-128 will proceed to pass 2; otherwise, assembly will be taken to have been completed and so end.

EQU Assign or *equate* the result of the argument to the preceding symbolic label. Requires one argument. Note that it is good form to define your zero page symbolic labels before referencing them. Symbolic labels which are taken to be *absolute* may *not* be used in operations that are reserved for zero page. *Local* labels should not be defined using the *EQU* pseudopair. Similarly, *absolute* or *zero page* labels should not be defined in whole or in part by *local* labels.

EXAMPLES

```
start_of_code EQU $0088 ; absolute definition
same_address EQU $88 ; zero page definition
another_place EQU work+job ; absolute if either work
```

or *job* is absolute

HEX Store hexadecimal constant data in memory. The dollar sign should *not* be used with the hexadecimal expressions. Requires at least one argument. The argument will be evaluated preferentially a byte at a time. Where otherwise only a *nybble* is given, the data stored will have an upper nybble equal to zero. In the examples given below, it is the very *last* example that will be parsed and assembled the fastest. However, for the sake of clarity you may wish to go with the *first* example. The *second* example is probably the *worst* in terms of clarity.

EXAMPLES

HEX 00 01 02 03 04 05	; store bytes 0, 1, 2, 3, and 4 in memory in that order.
HEX 0 01 2 03 4 05	; store the same bytes in memory as in the example above.
HEX 0 1 2 3 4 5	; store the same bytes in memory as in the example above.
HEX 0001 0203 0405	; store the same bytes in memory as in the example above.
HEX 000102030405	; store the same bytes in memory as in the example above.

INP Solicit input from the user during the first pass of assembly. This command lets the user define his symbolic labels in mid-assembly. The symbolic label should always precede the INP pseudop. The cursor will blink and permit the user to enter data. After the user finishes with a return key, the expression will be evaluated and assigned to the symbolic label. In order to abort the input, simply press CONTROL RESTORE.

EXAMPLE

```
MSG "Enter value for alpha"
alpha INP
```

LNK Link file off diskette. Note that only master files (as specified by the MAS pseudop) may process linker files from diskette. Other pseudops dealing with disk access are DEV, OBJ, SYM LOD, SYM SAV, TXT, DSK CMD and DSK MSG. Note that one argument, an ASCII statement, is required. Carriage returns, null characters, and reverse ASCII characters may be embedded in the file name, but you should refrain from using question marks, colons, or asterisks since they are ordinarily reserved by CBM for special use by the Disk Controller of your disk drive.

EXAMPLE

```
MAS ; permit file linkage
```

LNK "0:definitions" ; bring in equates file
 LNK "0:body 1" ; do first file of main body
 LNK "0:body 2" ; do second file of main body
 LNK "0:body 3" ; do third file of main body
 LNK "0:end" ; do last file

MAS Specify file as master file. This is one of the pseudops that should be placed on the first few lines of a file. Master files should not be subordinated to other master files via *file linkage* (refer to the LNK pseudop), as it may lead to infinite linking, as when a MASTER file links itself, for example, or when a subordinated master attempts to link a calling master. However, in a few rare programming conditions this may not necessarily constitute an error, so permission is requested from the user before linking to another master file. Files not specified as MASTER files with the MAS pseudop are presumed to be ordinary (slave) files.

MOV This pseudop denotes a data transfer via the accumulator from the origin to the destination; it is nothing more than a convenient way of denoting a LOAD and STORE of the accumulator. Two arguments are required.

EXAMPLE

MOV origin destination
 MOV zero_page,x absolute,y
 MOV (zero_page),y (zero_page),y

The above instructions will yield the same object that the following instructions will yield:

LDA origin STA destination
 LDA zero_page,x STA absolute,y
 LDA (zero_page),y STA (zero_page),y

EXCEPTION- If the *first* argument involves immediate addressing, and if the operand is also evaluated out to be *absolute*, then a 16 bit data transfer will be effected.

16 BIT EMULATION

MOV #\$C000 vector
 MOV #\$1300 RES_vector
 MOV #500 counter
 MOV #3000 revolutions

will yield the same object as

LDA #C0	STA vector + 1	; high byte first
LDA #00	STA vector	; low byte second
LDA #13	STA RES_vector + 1	; high byte first
LDA #00	STA RES_vector	
LDA #500/2	STA counter + 1	; high byte first
LDA #500	STA counter	; low byte second
LDA #3000/2	STA revolutions + 1	; high byte first
LDA #3000	STA revolutions	; low byte second

MSG This pseudop is meant to be used with the INP pseudop for prompting the user for input. It can also be used to display any other messages. Please compare this command to the DSK MSG command above.

EXAMPLE

MSG "Now loading character set data"

OBJ "character set"

OBJ This pseudop permits the loading of specially prepared (MAS format) Binary Object Files into memory. You don't have to use a "master" file (specified by the MAS pseudop) in order to use this pseudop. In order to create *prepared* MAS object files, you'll want to LOD them into memory as Commodore Binary Files, then SAV them back to disk as MAS Binary Files.

EXAMPLE

OBJ "sprite data 1"

OBJ "sprite data 2"

OBJ "sprite data 3"

ORG Specify origin of assembly for pseudo "program counter". Unlike some assemblers, you can have several origins in the course of your assembly, or none at all. When assembly begins, the PC defaults to \$0200 with object output suppressed. This is an innocuous address if MAS-128 is executing from MMU Configuration \$3F; code would be generated into MMU Configuration \$7f *underneath* Common RAM. If you *are* assembling in sensitive locations such as zero page or the stack, you should be sure to relocate any object output over to a safe area with the REL pseudop (q.v.).

OUT Set output mode for assembly. This pseudop requires one argument; only the bottom nybble of which is significant. Bit 3 is intended for later versions which may implement the 65C02 or 65816 instruction set; for the while, however, it must remain

reserved for expansion. Only the bottom nybble of the argument is significant:

Bit	Function
3	Reserved for Expansion
2	1 = Screen Output, 0 = No Screen Output
1	1 = Object Output to Memory, 0 = No Object Output
0	1 = Fast 2 MHz Assembly, 0 = Slow 1 MHz Assembly

EXAMPLES

```

out 4      ; Display listing but suppress object output, go slow
out 3      ; Suppress Listing, output object, proceed fast
out 2      ; Suppress Listing, output object, go slow
out 1      ; Suppress Listing, suppress output, proceed fast

```

REL Specify origin for generation of relocated object code. This pseudop is just like the **ORG** pseudop except that it has no effect until the second pass. You should use this pseudop immediately after the **ORG** pseudop. Please study the example of source code given below:

EXAMPLE

```

                ORG $50      ; start in zero page
                DST End_of_Reloc - Elsewhere
Start          ; save current PC in label "Start"
                REL $9A40    ; but store it elsewhere
Elsewhere      ; save new PC in label "Elsewhere"
                LDA #$00     ; example of normal source code
                LDX #$01
                LDY #$02
                NOP
                NOP
                NOP
End_of_Reloc
                ; Now Resume
                ORG End_of_Reloc - Elsewhere + Start

```

REV Reverse object generation. This allows names, ASCII text, sprite data, etc., to be spelled/stored *backwards* as generated. A series of reversed bytes may not exceed \$7FFF. The whole point of reverse generation is the implementing of structures which save both bytes of memory and clock cycles in the manipulating, or *moving*, of such ranges of data- In the example below, two subroutines are given which both print the same ascii string. Because of program flow, the second routine in the sample source code below will *always* take longer than the first to execute. Can you see why?

SAMPLE SOURCE CODE

```

100print_name1
110          ldy #length
120          lda string_start-1,y
130          jsr chROUT
140          dey
150          bne =-7

```

```

160          rts
170
180print_name2
190          ldy #0
200          lda string_start-1,y
210          jsr chrout
220          iny
230          cpy #length bcc =-9
240          rts
250
260length          equ string_end-string_start
270
280          rev length
290string_start
300          asc "This is an ascii string"
310string_end

```

SCR This pseudop is similar to the ASC pseudop. It also requires an argument which is delimited by the first and last quotation marks of the text line. However, when it comes to storing the bytes in memory, it stores in memory the *screen codes* and *not* the CBM ascii values for the characters in the argument.

SYM LOD This program directive consists of *two* mnemonics that *must* be separated by a simple lowercase space when entered on the keyboard. It allows for the loading from disk of a [sequential] table of symbolic labels. One argument is required, the file name, delimited by the first and last quotes of the text line.

SYM SAV This directive is the complement of the SYM LOD command given above, and is used conversely for *saving* a table of symbolic labels to disk. Just like SYM LOD, this pair of mnemonics must also be separated with a lowercase space when entered on the keyboard. A filename must be supplied for use as an argument.

TOK Not implemented. On the C-64 version, this pseudop stored the BASIC 2.0 one-byte token of the command given in quotes. Since exceedingly few programmers used this feature of MAS-64, and since MAS-128 makes no presumptions that the individual developer deigned to retain his BASIC ROM [where the table of BASIC tokens are stored], the author saw fit to leave this pseudop unimplemented. Enterprising programmers may wish to implement this pseudop or otherwise substitute their own pseudop in place of this one. Please contact *Mountain Wizardry Software* for technical details.

TXT Create a pure ASCII text file on diskette. This requires an ASCII argument delimited by the first and last quotes of the text line. If the assembly output mode has been set to output the text listing to the screen, then those same characters will be delivered to the disk drive to generate an ASCII text file. *Care should be taken in constructing a pure ascii text file when more than one drive or more than one diskette are being used.*

EXAMPLE

```

100  MAS          ; permit file linkage
110  OUT 4        ; listing turned on

```

```
120  TXT "file name" ; create ascii file
130  ORG $2000      ; start at an arbitrary address
140
150  LNK "file 1"   ; assemble several files
160  LNK "file 2"
170  LNK "file 3"
180  LNK "file 4"
190
200  END
```

The purpose of the TXT pseudop is to facilitate printer support of MAS source code by sending all LISTED text to the disk drive. On creation of the pure PETASCII text, you can use your favorite word processor to rewrite & polish things up for final printing.

USER EXPANSION

MAS-128 is an open-ended system!

You can expand it either by employing MAS USR EXEC files in the *immediate* mode or by employing USR Expansion pseudops in the *assembly* (programming) mode. Up to Twenty-Six such pseudops may be implemented by installing a custom-programmed user-expansion module in address \$C000. (The actual address for the module can be elsewhere.) During assembly, this module will be checked once per pass, every time a USR-op is encountered in the source code.

USR PSEUDOPS

```
A@@ B@@ C@@ D@@ E@@  
F@@ G@@ H@@ I@@ J@@  
K@@ L@@ M@@ N@@ O@@  
P@@ R@@ S@@ Q@@ T@@  
U@@ V@@ W@@ X@@ Y@@  
Z@@
```

Following the date and size stamp, the first sixteen bytes of the user expansion module should consist solely of the letters "mas usr routines" with bit 7 clear and all the letters in lowercase PETASCII. The entry point to the module should then be at address \$C010. All USR-pseudop calls to the expansion module are conducted to this point in memory. To ascertain which of the USR-pseudops brought about the entry to the user expansion module, the entry point should read like this:

```
lda optok sec sbc #37 tax
```

After which XR will contain a value from 0 to 25. If you are using the BREAKER module described elsewhere in this manual, you can then use the BRA INDIRECT instruction and simply provide a table of 26 immediate vectors to direct assembler processing to the correct expansion routine. To effect the return from the expansion routine, a simple RTS instruction should be executed.

Some degree of error checking is provided when using these pseudops, such as ensuring the use of a zero page or absolute argument for pseudops a@@ through y@@ (but pseudop z@@ requires an ascii argument), but the programmer is ultimately responsible for determining his own error conditions. *For details in checking for legal addressing modes for these pseudops please contact Mountain Wizardry Software for technical details.*

SAMPLE CODE FOR USER EXPANSION MODULE AT \$C000

```
10          org $C000  
20          asc "mas usr routines"  
30          bra (vector_table,x)  
40vector_table  
50          lh@ subroutine_1  
60          lh@ subroutine_2  
70          lh@ subroutine_3  
80          lh@ subroutine_4  
90          lh@ subroutine_5
```

Expansion with USR Exec Files

When you're in the Immediate Programming Mode, that is, not assembling source code, you may wish to avail yourself of features not regularly provided in MAS-128 proper. This is made possible by pulling specially prepared MAS USR EXEC files in off disk, executing them, and then returning back to MAS. Do not call MAS-128 internal routines except through such expansion files.

Note that MAS-128 presumes all MAS USR EXEC expansion files will execute with MAS-128 Ze Page and Privileged areas *resident*. In other words, if you write your own MAS USR EXEC files, be sure not to tamper with Zero Page, the Stack, Common RAM \$0380-\$0AFF, or High Privileged RAM \$FD00-FFFF or you will quite possibly bring ruin to MAS-128 on return.

MAS USR EXEC files have the following structure:

```
Bytes 0-1  Size
Bytes 2-3  Date
Bytes 4-5  Origin
Bytes 6-7  Limit
Bytes 8-F  asc "MAS EXEC"
Bytes 10+  Executable Object Code
```

The following is an example of the source code for a trivial expansion file. Each time you execute it the VIC screen border (\$D020) will change.

MAS EXEC USER EXPANSION FILE STRUCTURE

SAMPLE SOURCE

```
10          org $BFF0    ; Make room for USR Header
20          rel $C000    ; Store it at $C000
30          out 3
40Header
50          lh@ End_of_file - Start_of_file
60          lh@ $1234
70          lh@ Start_of_file
80          lh@ End_of_file
90
100         asc "mas exec"
100Start_of_file
110         lda $ff00 lsr as; Bring in I/O
120         inc $d020    ; Increment border color
130         sta $ff01    ; Return to home config
140         rts
150End_of_file
```

SOURCE FILE CONVERSION

The author recognizes that some programmers have considerable libraries of assembly language files and modules that are currently incompatible with the file structures required of MAS-128 source files. For instance, all three-byte mnemonics are tokenized for more compact file storage and faster parsing. More over, all discreet elements of text are prefixed with invisible pre-processing codes.

To facilitate conversion of these vast libraries, MAS-128 contains a number of built-in file conversion utilities.

- ASC Converts PETASCII files to MAS source structure. For proper conversion, the lines should be numbered and terminated with carriage returns
- CBM Converts PETASCII files to MAS source structure. The source lines should be unnumbered and terminated with carriage returns
- MAE Converts USA ASCII files to MAS source structure. The first two bytes of each lines is evaluated as *binary coded decimal*. Succeeding characters are taken to constitute the main body of source for the line, until terminated by a character whose 7th bit is set.
- PAL Converts files that are structured like Commodore BASIC. Tokens will be untokenized. The equal sign and a few other pseudops will be replaced with their MAS equivalents. Lines should be delimited with null bytes.
- SCR Converts Commodore screen code files to MAS source structure. Lines should be terminated with null bytes.
- USA Converts USA ASCII text files to MAS source structure. Lines should be terminated with carriage returns.

On the other hand, converting MAS source code back to the structures employed by PAL or MAE is much like changing a butterfly back to a caterpillar. For those of you who, for some unknown reason, really need to step down one rung on the ladder of excellence, behold here the method to your madness:

To begin with, you'll have to generate an ASCII text file. The easiest way on the C-128 is by using the TXT command in the immediate mode. First load the victim MAS file into memory and then enter: TXT "file name". You will see everything list down the screen with the disk humming away as the new file is created. After it's done listing, you will have a new file on disk. This new file, on investigation with any old disk utility will reveal an interesting file structure. -It's mostly simple PETASCII with carriage returns as delimiters between text lines. You can display this ASCII file by using the *@display "file name"* command as given in the section on immediate commands.

```
100; Source Code for Converting TXT to SEQ
110
120                   sym lod "mas library"
130start_here
```

```

140          lda #$93 jsr chrout ; clear screen
150
130          mov #0 status
140
150          jsr open_read_file bne eh?something_went_wrong
160          jsr open_write_file bne eh?something_went_wrong
170
180; _____
190
200read_line
210          lda devnum jsr TALK ; okay, device, you can
220          lda #$62 jsr TKSA ; go spill your guts now
230
240readbyte
250          jsr ACPTR jsr chrx
270          lda status bne write_line
290          cmp #$0d bne readbyte
280          bit quotflag bmi readbyte ; embedded carriage return
300
310write_line
320          jsr UNTALK
330
340          lda devnum jsr LISTEN
350          lda #$63 jsr SECOND
360
370          jsr readline
360          jsr getlength
370          iny
380          cpy tab =-9
390
400          lda #$00 jsr CIOUT ; end of line terminator
410          lda #$0d jsr CIOUT jsr CIOUT jsr CHROUT ; dummy link
410
420          jsr UNLISTEN
430          lda status bne eh?something_went_wrong
440          jsr checkSTOP bpl read_line
450          rts
460

1000open_read_file
1010          clc hex 24
1020open_write_file
1030          sec
1040          ror namflag ; carry indicates which name
1050          lda devnum jsr LISTEN
1060          lda #$f2 bit namflag bpl = +4
1070          lda #$f3 jsr SECOND
1080
1090          ldy #length1 bit namflag bpl = +4
1100          ldy #length2
1110sendloop
1120          lda nam1,y bit namflag bpl = +5
1130          lda nam2,y jsr CIOUT
1150          dey
1160          bne sendloop

```

```
1170          jsr UNLISTEN
1180
1190          lda status
1200          rts
1210
1220nam1      rev length1
1230          asc "name of read file"
1240          asc ",p,r"
1250
1260nam2      rev length2
1270          asc "name of write file"
1280          asc ",p,w"
1290nam3
1300
1310length1   equ nam2-nam1
1320length2   equ nam3-nam2
```

