



Fine Systems Designs

Mountain Wizardry Software

P.O. Box 66134
Portland, Oregon 97266
(503) 265-2755

C-128 MIDNIGHT ASSEMBLY SYSTEM

Legal Status

Mountain Wizardry Software holds all reproduction rights over the Midnight Assembly System for the Commodore 128 (MAS-128) and attendant documentation. This pre-release alpha copy is not for general distribution. It has been provided solely for technical review.

Any attempt at mass distribution will be a violation of the author's proprietary rights. Bona Fide Recipient Beware- This copy *may* have been stamped electronically for serial identification purposes!

Amnesty

If you have inadvertently received an illicit MAS-128 alpha copy, you can make amends for your foul deed by paying the author \$2.25 (USA currency) as a consolation fee. Be sure to include your area code and voice phone number with your mailing address. Also please specify whether you are interested in beta testing professional or recreational software.

Special Features

- * Absolutely *no* ROM routines called! This feature alone makes it indispensable to all of you contemplating the development of custom replacements for any or all of the internal ROMS! (You know who you are!)
- * Contains its own built-in 1700/1750 RAMDOS, *complete with upload & download commands!*
- * Executes from RAM 0 *or* RAM 1 --- Generates code into opposite Bank of Memory. If you've rigged your old assembler to generate code into another slave C-128 or C-64 via cabling, well, that can be done with MAS-128 too!
- * Multiple statements per line of text! Free-Format Text Entry!
- * Sorry, 80 Column RGB Only! Default Screen is 80 columns by 26 rows. If you have a good monitor, you can pump it up to 32 rows. If you design applications for the 40 column VIC-II chip, you are seriously advised to use two monitors simultaneously.
- * Full Screen Scrolling Text Editor
- * Extensive Disk Support. Burst-loading with 1571/1581 disk drives!
- * Copious Doc's- Over 100 pages of regular doc's plus 220 pages of actual source code for MAS-128! (One of the delays in issuing the doc's is the truly great difficulty in finding a C-128 word processor that fully supports Laser Printers.)

GETTING STARTED

The disk is *not* autobooting! While an autobooting feature might be nice for users, it is a source of frustration for programmers who are always hitting the RESET button.

In order to get started, follow these steps:

- 0) You May Wish to Power All Devices Off
- 0) You May Wish to disconnect ROM-Dependent Peripherals
- 0) You May Wish to connect a 1700 or 1750 Expansion RAM
- 1) Turn a disk drive on; it need not be device 8
- 2) Turn the Computer on pressing *[STOP]* simultaneously
- 3) Insert MAS-128 System Disk into Drive
- 4) Load the program into a Bank of RAM
- 5) G 1700 (i.e., JMP \$1700)
- 6) Witness Credits & Flash
- 7) Enter Current Date (or Accept Current Date)
- 8) Press HELP key to Display HELP File

Now you are home free! In addition to pressing the HELP key you may also wish to review some of the source files included on the diskette. Simply employ the immediate command *@read "0:file name"* to read a source file. If you inadvertently try to review an object file with the *@read* command you should expect to see lots of garbage on the screen. That's okay- just press CONTROL RESTORE to clean up the MAS-128 Screen Editor, then press CTRL 2 to restore the white cursor.

MAS-128 contains its own proprietary RAMDOS for owners of the 1700 and 1750 RAM expansion units. It is not desirable to load in another RAMDOS. Developers may wish to license this RAMDOS at *really* cheap rates.

It may be desirable to see what happens when you leave device 8 turned off and instead use a 1571 disk drive with its dipswitches set to indicate device 9. On Cold Start MAS-128 will seek out valid serial handshakes and accept as its main drive the first device it finds.



Fine Systems Designs

Mountain Wizardry Software

P.O. Box 66134
Portland, Oregon 97266
(503) 265-2755

These MAS-128 Alpha Notes should *not* be taken to represent the actual doc's for MAS-128. Instead, these notes are here to expedite the 6502 assembly language programmer's familiarization with the MAS-128 6502/8502 programming environment. This Pre-release of MAS-128 is based on its predecessor, the *C-64 Midnight Assembly System* and as such shares a number of similarities, but is considerably more versatile and comprehensive. Reviewers (and Previewers) should *please* peruse the MAS-64 Doc's before reading these notes. Some of the notable differences between MAS-64 and MAS-128 will be detailed below-

A FEW OF THE IMMEDIATE COMMANDS

- BNK** Select memory configuration. Please don't confuse this with the CBM BASIC command of the same name. This command specifies the value to be stored in \$FF00. For example, *Bnk \$3F* selects RAM 0 and *Bnk \$7F* selects RAM 1. You would use this command in connection with the JSR and DUMP commands. Ordinarily, MAS-128 generates code into the bank of RAM opposite to the one it resides in. You can change things around using this command. If you're not sure which Bank of RAM you're working in, the *WHR* command will tell you.
- CMD** Display a list of commands currently implemented by MAS-128. Some of the commands listed are reserved for expansion and may be implemented by the technically advanced user plugging the address of his own subroutine into the Immediate Command Vector Table. For example, the 3 letter immediate commands *Nop* and *Usr* will always be left open for definition by the user.
- VID** Review/Adjust Variables employed by the MAS-128 Screen Editor. The Screen Editor is restored to these defaults when you press [CONTROL RESTORE]:
- VID 0 \$1A \$0000 \$0C00 \$20
- VID [*Color*] [*Rows*] [*Screen*] [*Attributes*] [*Vertical Fine Adjust*]

A FEW OF THE PSEUDO-OPCODES

- REL** Relocates code. This is just like the *org* pseudop but affects output on the second pass. An argument must be supplied.
- DEV** Changes serial device numbers in mid-assembly. *Dev 0* directs all disk access to the internal 1700/1750 MAS-128 RAMDOS. The argument may range from device number 0 to 30. The usual device number (primary address) is device 8.
- DSK MSG** Prompts user to swap diskettes for *big* assemblies. You should be sure to

supply an ascii statement delimited by quotes to serve as the argument. Please remember that ascii statements are defined by the *first* and *last* quotes of the line: carriage returns, cursor controls, nulls, control codes, and internal quotation marks may all be embedded within the ascii statement. These embedded control codes will *not* scramble the listing.

DSK CMD Issues disk commands. For example, *dsk cmd "i0"* initializes drive 0 of the currently defined device. This command may be of some value when assembling a variety of files from various disk partitions on the 1581 disk drive.

Other Immediate Commands- If you have a monochrome monitor you would do well to turn off *Color Sensitivity* by entering the command *Color 1*. If you wish to re-enable color discrimination simply enter the command *Color 0* (or just drop the zero argument altogether by entering *Color!*). If you simply hate the use of colors, drop me a line and I'll provide you with info on how to customize your copy of MAS-128 so that it will always default to a *colorless* state on start up. There is no way yet to turn off case sensitivity. (Symbolic labels are unique not only by spelling, but by color and case.)

Here are some of the extra immediate commands for facilitating file conversion:

CON PAL "0:file name"	Loads in PAL source files or any files that look like your average run of the mill CBM BASIC program.
CON MAE "0:file name"	Loads in MAE source files. MAE files are peculiar in that all characters are only 7 bits long. If bit seven is ever set, that flags the end of the line. The actual line numbers are stored in Binary Coded Decimal format.
CON CBM "0:file name"	Loads in CBM ASM source files. These source files are presumed to be standard PETASCII files delimited by carriage returns and having no line numbers.
CON ASC "0:file name"	Loads in generic PETASCII source files. These files are presumed to be similar to CBM source files in that all text is stored as PETASCII. Unlike CBM ASM source, however, it is assumed that the generating assembler was line-oriented and therefore no line numbers will need to be generated.
CON SCR "0:file name"	Loads in generic Screen Code source files. Some assemblers generate source identical to ASC files but store data simply as regular screen codes with source lines terminated by null bytes (\$00's).
CON USA "0:file name"	Loads in generic source files that consist simply of the industry standard USA ASCII. Source lines are delimited by carriage returns.

If you try to convert a file that contains bytes that garbage out the MAS-128 screen editor, press CTRL RESTORE to clean things up. If you're not quite sure of what file

structure your assembler was sticking you with, try the view command `@view "0:file name"`; that's a lot easier than digging out the old disk utility! If it turns out that *none* of these 6 built-in file conversion utilities can do the trick, drop me a line.

Note that in file conversions, the source descriptor "MAE" or "PAL" need only be specified for the *first* conversion. Subsequent file conversions will default to the file type last specified.

You should realize that translating source files to MAS structure will entail at least *some*, and sometimes *lots* of recoding and rewriting in order to make the best of the MAS-128 assembly language environment. For example, you should optimize your source by changing all `lda argument1 sta argument2` sequences to comparable `mov argument1 argument2` sequences, which assembles faster and takes up less room. You should just as well change all `lda argument1 cmp argument2` sequences to comparable `chk argument1 argument2` sequences.

You should also be alert to the special expansions of the `mov` and `chk` operations when used in the immediate mode; if the addressed argument is taken to be *absolute* in the immediate mode, a pseudo-16 bit operation will be emulated! Wow! Who could ask for more? In the examples given below, the label "*skip*" may be referred to. That label is given here only to illustrate the destination of the branch; the branch is internal to the format and should not be written out separately.

EXAMPLES OF 16 BIT EMULATION

FORMAT: `chk #argument1 argument2` CASE: `#argument1 >= 256`

```

lda #argument1/256
cmp argument2+1
bne skip ; checks high bytes first
lda #argument1
cmp argument2 ; checks low bytes
skip

```

FORMAT: `mov #argument1 argument2` CASE: `#argument1 >= 256`

```

lda #argument1/256
sta argument2+1 ; transfer high bytes first
lda #argument
sta argument2 ; transfer low bytes second

```

FORMAT: `up@ argument1` CASE: any legal addressing mode

```

inc argument1
bne skip
inc argument1+1
skip

```

FORMAT: `dn@ argument1` CASE: any legal addressing mode

```

lda argument1
bne skip
dec argument1+1
skip dec argument1

```

HEX	"0:object file"	Load in file and convert it to a series of <i>hex</i> statements. You may wish to use this to analyse unusual object files, character sets, sprites, etc.
DIS \$2000 CBM	"0:object file"	Disassembles CBM Structure Object File
DIS \$2000 MAS	"0:object file"	Disassembles MAS Structure Object File

The above two examples disassemble an object file *as if it loaded at address \$2000*. The "load" address and the format descriptor need not be specified in all cases; they are optional.

Note that MAS object files are unique unto themselves. They are not to be executed in a CBM environment but rather in an environment where MAS-128 is resident, present, and in control. The copy of MAS-128 that is on your disk is not a MAS object file. Some MAS object files are not executable but are rather data files that may be used in assembly. For example, the user may wish to store sprites or character sets on disk in MAS object format.

LOD \$2000 CBM	"0:object file"	Loads file to RAM address \$2000
LOD \$2000 MAS	"0:object file"	Loads file to RAM address \$2000
SAV \$2000 \$2FFF	"0:file name"	Saves object from RAM to disk drive
SAV \$D000 \$DFFF	MAS "0:chrset data"	Saves binary data as MAS file to disk drive
GET 5 7	\$C000	Load Track 5, Sector 7 to address \$C000
PUT 1 0	\$C000	Write data \$C000-C0FF to Track 1, Sector 0
SEE 18 1		See hard ASCII image of Track 18, Sector 1

EXERCISE GREAT CAUTION- The *Get*, *Put*, and *See* commands all work the disk zero page job queue to seek tracks and read/write sectors. If you specify tracks that are physically outside your drive's stepping abilities, you are running the very likely prospect of jamming the r/w head. MAS-128 does *not* provide for checking your Track/Sector specifications. Mountain Wizardry Software will not be held liable for repairs or damages to your disk drive incidental to or consequent from the issuing of improper *get/put/see* commands. You should refrain from trying to access tracks beyond 35 (for the 1541 Disk Drive), tracks over 70 (for the 1571 Disk Drive), and tracks over 80 (for the 1581 Disk Drive). You are also seriously advised to review the specifications included in your drive's manual!

A couple of commands have been added to the MAS-128 "disk wedge":

@display	"0:pure PETASCII"	Print ASC file to screen
@view	"0:weird file"	View hard ASCII image of file
@#1		Redefine Device's own Identity Number to 1 (i.e., its primary address)

A major shortcoming of MAS-64 was its failure to include any memory-resident means of searching and replacing. This has been corrected in MAS-128 with the EDIT command. After you enter the argument to the EDIT command, press the RETURN key and you will be prompted for the replacement argument.

Of great importance is the enhanced keyboard scan. The numeric keypad to the right of the keyboard can be utilized as 16 redefinable function keys by simultaneously pressing a SHIFT key and a KEYPAD key. When you first power up, you should try SHIFT <keypad 4> and see what it does. Also, SHIFT <keypad 8> is another one to try. The "Classic Function Keys" F1-F8 also do a few things too- Try them out, preferably with a screen full of text.

Any of these special keypad function keys may be redefined with the KEY command. For example, KEY 1 "M. Montchalin" defines function key 1 (on the keypad) with the name of the author of MAS-128. A special case, however, is function key 15. This function key is expressed not through any combination of the shift keys and the numeric keypad keys. It is expressed by pressing the ESC key simultaneously with the asterisk "*" key. This key can also be defined in a unique way- By pressing the ESC key simultaneously with the equals "=" sign. It is best used for quickly copying phrases.

BREAK-INTERPRETED OPCODES

On the system disk you will find the source code for Breaker-128. Using this small program you can enhance the C-128's 8502 microprocessor in a big way- A whole new class of microprocessor instructions can be implemented via software! Many programmers will appreciate this added variety. As a matter of fact, it is quite possible to implement a series of extra BRK instructions differently according to the Bank of RAM that each BRK interpreter is installed in.

Note that the following binary representations are in disagreement with those listed in the MAS-64 Doc's: The MAS-64 doc's contained typesetting errors, as was detailed in the MAS-64 errata file. Moreover, the AIX and SIX instructions have been incorporated into the ADX and SBX instructions. When using the LOW addressing mode for ADX and SBX, the argument's highest nybble *must* be clear or a ?BAD ADDRESSING MODE error will be reported. For example, *adx \$0E32* is legal but *adx \$1E32* is not. The Low Addressing mode recognizes only those arguments from \$000-FFF as legal, i.e., any arguments 12 bits long or less.

Opcode	Binary Representation	Bytes	Flags	Name
BRK	----- 0000 0000 0000	2	-----	Old Break on Dummy Argument
STG	----- 0001 0000 0000	2	NZ	Get USP into AC, XR
STS	----- 1001 0000 0000	2	-----	Set USP from AC, XR
TXY	----- 0010 0000 0000	2	NZ	Load YR from XR
TYX	----- 1010 0000 0000	2	NZ	Load XR from YR
ADX	vvvv vvvv vvvv 0011 0000 0000	2	NVZC	Add (Low) to XR
SBX	vvvv vvvv vvvv 1011 0000 0000	2	NVZC	Sub (Low) from XR
ADX	vvvv vvvv vvvv 0100 0000 0000	2	NVZC	Add Low to XR
SBX	vvvv vvvv vvvv 1100 0000 0000	2	NVZC	Sub Low from XR
PHX	----- 0101 0000 0000	2	-----	Push XR onto User Stack
PLX	----- 1101 0000 0000	2	NZ	Pull XR off User Stack
BSR	vvvv vvvv vvvv v110 0000 0000	3	-----	Branch to Subroutine (REL,X)
BRA	vvvv vvvv vvvv v111 0000 0000	3	-----	Branch Always (REL,X)

<u>Digit</u>	<u>Meaning</u>
0	Clear Bit
1	Set Bit
v	Evaluated
--	Disregarded

The BRA and BSR instructions are *Relative Indexed Indirect* operations. The branch may be up to \$1000 bytes forward or backwards relative to the PC. This point is presumed to be the beginning of a table of vectors, with each vector stored in a normal low-byte high-byte sequence. The vector table may be up to 512 bytes long, with XR specifying which of 256 vectors will be loaded into the PC --Now, if XR = 00, the *first* vector in the table is used. The BSR pushes PC onto the normal 6502 stack just like the JSR instruction. In order to return from BSR you should use the RTS instruction, *not* RTI.

Example of Source with BRK-Opcodes

```

10Entry      sei                ; Stop Soft IRQ's
20           cld                ; Kill Decimal Mode
30           lda #$00
40           ldx #$A0
50           sts                ; Initialize User Stack Pointer at $A000
60           tax                ; Select first subroutine
60do'em_all  bsr (vector,x)    ; Do subroutine (USP not affected by BSR)
70           inx
80           bne do'em_all
90
100          ldx status
110          bra (Terminate,x)  ; Table must be within $1000 bytes
120
130vector    lh@ subroutine0    ; Table of 256 vectors
140          lh@ subroutine1
150          lh@ subroutine2
160          lh@ subroutine3
...

```

Of course, you'll have to install Breaker-128 somewhere in RAM for these BRK-opcodes to work properly.

You can rest assured that employing these BRK-implemented opcodes will virtually *guarantee* security from scrutiny -the *FIRST* step toward software protection- especially if you use them with liberal abandon! On the other hand, there *is* a trade off between space and time; executing a single BRK-instruction can save a handful of bytes of RAM but conversely *will* take up some 30 to 100 clock cycles. The bigger the program, the greater the need to use your RAM sparingly. You shouldn't use these instructions in time-critical loops. . . But then again you can simplify program flow and clean up horribly tangled sections of code by using them with deliberation and measured calculation.

You don't have to use device 8 as your main drive; on power up MAS-128 will automatically search for any devices present on the serial bus and accept the first one that it finds. (You'd *think* that the designers of the C-128 would have done this!) As for myself, I use a 1571 with its dipswitches set to signify device 9 on power up. MAS-128 lets you address serial devices 1 through 30.

MAS-128 has a great many features which I have not yet described. This Beta Version of MAS-128 also has the proprietary 1700/1750 RAMDOS that has occupied my time for the past few months. The RAMDOS is not yet complete, but you can access it as device 0 through MAS-128's TALK and LISTEN routines.

Always be sure to issue the `@0,"n0"` command before accessing the RAMDOS for the first time you power up! This resets the RAMDOS and gets rid of garbage files and spurious BAMS that interfere with proper downloading and file maintenance. This "new" command is *not* part of the regular MAS-128 initialization routines, so you'll have to do this manually.

The 1750 Expansion RAM can contain more than 2000 blocks of information (the rough equivalent of three 1541 disks).

These commands are implemented via the RAMDOS command channel:

<u>Command</u>	<u>Example</u>
Initialize	@i0"
New (reset)	@n0"
Validate	@v0"
Scratch	@s0:file name"
Rename	@r0:file name1 = 0:file name2"
Copy	@c0:file name1 = 0:file name2"
Exchange	@x0:file name1 = 0:file name2"
Lock	@l0:file name"
Download	@d:8"
Upload	@u:8"

DOWNLOADING *and* UPLOADING

The wholesale transferring of data *en masse* into the 1750 REU from an external disk drive is said to be *downloading*. Conversely, the transferral of the 1750 REU's contents *back* to an external disk drive is said to be *uploading*. The MAS-128's built in RAMDOS commands `@0,"d:9"` and `@0,"u:9"` facilitate the programmer's use of the RAM Expansion for temporary storage of standard disk files.

Special Note Regarding Downloading with the CBM 1581 3-1/2" drive-

The 1700/1750 RAMDOS performs downloads by working the drive's job queue in zero page. (This permits reading through most drive errors). In most cases, the drive directory begins at Track 18, Sector 0. With the 1581 Drive, however, the Directory begins at Track 40, Sector 0. Other Drives may store the directory at still *other* tracks. The first few sectors of the Directory Track usually store the BAM (Block Allocation Map) and should be skipped as they are immaterial to the download procedure. Additional information regarding Directory Storage is therefore required for downloading. You should issue the command `@d:9.80.40.3"`, say, to download from a Device 9 which has up to 80 tracks, with a directory starting at Track 40, Sector 3. All numbers in the RAMDOS command string are delimited by periods. It is presumed that the directory sectors abide by standard CBM directory structures with the first two bytes serving as a link to the next sector. You should exercise *extreme* caution with this command since specifying a Track Range (80 in the example above) that exceeds the physical abilities of the drive in question [for example, the 1541 Drive should really not be stepped beyond Track 36] may well jam the r/w head physically and require drive disassembly and realignment! Nevertheless, this command may prove of value with some disk cataloguing and file maintenance systems.

If MAS-128 seems to hang during a download and the *Drive Active* light fails to flash in the course of sixty seconds, you have probably encountered a bad diskette. Press [CTRL RESTORE] to regain control of the system. Read the status channels for both the device in question and for the RAMDOS. Read the RAM Directory. It may be desirable to issue the @*"i0"* command to both the device in question and to the RAMDISK. You may resume your download at this point; the bad file will be open. Do *not* validate your RAMDISK until you have finished downloading all your desired files. Since you have apparently encountered a corrupted file or a corrupted diskette, it is desirable to format a new diskette and then upload everything back immediately. To recover files that are laden with such disk errors it may be desirable to use a disk utility to patch things up. Note: To this end the in-house disk utility, S-128, may be packaged with the final version of MAS-128 as freeware; S-128 is modeled after the *C-64 Disk Zapper* which I wrote and copyrighted four years ago.

Documented Glitch! When you receive the "file not found" or "channel open" DOS status message, you should read the status channel a second or third time to remove redundant (usually identical) messages. After the "00,ok,<Ramtrack>,<Ramsector>" status is returned, you might want to initialize your RAMDOS with the command @*"i0"* to straighten things out. This generally harmless glitch only occurs when you try to access non-existent files.

Note that the exchange command merely exchanges the directory entries. It may be used for sorting. Eight different file types are supported (DEL, SEQ, PRG, USR, REL, TXT, OBJ, MAS). All eight file types are structured like regular Commodore binary PRG files; the Commodore relative file structure is not implemented. Should a 1 megabyte expansion RAM cartridge someday be released, I will probably utilize the numeric digit following the first letter of the DOS command to signify a "second" RAM drive, thus allowing for drives 0 and 1 to be emulated in RAM.

I've also noticed that the 1750 REU can sometimes, if not properly seated (plugged into the 128), jam the whole system, so that not even the RESET button can fix things up! If you fail to seat it completely, you should take care not to jog your C-128 around while programming. For that matter, it's not a good idea to spill your pop on your C-128 either!

Matthew Montchalin
6502 Systems Department
Mountain Wizardry Software
9870 S.E. City View Drive
Portland, OR 97266
(503) 265-2755