Exidy

SOFTWARE

INTERNALS MANUAL

by

Vic Tolomei

This is a detailed technical document written by a professional programmer about the internal software architecture of the Exidy Sorcerer. Included is a full explanation of the Monitor, BASIC areas, hardware ports, and assembly language interfacing from BASIC. All this is in easy to understand language. It is intended to be a supplement to the excellent documentation distributed by Exidy as a further aid to the Sorcerer programmer.

TABLE OF CONTENTS

PREFACE

This document is designed to aid the Exidy programmer in easily utilizing the myriad of wonderful facilities of the machine. There are many Monitor subroutines, uses of cassette tapes, BASIC programming techniques, and uses of the Input/Output ports which require a detailed explanation to be used to the fullest extent.

To obtain all the benefits from this manual, please read the two books that come with the Exidy "A Guided Tour of Personal Computing" and "A Short Tour of Basic". This internals manual is a supplement to these.

The manual is divided into several sections. Each is intended to be an independent "mini-manual" describing fully the topic under discussion.

INTRODUCTION TO THE Z80

Before you can understand how the Exidy really works, a few fundamentals have to be covered about the architecture of the Z80 MPU (MicroProcessing Unit). First of all, let's discuss the concept of "hex".

HEX, BINARY, AND DECIMAL

"Hex" is short for hexadecimal. This is a number system based on 16, not 10 as we are used to (decimal). In decimal, we have 10 possible digits, 0, 1, 2, ..., 8, and 9. In hex, we have 16. Of course the first 10 are 0 through 9 as with decimal. But there are 6 more, A, B, C, D, E, and F. "A" means 10, "B" means 11, "C" 12, "D" 13, "E" 14, and "F" 15. So a number like 1CB3 makes sense in hex. In decimal numbers each digit represents a "power" of 10, namely "ones", "tens", "hundreds", and "thousands". For example, the decimal number 1895 means 1 thousands plus 8 hundreds plus 9 tens plus 5 ones, or

```
1895 = 1x1000 + 8x100 + 9x10 + 5
     = 1000   + 800   + 90   + 5
```

In hex however, each digit (0 through F) represents a power of 16, "ones", "sixteens", "two hundred fifty sixes", and "four thousand ninety sixes". For example, the hex number 1895 can be written as in the example above

```
1895 = 1x4096 + 8x256 + 9x16 + 5
     = 4096   + 2048  + 144  + 5
     = 6293 (decimal)
```

Another hex number 3CF1 can be seen as

```
3CF1 = 3x4096 + 12x256 + 15x16 + 1
     = 12288  + 3072   + 240   + 1
     = 15601 (decimal)
```

The reason why understanding the hex number system is so important is because the majority of computers today, big, mini, and micro, are based entirely on hex. This includes the Z80 MPU, which is the basis of the Exidy Sorcerer. Its machine language instructions are in hex; its arithmetic is done in hex; characters typed on the keyboard, displayed on the screen, placed on cassette tape and printed on a printer are all in hex.

If you understand hex, then "binary" (the number system based on 2) should present no problems. There are only 2 digits possible to make any binary number, 0 and 1. These binary digits are called "bits". A bit can be 0 or 1. Each of these digits represents a power of 2 (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32768). So a number in binary like 0011110011110001 is

```
0011110011110001 = 0x32768 + 0x16384 + 1x8192 + 1x4096 +
                   1x2048  + 1x1024  + 0x512  + 0x256  +
                   1x128   + 1x64    + 1x32   + 1x16   +
                   0x8     + 0x4     + 0x2    + 1
                 = 8192 + 4096 + 2048 + 1024 +
                   128  + 64   + 32   + 16   + 1
                 = 15601 (decimal)
```

But that means, according to the previous example, that since 15601 decimal is also 3CF1 hex, then

0011110011110001 (binary) = 3CF1 (hex).

This is no mere coincidence. Let's see why. If we look at a "4-bit binary number" (ie, a number in binary made up of only 4 digits of 0's and 1's), then the smallest it could be is 0000 (0 decimal), and the largest it could be is 1111 (15 decimal or F hex). Thus every digit in hex, 0-F, can be expressed exactly as a 4-bit binary number:

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

In other words, a hex digit is really just another way of writing 4 bits, or, every 4 bits of a binary number can be grouped as 1 hex digit. Let's see how that works with the numbers we just did. 0011110011110001 can be broken into groups of 4 bits (right to left) as follows:

0011 1100 1111 0001

If each 4-bit group is viewed individually, they calculate to

0011 = 3 decimal (3 hex)
1100 = 12 decimal (C hex)
1111 = 15 decimal (F hex)
0001 = 1 decimal (1 hex)

So it can be written

0011	1100	1111	0001	binary
3	C	F	1	hex

So hex and binary are actually the same thing, with different groupings. Another example, to write 0F8D hex in binary

0	F	8	D	hex
0000	1111	1000	1101	binary

which, when pieced back together, becomes

0000111110001101 = 0F8D.

BITS, BYTES, ADDRESSES, AND "K"

Enough about decimal, hex, and binary. We now know how numbers are written on the Z80. Let us take a look at how memory is organized.

The smallest unit of information that can be placed in the memory of just about any computer made, including the Z80, is a bit, the same bit we saw earlier. This only holds a 0 or a 1 however, and is too small for normal numerical use. So a larger unit was created, called a "byte". A byte is just 8 bits or 2 hex digits grouped together.

So a byte can contain a number from 00000000 binary (00 hex, 0 decimal) to 11111111 binary (FF hex, 255 decimal). Each unique byte in the Exidy's memory space is assigned a 4-hex digit (2 byte) number called an "address". This address identifies the particular byte and its contents. Addresses start at 0000 hex and end at FFFF hex (65535). Thus the Exidy (Z80) can have up to 65536 bytes of memory. Another way programmers like to put this is to use the term "K". A "K" is just another way of saying the number 1024 decimal (400 hex). So 65536 boils down to 64K (64x1024 = 65536).

RAM VERSUS ROM

Since we are on the subject of memory, there are two types. In one type the contents can never be changed. Information can only be "read" from it. This is called Read Only Memory or ROM (computerists love abbreviations or acronyms). ROM is usually used to contain programs or data which is to be present in the same state all the time. For example, the Exidy Monitor program is in ROM (starting at memory byte address E000) and Exidy BASIC is in ROM (the ROM-PAC starting at address C000). ROM can have its contents "burned in" permanently at the factory, or can be burned in once by the programmer (called PROM or Programmable ROM), or can be erased by strong ultraviolet light and burned in over and over again (called EPROM or Eraseable PROM).

However, for programmers to write and run programs, we need memory which we can change or modify the contents. This is called Random Access Memory or RAM. When the size of an Exidy's memory is given (eg, 8K, 16K, 32K), this number applies only to RAM, or user-modifiable memory. All Exidy's have the same ROM area potential. So a 16K Exidy has 16x1024 or 16384 bytes of RAM.

STATIC VERSUS DYNAMIC

The above two terms are usually only applied to RAM. Static RAM has the ability to hold its contents indefinitely as long as electrical power is applied. Dynamic Ram on the other hand quickly (in milliseconds usually) loses or leaks its contents, and the data must be re-written or refreshed to the RAM often enough to keep the data from disappearing altogether. Typically static RAM requires more power, is more expensive, but is faster. The Exidy and many other Z80 based systems use dynamic RAM because of power and cost considerations, and also because the Z80 MPU is well-suited to interface to dynamic RAM (eg, it can be made to do the RAM refreshing).

Z80 ARCHITECTURE

The Z80 microprocessor is an 8-bit based machine. In other words, its data flow and arithmetic is usually on a 1-byte basis. It can address up to 64K bytes of memory. On the Exidy, a maximum of 32K bytes of this can be placed onboard (in the keyboard unit), while another 16K can be located as ROM for the Monitor and various ROM cartridges.

In addition to having 64K of possible memory, the Z80 has 22 registers. These are special high speed memories which reside on the MPU chip, and are used for arithmetic and program logic functions. These are all 1 byte in size unless otherwise noted:

A - the accumulator. This is the central register
F - the flags register. Each bit represents a CPU status. Eg, the "Z" bit is on if the A register contains 0. The "S" bit is on if A is negative
B - general use register
C - general use register
D - general use register
E - general use register
H - general use register
L - general use register
SP - 2-byte register containing the current stack address
PC - 2-byte program counter containing the address of the next instruction to be executed.

IX - 2-byte index register. Usually will contain an address to be used with a constant offset or displacement.
IY - 2-byte index register with the same type of use as IX.
I - register used to allow processing of external interrupts to the Z80 from the S100 bus
R - refresh register which can be used to provide dynamic RAM refreshing operations.

Registers A, F, B, C, D, E, H, and L have an alternate register called A', F', B', C', D', E', H', and L'. Only one set can be used at a time, while the other set allows space to save important program information. The EXX and EX Z80 instructions are used to flip back and forth between them. Also some registers can be connected together to create 2-byte, 16-bit register pairs. These are AF, BC, DE, and HL.

For more detailed information on the Z80 MPU I refer the reader to the Zilog publication "Z80 CPU, Z80A CPU Technical Manual" product number 03-0029-01.

EXIDY DEVICES AND PORTS

The Sorcerer has the following I/O devices or ports. Listed also is the Monitor command(s) to activate each:

a.	the keyboard	SET I=K
b.	the video screen	SET O=V
c.	cassette tape #1	SET I=S, SET O=S
d.	cassette tape #2	SET I=S, SET O=S
e.	serial RS-232 interface	SET I=S, SET O=S
f.	parallel interface	SET I=P, SET O=P
g.	Centronics printer interface	SET O=L

Note that these are onboard ports. This list does not include any devices added to the Exidy via the S100 expansion facility.

The keyboard is implemented as part of the Z80 I/O port number FE hex (245), input bits 0-4, output bits 0-3. The video screen needs no port but uses the 1920-byte RAM area at address E080 as a 64 by 30 screen. There is a port FE bit (input 5) indirectly related to video processing which signals when vertical retrace is in progress on the TV screen. The two cassette interfaces are part of the serial interface and provide an audio translation of the digital data suitable for recording on tape quite reliably.

EXIDY SERIAL PORT

The serial port allows data transfer to occur between the Exidy and external devices (such as printers, modems, cassette tape, and the like). Data travels one bit at a time in a predefined conventional sequence called asynchronous transmission protocol.

The protocol defines how the data is to look, and the speeds at which it is to travel. For example, each 8-bit byte of data is actually sent as a 10- or 11-bit stream, sometimes even longer. The 8-bits must be preceeded by a bit called a start bit, and must be followed by 1 or usually 2 or more stop bits. These bits also must be sent and received at a particular speed, predetermined by the sender and receiver. The speed is given in bits per second, or commonly called "baud" (derived from Baudot, the name of one of the for-runners of terminal communications). Thus 300 baud means 300 bits per second. Since it takes about 10-11 bits to transmit a byte or character, this means about 30 characters per second. The Exidy serial interface "speaks" this common language, and operates at one of two speeds, either 1200 baud (120 cps) or 300 baud (30 cps).

The serial port is actually two devices, an RS-232C interface and the dual cassette interface. RS-232C is the name given to a widely accepted standard of signal voltage and logic levels and the pinouts of the 25-pin plug or connector used for cableing between the sender and receiver. The asynchronous protocols signals are usually sent via this RS-232C standard. Another part of Z80 port FE (output bit 7) determines whether the serial port is RS-232C (bit on) or dual cassette (bit off). Cassette is the default. Output bit 6 controls the baud rate (1=1200, default, 0=300). Port status is placed on port FD while data transfer occurs on FC. For example, to connect a 300 or 1200 baud RS-232C serial printer to the Exidy, follow instructions given with the printer and from Exidy. However, the following guidelines may be used:

1. Connect pin 7 of the serial DB25 connecter to printer ground pin 7.
2. Connect pin 3 to printer pin 2.
3. Connect pin 2 to printer pin 3.

Reset the Exidy, enter the Monitor (BYE in BASIC), enter the command SET 0=S, and all output which would've gone to the screen will go to the printer, until Reset or SET 0=x is entered (x is usually V to return to video). There is also software available from Exidy providing a serial driver, and the ability to use the serial interface to turn the Sorcerer into a dumb terminal connected to another computer. Typically a modem and possibly an acoustic coupler may be required here. Reverse pins 2 and 3 in the above guidelines for this use.

The cassette interfaces may also be used with motor control. Pins 12 and 24, 13 and 25 can be used to turn cassette number 1 and 2 off and on for SAVEs, LOADs, FILEs and BATCHs commands. Pins 15, 5 and 20, 16, 18, and 21 are the mike input, auxiliary input, and earphone output connections. Note that cassette number 1 has these mike and ear connections duplicated as RCA plugs on the back of the Sorcerer.

EXIDY PARALLEL PORT

The parallel port differs from the serial port mainly in that data is transferred an entire byte at a time. This is ideal for fast printers and sometimes even some floppy disk units. The Sorcerer also provides an interface to the popular Centronics printer. The same parallel port is used, but unique software "handshaking" is done by the Monitor I/O driver. An example of the handshaking which occurs between the Sorcerer and printer might be the following "electronic conversation" over port FE, the parallel interface status port:

Printer: "Wait, I'm still busy, send no data."
"OK, now you can send."
Exidy: "Here it is, let me know when I can send more".

The 8-bit data (and at times status) rides on port FF.

To successfully hook up a Centronics or Centronics-like printer to the parallel port, again follow the printer's and Exidy's instructions. Here are some additional guidelines:

1. Connect parallel pins (DB25 connectors again) 5-7 and 16-19 (data bits 0-6) to the printer's data lines 0-6 (see printer's pinouts).
2. Connect pin 4 (data output bit 7) to the printer's input strobe line, a negative (true is low, false is high) pulse indicating data is ready to be transmitted.
3. Connect pin 1 to the printer ground.
4. Connect pin 25 (input data bit 7) to the printer busy line, indicating the printer is not ready to accept any data (probably still printing previous data).
5. Pins 2 and 3 (output accepted and available) and others may also be required depending on the printer model.

Once this is done, Reset the Exidy, enter the Monitor, type in the command SET 0=L, and from that point on all output will be routed to the screen and the printer, until Reset occurs or until another SET 0=x command is entered.

EXIDY MONITOR MEMORY MAP

To get an overall picture of how the Exidy utilizes the 64K of (possible) memory, a "memory map" is given.

Memory is cut up into pieces and each piece is used for a different purpose. In the map below the address of the first byte of each piece is listed along with the use of that area. The address is given in both hex and a form of decimal that is usable directly in BASIC with the PEEK and POKE commands. Note that some of these decimal numbers are negative. If the address exceeds 32767 (hex 7FFF), then BASIC requires that the "twos-complement" form of the number be used, or the negative form. For numbers greater than 7FFF, 65536 is subtracted from the number.

Be aware also that this is an overall wide angle view of memory. Detailed maps of certain areas (such as the Monitor Workarea and the BASIC Control Area) will follow.

ADDRESS		DESCRIPTION
0000	0	256-byte Z80 Restart space (RAM)
0100	256	User RAM start, begin BASIC Control Area (RAM)
1F00	7936	8K Monitor Stack end (8K machines) (RAM)
3F00	16128	16K
7F00	32512	32K
1F90	8080	8K Monitor Stack start (8K machines) (RAM)
3F90	16272	16K
7F90	32656	32K
1F91	8081	8K Monitor Workarea start (8K machines) (RAM)
3F91	16273	16K
7F91	32657	32K
1FFF	8191	8K End User RAM (8K machines) (RAM)
3FFF	16383	16K
7FFF	32767	32K
C000	-16384	Begin 8K ROM PAC (eg, begin BASIC) (ROM)
E000	-8192	Begin 4K Monitor Program (ROM)
F000	-4096	128-byte video driver space (RAM)
F080	-3968	1920-byte video screen (64x30) (RAM)
F800	-2048	1K standard Exidy ASCII alphanumerics (00-7F) (PROM)
FC00	-1024	512-byte Exidy keyboard standard graphics character set, accessed by depressing GRAPHICS key, character codes hex 80-BF (128-191) (RAM)
FE00	-512	512-byte User Programmable graphics character set, accessed by depressing SHIFT and GRAPHICS keys, codes hex C0-FF (192-255) (RAM)
FFFF	-1	End Exidy address space (64K)

MONITOR WORKAREA

This is a detailed description of the area of memory shown above at locations 1F91, 3F91, or 7F91, depending on the size of the machine.

The Monitor Workarea, hereafter called MWA, is the area in RAM used by the Exidy Monitor program to save important information needed for its successful operation. This area is always located right next to the Monitor Stack, and is always placed at the very top of available RAM space. For an 8K

machine, the top of RAM is at 1FFF (8191), for 16K 3FFF (16383), and for 32K 7FFF (32767). This number, Himem, is placed by the Monitor in the two bytes at address F000-F001 (-4096 to -4095) in the video driver RAM space. Remember as with most micros, the two bytes are reversed in storage. For example, for a 16K Exidy, F000-F001 contains FF3F, not 3FFF. The address of the MWA can be obtained from this Himem address so that you don't have to worry about what size machine your programming is running on. To do this, you must get the Himem value at F000-F001 and subtract 6E (110) or add FF92 (-110). For example, in Z80 Assembly Language:

```
LD    HL,(F000)    ;GET HIMEM
LD    BC,FF92      ;GET -110
ADD   HL,BC        ;HL POINTS TO THE MWA
```

Or in BASIC:

```
100 AD=256*PEEK(-4095)+PEEK(-4096)
110 IF AD>32767 THEN AD=AD-65536
120 AD=AD-110
```

There is also a Monitor subroutine designed to do this calculation for you. It is at address E1A2 (-7774). When CALLed, it puts the MWA address in Z80 register IY. Eg:

```
CALL  E1A2         ;IY POINTS TO THE MWA
```

A detailed map of the contents of the MWA will now be given. This will be in the same fashion as the overall memory map listed above, except that the addresses will be shown in a different form. First the offset in hex from the beginning of the MWA will be given. This can be used in Z80 Assembly Language as a displacement away from an index register such as IY, which points to the MWA. For example, if the displacement is listed as +41 to a particular field, then that field can be addressed in Z80 by (IY+41) or by 41(IY). The second part of the address is given as an absolute address of the field in RAM. Since the whole MWA moves dependent on the size of the machine, the first two hex digits of these addresses can change. The last two digits are always the same. So only these last two digits are listed. The first two will either be 1F (8K), 3F (16K), or 7F (32K). Note: if the user coldstarts the Sorcerer (Resets) with a size other than the above sizes (such as 21239 bytes, not even a whole multiple of a K) then the above addressing scheme is not applicable and only the displacement from the index register scheme may be used.

ADDRESS		DESCRIPTION
+00	91	60-byte Monitor command input buffer. Any command entered from the current RECEIVE device (SET I=x) such as the keyboard, serial or parallel ports is placed in this area. It is left-justified, and terminated by an ASCII carriage return character (hex code 0D, 13 decimal, hereafter called a CR). The Monitor subroutine at E13A (-7878) builds this buffer from the input.
+3C	CD	Port FE interface status
+3D	CE	Serial interface and dual cassette interface baud rate save area. 1200 baud is indicated by hex 40, 300 baud by the value 00. Serial port or cassette baud rates are set to the default of 1200 baud (hex 40) by the Monitor COLD Reset routine (at E000, -8192) and by the Monitor USER Reset entry point (at E003, -8189). Such a coldstart is done, for example, when the RESET keys are depressed. This byte is also set by the SET T=0 and SET T=1 commands (at Monitor routines at E5A2, -6750)
+3E	CF	SEND delay time. This value is used to delay before a SEND (to video, serial, or parallel) is done. The actual delay is about 1500 times this value machine cycles. This delay can therefore range from 0 to approximately 400000 cycles. The value is set by the SET S=n command.
+3F	D0	Current SEND routine address. The default address set by COLD starts is the video routine at E9F0 (-5648). It can be changed by the SET O=x command.
+41	D2	Current RECEIVE routine address. The default is set by COLD starts to be the keyboard routine at EB1C, -5348. It can be changed by the SET I=x command.
+43	D4	Batch mode status. 00=normal input, nonzero=batch mode. This byte is used by the Monitor command input routine (E142) to determine whether commands are to be gotten from the RECEIVE device or from the batch tape serial port. The OVER command turns this off and the BATCH command turns this on.
+44	D5	Monitor output prompt character. The default is the character ">" or ASCII code 3E (62) set by COLD starts. It can be changed by the PROMPT x command. It is output to the SEND device every time a Monitor input command is being requested (at E0ED, -7955).

ADDRESS		DESCRIPTION
+45	D6	tape status, baud rate, motor control save area. This is zeroed when the tape(s) is turned off, and otherwise remembers the status of the tape baud rates (00=300, 40=1200) and motor controls (10=motor #1 on, 20=motor #2 on).
+46	D7	tape input and output CRC (Cyclic Redundancy Check). The CRC is used to check whether the data has been transmitted successfully to/from the tape. This technique is described in detail in a subsequent section.
+47	D8	Beginning of the 16-byte tape output file header area. The first 5 bytes here contain the 5-character ASCII file name as entered on the SAVE or CSAVE command. It is left justified and padded to the right with ASCII blanks (code 20, 32 decimal).
+4C	DD	File header id, usually hex 55.
+4D	DE	File type. Usually C2 (194) for a BASIC save file. If the high order bit (80, 128 decimal) is on, the file cannot be automatically executed with the LOADG command. This is set by the SET F=xx command
+4E	DF	2-byte length of the file in bytes.
+50	E1	2-byte program loading address. For BASIC files, this is always 01D5 (469) because BASIC programs always start at that address. See the BASIC Control Area description following. For other programs such as those in machine language, this address is the "ssss" of the command"SAVE name ssss eeee."
+52	E3	2-byte program "go-address" for auto execution files. The Monitor will automatically begin execution of the program at this address with the LOADG command. This address is set by the SET X=nnnn command.
+54	E5	3 bytes of reserved space, ending the output tape header
+57	E8	16-byte tape input header area. The format is identical to that of the area at +47. This area is filled in from reading the tape for commands such as CLOAD, LOAD, FILES, and so on.
+67	F8	character under the cursor. Since the cursor is an underscore character (ASCII code 5F, 95 decimal), it actually replaces the character at the cursor location. This hidden character is saved to be put back when the cursor is moved. The save is done by E9CC (-5684), and it is replaced by E9E8 (-5656).
+68	F9	2-byte line number where the cursor is times 64. This ranges from 0x64 (0) to 29x64 (1856), and is the offset from the beginning of the screen to the cursor line start.
+6A	FB	2-byte cursor column number (0-63). When added to +68 the actual cursor offset into the screen is found.

ADDRESS		DESCRIPTION
+6C	FD	Last character entered from the keyboard. This is is used for the processing of the REPT (repeat) key logic. This character is entered to the keyboard input routine about every 30000 machine cycles as long as the REPT key is depressed. It is always the last key entered, and is saved and used by the keyboard processing routine at EB1C (-5348).
+6D	FE	two bytes of reserved space. This brings us to the end of the MWA, and in fact the end of user RAM.

CASSETTE TAPE FILE FORMAT

When a SAVE, LOAD, or FILES command is done from the Monitor, or when a CSAVE or CLOAD is done from BASIC, files are processed from the cassette tape device on the serial interface. This applies to both cassette #1 and #2. Cassette tape motor-on routine can be found at E024 (-8156), motor-off at E027 (-8153), cassette save at E02A (-8151), and cassette load at E02D (-8148).

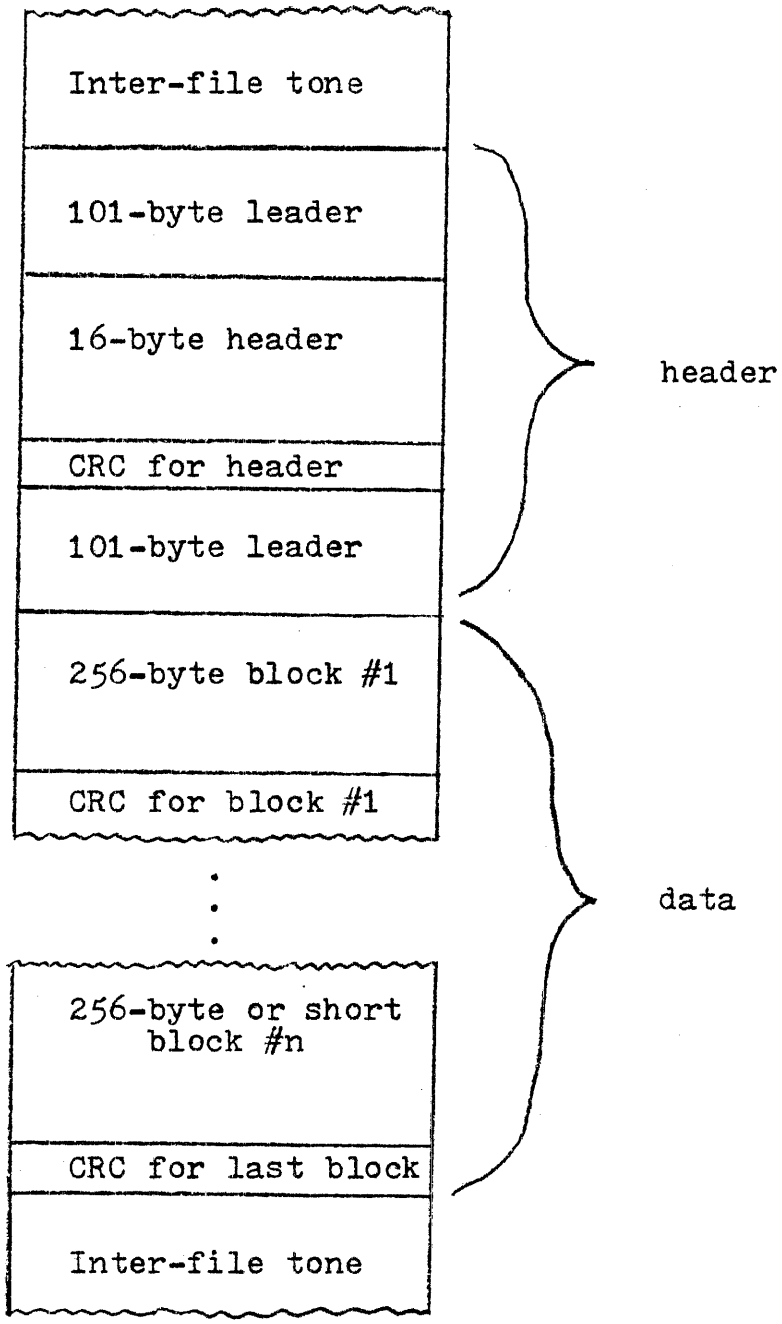
Cassette files on the Exidy have the following appearance, whether at 300 or 1200 baud:

1. Inter-file tone
    a. a high frequency tone always output by the cassette interface when data is not present.
2. 101-byte leader
    a. 100 bytes of 00 (nulls)
    b. 1 byte of 01 (control-A or SOH, Start-Of-Header)
3. 16-byte file header (see description in MWA above)
4. CRC for header
    a. 1 byte CRC for error checking. Details later.
5. Up to 256 bytes of data
6. CRC for above data block (1 byte again)

7. Repeat 5 and 6 until data exhausted. The last data block may be short (less than 256 bytes). CRC still follows.

8. Inter-file tone (same as before the file).

This format is used by both BASIC and machine language files. It is depicted pictorially as follows:



To LOAD or CLOAD a file, or to perform a FILES command, the Monitor scans the tape (whichever is on) for the leader. Then the header is read into the MWA and the "FOUND ..." message is sent to the current SEND device. The data portion is then either skipped (wrong file, or FILES command) or loaded. All CRC's are always validity checked for any of these commands. Thus to check all the bits on an entire tape for errors, it is sufficient to perform a FILES command.

Note that the default tape transfer rate is 1200 baud. A much more reliable method of saving data is to use 300 baud. However it will take 4 times longer to SAVE and LOAD, and use a lot more tape. This is accomplished with the SET T=1 command.

Still, even at 1200 baud, the Sorcerer tape system is the best I've come across. It is the most reliable, and with its file headers, it is the easiest to use. The user does not even need a recorder with a tape digital counter to find files with these headers. The cleverness of the tape system makes the Exidy basic offering (just cassette, no expansion to S100 capability, diskette, etc.) a very attractive low-priced system.

TIPS ON LOADING AND SAVING FILES ON TAPE

The following hints can be used to minimize problems with cassette recording of files:

To Load:

1. Use a relatively inexpensive cassette recorder ($30-$60) with ALC (Automatic Level Control). This means you have no control over the volume or tone of the recordings. All are made exactly the same way. Strangely enough, experience shows that expensive recorders work worse.
2. Connect the MIC wire to the microphone input. Do not use the auxiliary input on most recorders. The signal will be too weak.
3. Connect the EAR wire to the earphone or monitor jack.

To Play:

1. You must find the correct volume and tone for your recorder. As a first guess, set volume and tone to 7-8 out of 10, or 3/4 high.

2. Listen to the tape play through the speaker. The intra-file tone should be louder than normal listening volume, maybe even as loud as possible without distortion and noise. The data should sound high-pitched and clear, like static.
3. Try loading a file. Tinker with volume and tone until at least a file header is read without a CRC error ("FOUND ..." message appears). Now you are close enough to the correct settings.
4. Once found, the correct settings should be able to be used for all tapes recorded on that recorder.

CASSETTE TAPE ERROR CHECKING

The CRC (Cyclic Redundancy Check) method is used to detect bit transmission errors in cassette data recordings. The CRC is stored at MWA+46. CRC checking is done with this algorithm: When the file is first written to tape (ie, when the 101-byte leader is written), the CRC is 0'd. For every data byte, in program or header, the current CRC is subtracted from the data (data-CRC), and the ones complement of this is used as the next CRC for the next byte (ie, FF-(data-CRC), or all the bits are flipped - 0's become 1's, and 1's 0's). When the file or block is completely written, the current CRC is written as the final byte. Note: this is why BASIC programs grow by one byte every time they are loaded and re-saved. When the file is loaded again, the CRC is calculated again as above, and is compared to the last byte of the block (the CRC written). A match means no errors (almost always), while a mismatch means an error. This is identical in BASIC files as in machine language files, since the same Monitor routines are used to write/read tapes.

PROGRAMMABLE GRAPHICS CHARACTER SET

Each byte in memory can contain exactly one character which can be input from the keyboard, displayed on the video, printed, etc. Thus there are 256 possible combinations of these characters (00-FF, 0-255). These codes can be mapped as follows on the Exidy. Again, codes are given in both hex and decimal.

CODE		DESCRIPTION
00-7F	0-127	128 standard ASCII characters:
00-1F	0-31	32 ASCII control characters (eg, CR, LF, etc).
20	32	ASCII blank
21-2F	33-47	ASCII punctuation
30-39	48-57	ASCII numbers 0-9
3A-40	58-64	ASCII punctuation
41-5A	65-90	ASCII upper case A-Z
5B-60	91-96	ASCII punctuation
61-7A	97-122	ASCII lower case a-z
7B-7F	123-127	ASCII punctuation and "delete" character (7F)
80-BF	128-191	64 standard Exidy keyboard graphics. These are obtained by depressing the GRAPHICS key
C0-FF	192-255	64 programmable graphics characters. These are obtained by depressing SHIFT and GRAPHICS keys:
C0	192	GRAPHIC SHIFT 1
C1	193	2
C2	194	3
C3	195	4
C4	196	5
C5	197	6
C6	198	7
C7	199	8
C8	200	9
C9	201	0
CA	202	:
CB	203	- (hyphen)
CC	204	^
CD	205	(tab)
CE	206	Q
CF	207	W
D0	208	E
D1	209	R
D2	210	T
D3	211	Y
D4	212	U
D5	213	I
D6	214	O
D7	215	P
D8	216	[
D9	217	]
DA	218	A
DB	219	S
DC	220	D
DD	221	F
DE	222	G
DF	223	H
E0	224	J
E1	225	K
E2	226	L

E3	227	GRAPHIC SHIFT	;
E4	228		@
E5	229		¦
E6	230		_ (underscore)
E7	231		Z
E8	232		X
E9	233		C
EA	234		V
EB	235		B
EC	236		N
ED	237		M
EE	238		, (comma)
EF	239		. (period)
F0	240		/ (slash)
F1	241		- (on numeric pad)
F2	242		7 (on numeric pad)
F3	243		8 (on numeric pad)
F4	244		9 (on numeric pad)
F5	245		+ (on numeric pad)
F6	246		4 (on numeric pad)
F7	247		6 (on numeric pad)
F8	248		x (on numeric pad)
F9	249		1 (on numeric pad)
FA	250		2 (on numeric pad)
FB	251		3 (on numeric pad)
FC	252		+ (on numeric pad)
FD	253		0 (on numeric pad)
FE	254		. (on numeric pad)
FF	255		= (on numeric pad)

Each of the above 64 characters can be defined to be any design or shape desired. Each consists of 8 bytes in memory, or 64 bits. These sets of 8 bytes (64 of them) start at address FE00 (-512). On the screen each character consists of 8 lines of 8 dots, or 64 dots, Thus each of the 8 bytes defining the character in memory corresponds to one of the 8 lines of the character in the display, and each of the 8 bits in that byte is a dot in that line. If the bit is on (1), then the dot is white. If the bit is off (0), then the dot is black. For example, a circle with a dot in the middle could be defined as a character. It would require defining each of the 64 (8x8) dots as 64 (8x8) bits in memory. So

.....	00000000 binary	00 hex	0 decimal
..xxx...	00111000	38	56
.x...x..	01000100	44	68
x.....x.	10000010	82	130
x..x..x.	10010010	92	146
x.....x.	10000010	82	130
.x...x..	01000100	44	68
..xxx...	00111000	38	56

The first 128 characters (00-7F, ASCII) are not under user control. The information required to display these characters is located in PROM at F800-FBFF (1K). The next 64 characters (80-BF, Exidy Graphics) can be programmed if desired, but they are already programmed to be standard keyboard graphics. The 64x8 (512) bytes for these are located at FC00-FDFF. This RAM can be changed at any time by the programmer to redefine these characters. However, the Monitor refreshes this area from its ROM every time a RESET occurs, or whenever the video screen is cleared (eg, when CLEAR is pressed, or when a Form Feed ASCII control is displayed). This will clobber any such modifications.

The last 64 characters (C0-FF) are completely under programmer control. They are always displayed as nonsense until they are "defined" by turning on and off the bits of the 8 bytes associated with the character. These bytes are in RAM from FE00 to FFFF (-512 to -1). For example, the character C0 (192) is at FE00-FE07 (-512 to -505), C1 (193) at FE08-FE0F (-504 to -497), C2 at FE10-FE17, and so on, till FF (255) is at FFF8-FFFF (-8 to -1). The formula to calculate where the 8 bytes in RAM begin for any of these 128 characters which can be programmed (80-FF) is (assume "c" is the character code of the character to be programmed):

```
FC00 + (8 * (c - 80))          hex, or

(8 * (c - 128)) - 1024         BASIC decimal
```

where "c" ranges from 80-FF (128-255).

For example, to print a "blot" (all dots on, a white square) on the screen followed by the above circle with the dot in the middle, the following BASIC program can be written. The blot will be made from the first programmable graphic 192, and the circle/dot will be 193:

```
10 FOR I=0 TO 7: REM 8 BYTES AT FE00 (-512) FOR BLOT
20 POKE -512+I,255: NEXT: REM TURN ON ALL BITS/DOTS
30 FOR I=0 TO 7: REM 8 BYTES AT FE08 (-504) FOR CHR #193
40 READ J: REM GET A BYTE VALUE FROM THE TABLE AS ABOVE
50 POKE -504+I,J: NEXT: REM TURN ON CORRECT DOTS
60 PRINT CHR$(192);CHR$(193): REM PRINT THE 2 NEW CHRS
70 DATA 0,56,68,130,146,130,68,56: REM DATA CHR #193
80 END
```

BASIC FLOATING POINT FORMAT

Numbers in BASIC are not integers. Fractions are allowed. Thus the decimal point can move. For example, the decimal point "floats" when 13.25 is divided by 10 - 1.325. It is from this idea that the term "floating point" was derived.

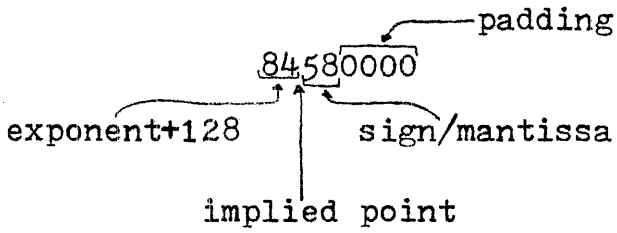
These numbers are stored by BASIC in 4 bytes of memory. Each number has 3 parts

1. the sign (+ or -)
2. the "mantissa" (the actual number, but with the point shifted to the left of the leftmost 1 bit of the number). So the number 127 decimal (7F, 01111111) is a mantissa if it is thought of as .1111111
3. the "exponent", which is how much the point had to be shifted in the number to produce the mantissa with the point at the left

This all sounds very complex, but it actually is not. Let's take an example, say 13.5 decimal. In hex this would be equal to D.8 (13 + 8*1/16). Remembering that hex is just groups of 4 bits, the binary equivalent of 13.5 would be 1101.1000 To create a mantissa from this, we must shift the point (in this case, the "binary point", not the decimal point) to the left 4 places, producing .11011000
The exponent can now be calculated. It is always positive if the mantissa shift was to the left, negative if to the right, and zero if no shift was necessary. Thus the exponent in this example would be +4 (4 to the left). However, we are not quite done. Rather than worrying about how to express a negative number exponent, 128 decimal (hex 80) is always added to the exponent to produce the final result. Thus the final exponent is 84 (132). Now we come to the sign. Since the digit to the far left of the mantissa is always a 1 (because we shifted until that was the case), then the sign can be stored in this bit without losing any information. If the number is positive or zero, then the sign bit will be 0. If negative, then the sign bit will be a 1. So the mantissa for 13.8 .11011000 changes to .01011000. To assemble this number, first we put the exponent 84 then the mantissa filled out to the right to fill out the 4 bytes:

10000100 .01011000 00000000 00000000

Now if we ignore the point, since it is always in the same place, and convert to hex, we have:



If the original number were -13.5 instead, then nothing would change except the sign. That is the mantissa would change from .01011000 to .11011000, so the new number would be

84D80000

In the reverse direction, to convert floating point back to decimal, let's use 88FF4000 as an example:

1. examine the exponent (88) and subtract hex 80 (128). In this example 88-80=08. But this may produce a negative number.
2. Examine the mantissa with the implied point (.FF4000).
3. If the left bit (high order, the one next to the point) is on (it is), then the number is negative, otherwise it is positive.
4. In either case, turn that bit on.
5. Shift the point according to the exponent from step 1 (08 here). If plus, shift right, if minus, left, if zero, no shift. Since we have +8, shift the point right 8 bits

   .11111111 01000000000000000

6. The number is now FF.4000, and with the sign, -FF.4000, or -255.25 decimal.

The only special case is the number 0. Here the exponent is 00. Other examples are:

1815	= hex	717	=	8B62E000
1		1	=	81000000
-1		-1	=	81800000
-.5		-.8	=	80800000
0		0	=	00610000

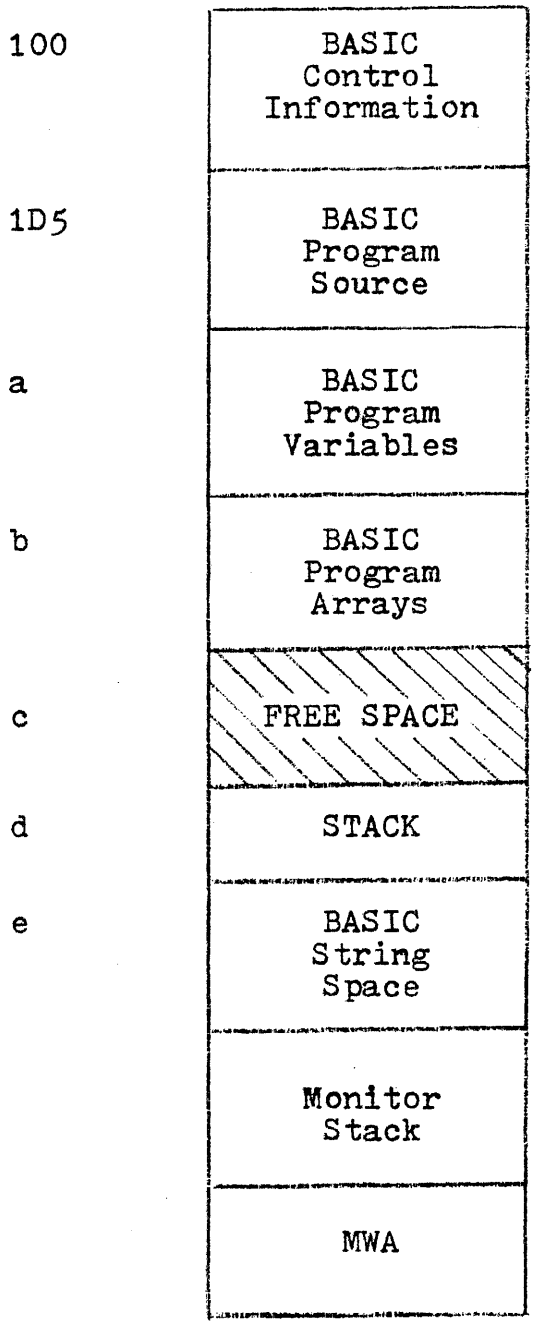
TO BE IGNORED

The last idea that must be mentioned is that the number is actually stored in memory in reverse, so the number eemmnnpp is stored ppnnmmee. For example, decimal 1815 in the above example:

00E0628B

BASIC CONTROL AREA

This is a discussion of the workarea in RAM used by BASIC, called the BASIC Control Area, or BCA. The BCA begins at adress 100 (256), and has an overall appearance like



In detail, RAM locations 100-14E (256-334) are copied from the BASIC ROM (address C258) when a BASIC Cold Start occurs (ie, after Reset or a PP X command is entered). The BCA described below includes only those areas which are of direct use to the programmer. It is intentionally sketchy, especially due to the great number of fields.

ADDRESS	DESCRIPTION
100/256	3-byte JUMP instruction to C06B (Warm Start). Done when PP command is entered without operands.
103/259	3-byte JUMP to C7E5 default (displays "FC ERROR" message). This is the USR function hook. See BASIC Assembly interface section later for details.
145/325	2-byte address of top of string space (letter "e" above) or the beginning of the BASIC stack. This is set by the BASIC CLEAR n command.
147/327	BASIC line input buffer and Direct Mode execution line.
18E/398	current line column number
1B1/433	2-byte address of instruction in the BASIC program about to be executed when Control-C break is entered. This could be in the middle of a line of multiple statements separated by colons.
1B3/435	2-byte BASIC line number of current line
1B5/437	2-byte address of the next full line to execute from the link pointer of the current line (see below).
1B7/439	2-byte address of the end of the program and the beginning of the BASIC Program Variable Area (letter "a" above).
1B9/441	2-byte address of the end of the Variable Area and the start of the BASIC Program Array Area (letter "b" above). Whenever changes are made to the BASIC program (adding, deleting, updating lines) the above two addresses are used to define a new Variable and Array area below the new BASIC program. Thus a program cannot be continued with old variable/array values once a change has been made.
1BB/443	2-byte address of the end of the Array Area and the pointer to free space (room for expansion - letter "c").
1BD/445	2-byte address of the last used data operand of a DATA statement so that the next READ will find the appropriate item. This is reset by a RESTORE command.
1BF/447	4-byte input parameter (usually floating point format) to the USR function, and output parameter from the USR function. If USR(3.5) is called, 3.5 is passed to the subroutine in floating point. See a later section for BASIC/Assembly interfacing details.
1D5/469	Beginning of all BASIC programs

FORMAT OF BASIC PROGRAM STATEMENTS

The first line of every BASIC program begins at location 1D5. All BASIC lines have the following variable length format:

OFFSET	DESCRIPTION
+0	2-byte link pointer address of the next sequential full line in the program. This is independent of multiple statements on one line (separated by colons). The last line of the program points to location 0000 to indicate the end.
+2	2-byte BASIC line number of the line in integer binary (a number between 0000 and FFF9, 0-65529).
+4	The BASIC statement(s), variable in length. Let us say they are "n" bytes long. Each BASIC "reserved word" such as GOTO, IF, END, DIM, PRINT, etc is encoded here to a 1-byte character not belonging to the ASCII character set (ie, hex codes greater than 7F). This speeds up processing and saves program memory space. When the program is LISTed, these special bytes are decoded back into their corresponding reserved words.
+4+n	Byte of 00 indication the end of this line and beginning of the next.

FORMAT OF BASIC FLOATING POINT VARIABLES AND ARRAYS

A BASIC floating point variable resides in the BASIC Program Varaible Area. Each one takes a constant 6 bytes:

OFFSET	DESCRIPTION
+0	2-byte ASCII variable name. * The high order bit is always 0. The letters are also reversed as usual.
+2	4-byte floating point value currently held by this variable. See the format description earlier.

BASIC arrays all reside together after the variables in the BASIC Program Array Area. A floating point array is variable in length. It takes a minimum of 7 bytes and looks like this: (Note: an array in Exidy BASIC can have any number of dimensions; call that number "n". Each can have any number of elements).

OFFSET	DESCRIPTION
+0	2-byte array name. The high order bit is always 0. The letters are reversed.
+2	2-byte total array length minus 4 (ie, the length of the array starting after these 2 bytes). This is used to find the next array in the area quickly.
+4	1-byte number of dimensions (we called it n).
+5	2-byte size (number of elements) in the 1st dimension.
+7	2-byte size of the 2nd dimension (if any).
.	.
.	.
.	.
+5+2(n-1)	2-byte size of the nth dimension
+5+2n	Beginning of a list of contiguous 4-byte floating point array elements. These are in Row order.

FORMAT OF BASIC STRING VARIABLES AND ARRAYS

A BASIC string variable is similar to a floating point variable. It is also 6 bytes long. It looks like:

OFFSET	DESCRIPTION
+0	2-byte variable name. The high order bit is always 1.
+2	1-byte current length of the variable length string value.
+3	00
+4	2-byte address of the string itself. It resides either in the string space or in the program statement itself (eg, 1005 A$="HI").

A string array is identical to a numeric array except for two very important features:

1. the high order bit of the array name is always 1
2. the 4-byte value is not floating point format but the length/00/stringaddress fields described above. All dimensioning remains the same.

BASIC TO Z80 ASSEMBLY LANGUAGE INTERFACE

To call Z80 Assembly Language subroutines from Exidy BASIC, certain general conventions and procedures must be followed:

1. The machine language program must reside either in the first 256 bytes of memory (00-FF, 0-255 - usually a bad idea) or in the BASIC free space area described earlier. Either BASIC control, program, variables, arrays or strings, or Monitor/video control resides in the rest of memory. This is the only way a BASIC and machine language hybrid can coexist without complicated machinations such as putting the machine language routine right after the BASIC program and fooling BASIC into thinking that it is part of the program. The BASIC free space is the best and easiest choice. However there are some potential problems:
   a. Free space is dynamic. As the program changes, as variables/arrays are added or change size, the start of the free space moves. A machine language program placed to close to the end of the Array Area can get walked on. The end of the free space changes too, since the BASIC stack (or string space) will grow and shrink, especially with the CLEAR command. Since this change is usually not as radical as that of the start of the free space, I recommend putting the program close to the <u>end</u> of the free space. But there are now other considerations.
   b. The free space ends near HIMEM of the machine (where the BASIC stack is). This changes with each different Exidy size. So a generalized subroutine designed to run on any machine (probably with several BASIC programs) would either have to be relocatable (able to be moved without affecting anything), or there will have to be different versions of the program to run on different size machines. This of course would allow the BASIC program to use the maximum amount of free space. A subroutine designed for a <u>particular</u> BASIC program could be placed at the top of the free space as long as the BASIC program does not grow too much.
   c. If the program is placed at the end of the free space an excessive CLEAR n BASIC statement could kill it.
   d. Thus no matter where the program is placed, certain restrictions have to be made to coexist with BASIC.

2. Assume a good location is found, and the Z80 program is written and relocated to that address in RAM. Assume this address to be 312A hex (12586). To call this subroutine from BASIC, it must already be in memory, and the USR function must be used. When BASIC executes it,

it converts the argument to floating point and places this number in the 4-byte USR parameter area at 1BF-1C2 (447-450). It then calls the subroutine at location 103 (259). For example, when the statement

```
2030 X=USR(25.7)
```

is executed, 25.7 is placed at 1BF and a CALL is made to 103.

3. Now, by default 103 contains the following Z80 instruction

```
JP    C7E5
```

or in machine language - hex C3E5C7. This unconditional JUMP to the instruction at address C7E5 in BASIC ROM. This default subroutine prints the error message "FC ERROR" (function call invalid) and stops the program. To call <u>your</u> subroutine, you must change the JUMP instruction address to the address of the beginning of your program. Again the instruction after a BASIC Cold Start looks like

ADDRESS	CONTENTS	DESCRIPTION
103/259	C3	JUMP Z80 operation code
104/260	E5	Low part of address
105/261	C7	High part of address

Leave the C3 JUMP, but change the address. If your program was at 312A as we said, you must make the jump to 312A, or

```
JP    312A
```

or in machine language - hex C32A31. It is a good idea to change the two address bytes every time the subroutine is to be called. Use the BASIC POKE statement for this (which requires <u>decimal</u> operands). Put 2A (42) at location 104 (260), and put 31 (49) at location 105 (261):

```
10000 POKE 260,42
10010 POKE 261,49
10020 XX = USR(Y)
```

When the USR function is executed in line 10020, your routine at 312A will be called. It could use the value in variable Y placed at 1BF as input. It could also put another value back as output. This value will be returned to the BASIC statement as the "result" of the USR function. In the above example, the value returned will be placed in variable XX. Note that the short BASIC routine shown above can easily be made into a GOSUB subroutine by adding the statement

```
10030 RETURN
```

Thus, to call your routine you need only say

```
GOSUB 10000
```

4. To terminate your subroutine, one of four things can be done:

   a. Return directly to the Monitor and exit BASIC altogether, eg for catastrophic errors. For Monitor Warm Start jump to address E003. For Cold Start use E000. The user will be shown the Monitor prompt (">").

   b. For lesser errors detected give an FC ERROR message, stop the program, and return to BASIC READY level. This is simply done by jumping to C7E5.

   c. If errors are detected and your routines have displayed the error message(s), you can stop the program and exit directly to BASIC READY level. For a BASIC Warm Start jump to DFFA, for a Cold Start DFFD.

   d. Of course you can return normally to BASIC so it will continue the program where it left off after the USR statement. This is simply done by the RET instruction. Fill in the parm at 1BF first if necessary.

Note that all the Monitor subroutines are available to the Z80 subroutine, including turning the tape on, reading a file, and turning it off; or getting input from the keyboard. See the section on Monitor Subroutines later.

Debugging of the Z80 routine is a little more difficult than debugging BASIC programs. BASIC loses control of the situation and of what you are doing while your routine is running, and can't "keep an eye out" for potential errors as it can within a BASIC program. Great care, desk checking, and modular programming are a must.

An assembly language routine can also use as input and output actual BASIC variables and arrays. Using the pointers in the BCA described earlier, the program can find the variable/array lists and scan for the one(s) with the correct name(s). Then using the floating point or string formats, the values can be examined or changed.

CURSOR POSITIONING

Cursor positioning is the process of moving the cursor (that underscore character) on the screen to locations other than where it usually is when standard BASIC or Monitor video output is done (eg, PRINT, DUMP, etc). This is very useful especially when data is to be placed on the screen but not in a line by line fashion. For example, if a graphic diagram is displayed and certain segments are to be labelled, the cursor can be moved directly to each one and the output generated in a random fashion on the screen. Also many times the usual output statements will destructively erase what is already on the screen. For example, if something is to be printed in the middle of a line but there is information already in the beginning of that line, and output statement will erase it. Cursor positioning to the middle will not.

To perform cursor positioning from Assembly Language or BASIC is quite simple:

1. Decide what line the cursor is to be on. There are 30 numbered 0-29. Call this "l".
2. Decide what column of that line the cursor is to be on. There are 64 numbered 0-63 on each line. Call this "c".
3. Calculate 64xl. This is the offset from the beginning of the screen to the first column (0) of line 1. This is easy in BASIC (Q=64*L). In machine language, just shift 1 left 6 times, or, assuming 1 were in register E:

```
      LD    D,0       ;DE = 01
      LD    B,6       ;TIMES TO SHIFT
   X: SLA   E         ;SHIFT E
      RL    D         ;SHIFT D
      DJNZ  X         ;6 TIMES, DE=64xl
```

Or if 1 were in register pair HL, just execute the ADD HL,HL instruction 6 times in a row to double 1 6 times, or multiply by 64.

4. Find the MWA. This is described in detail earlier. For the examples below, assume register IY points to the MWA for Assembly, and AD for BASIC.

5. At offset 68 hex (IY+68 or AD+104) is 2 bytes where 64xl is to be stored:

```
LD   (IY+68),E
LD   (IY+69),D
```

or in BASIC, POKE the low part (low byte) of the number 64xl (64xl MOD 256) into AD+104, and POKE the high part (byte) of 64xl (INT(64xl/256)) at AD+105. Now, 64xl MOD 256 is just the remainder when 64xl is divided by 256, and this can be calculated as follows in BASIC:

```
905 L2 = 64*L
910 MD = L2 - INT(L2/256)*256
```

To do the POKEs, assuming AD is already pointing to the MWA:

```
915 POKE AD+104,MD
916 POKE AD+105,INT(L2/256)
```

6. At offset 6A in the MWA (IY+6A, AD+106) is 2 bytes where "c" is to be stored. If it were in register A:

```
LD   (IY+6A),A
LD   (IY+6B),0
```

or in BASIC

```
930 POKE AD+106,C
940 POKE AD+107,0
```

BASIC also requires you to put c at location 1BE (398) in the BCA:

```
950 POKE 398,C
```

7. Call the Monitor cursor move routine. This will replace the current cursor with the character which was at that spot ("underneath" it), move the cursor to the requested spot and save the character there. From Z80:

```
CALL E9CC
```

From BASIC the USR technique must be used:

```
960 POKE 260,204: REM HEX CC
965 POKE 261,233: REM HEX E9
970 X=USR(X): REM CALL E9CC
```

8. Now a standard output statement like PRINT can be done and the output will begin at this new cursor location.

With this technique, horizontal and vertical tabbing can also be done.

Horizontal tabbing may also be done in BASIC directly with the use of the TAB(n) function.

Vertical tabbing may be done with Control-Z (down arrow) characters. For example, to tab to line 15 (0-29), home the cursor with a Control-Q - hex 11 - 17 decimal - and Control-Z 15 times (Control-Z is hex 1A, decimal 16):

```
2220 PRINT CHR$(17);: REM HOME
2240 FOR I=1 TO 15
2260 PRINT CHR$(26);: REM DOWN ONE LINE
2280 NEXT
```

PRINT TAB(n) can then be used to tab horizontally on that line.

EXIDY KEYBOARD ARCHITECTURE

The keyboard on the Exidy has a very clever physical (hardware) and logical (software) architecture.

It actually resides on small parts of input and output ports FE (254). It is composed of a potential 80 keys, organized as 16 rows of 5 columns each. For each one of the 16 rows of possible keys (0-F, 0-15, output port FE bits 0, 1, 2, and 3) any one of the 5 columns of possible keys can be depressed (0-4, input FE bits 0, 1, 2, 3, and 4).

For example, row 0 column 0 is ESC, row 9 column 3 is a P, and row 15 column 4 is the = key on the numeric pad. Not all 80 possibilities are in use (about 3 are meaningless). Each of the valid possibilities can assume any one of 5 states:

1. When SHIFT is depressed - upper case, punctuation; no numerics or graphics; cursor arrow keys operative
2. When LOCK is depressed - this is a CAPS LOCK, so upper case letters, numerics, and punctuation are valid, but no graphics or cursor movement keys
3. When CONTROL is pressed - this produces ASCII control characters, some numerics, and cursor movement; no graphics
4. When GRAPHICS is pressed - this is standard Exidy keyboard graphics (codes 80-BF). If SHIFT is also pressed simultaneously, the programmable graphics codes C0-FF are used
5. If none of the above are pressed - standard lower case and numerics and punctuation are used; no graphics or cursor movement

The Monitor ROM area EC1E-EDFD contains the tables necessary to allow the keyboard input routine to translate the row/column of the key pressed into a 1-byte character codes, depending on which of the 5 states the keyboard is in. These tables are actually broken down into 6 tables total: the first is a what-to-do table to calculate the state etc, and the last 5 are the character codes for the 5 states.

PERFORMING KEYBOARD INPUT

To get keyboard input from the user from BASIC or Z80 Assembly Language without INPUT statements, a very useful Monitor subroutine can be used. In fact, this can be done such that the program sees each character as it is typed without having to wait (or ever get) a carriage return (RETURN). For example, a program can react and respond immediately to input commands as they are typed.

From BASIC, characters can be input with the following example assembly routines. Place this simple and relocatable Monitor keyboard routine driver interface at, say, location F0 (240). It can go anywhere, but F0 is a good start.

```
F0: CD15E0 SCAN: CALL QCKCHK    ;Control-C pressed?
F3: C2FADF      JPNZ BASIC      ;Yes, back to BASIC (warm)
F6: CD09E0      CALL RECEIVE    ;No, get input character
F9: 28F5        JRZ  SCAN       ;Nothing yet, continue
FB: 32FF00      LD   (CHR),A    ;Got it, save at loc FF
FE: C9          RET             ;Return after USR call
FF: 00     CHR: NOP             ;Where byte stored for BASIC
```

The routine first checks to see if CTL-C, ESC, or RUN/STOP have been entered, meaning the user wants to quit. If so (Not Zero) back to READY level. If not, the current RECEIVE device (usually keyboard) is scanned for a character. If none (Zero), scanning continues. If found, the character is put at location FF (255). Control is then return to BASIC after the USR call. The following example BASIC program can use this routine:

```
10 PRINT "ENTER CHARACTER"
20 POKE 260,240: POKE 261,0: REM LOC 00F0 IS 240,0
30 Z=USR(Z): REM CALL SCAN
40 REM IF WE GET HERE LOC FF HAS A CHARACTER
50 A$ = CHR$(PEEK(255))
60 IF A$ = "S" THEN STOP: REM STOP IF S ENTERED
70 PRINT A$: REM ECHO THE CHARACTER
80 GOTO 20: REM LOOP TILL S ENTERED
```

These are both simple routines that can be modified to be as fancy as possible.

From Z80 machine language there is no need to necessarily store the character in RAM. It is returned in the accumulator by the RECEIVE routine.

The above programs accept their input from the current RECEIVE device. To set this device the SET I=x command is used.

MONITOR SUBROUTINES

The Exidy ROM Monitor is just packed with very well-written and useful subroutines which can be called from BASIC and assembly language. All are resident in the 4K ROM between locations E000 and EFFF. This is a brief description of all the useful routines, and how to interface to them. Here the addresses will be given in hex of course, but will also be given as a 2-part decimal number in the order necessary to POKE into the USR JUMP vector at locations 260-261.

ADDRESS	DESCRIPTION
E000 0,224	Monitor Cold Start (on RESET)
E003 3,224	Monitor Warm Start (on BYE command)
E006 6,224	Monitor User Cold Start - similar to E000 except HL is input containing what the user wants to use as HIMEM

ADDRESS	DESCRIPTION
E009 9,224	RECEIVE: returns NZ and a character from the current RECEIVE device in the accumulator (A), or Z if no character yet
E00C 12,224	SEND: sends character in A to the current SEND device
E00F 15,224	SERIAL IN: reads a character into A from the serial input device or from cassette tape
E012 18,224	SERIAL OUT: writes character from A to serial/tape
E015 21,224	QCKCHK: returns NZ if Control-C or ESC (RUN/STOP) is depressed, otherwise it returns Z
E018 24,224	KEYBOARD: the RECEIVE routine if SET I=K (default). See E009.
E01B 27,224	VIDEO: the SEND routine if SET O=V (default). See E00C.
E01E 30,224	PARALLEL IN: the RECEIVE routine if SET I=P.
E021 33,224	PARALLEL OUT: the SEND routine if SET O=P.
E993 147,233	CENTRONICS OUT: the SEND routine for SET O=L.
E024 36,224	CASSETTE MOTOR CONTROL ON: will turn motor on and set the baud rate of the requested cassette. MWA+3D must contain the baud rate (00=300, 40=1200) and reg B must contain the cassette number (1 or 2).
E027 39,224	CASSETTE OFF: turns off both tapes
E02A 42,224	TAPE SAVE: Save memory onto tape. MWA+50,MWA+51 must contain the memory address where SAVEing is to start. It must also be pushed on the stack. DE must contain the ending address. HL must point to a byte containing a CR (hex OD). MWA+47 through MWA+4B must contain the ASCII file name; MWA+4D must contain the file type; MWA+52,MWA+53 the GO address if any.
E02D 45,224	TAPE LOAD: load a file into memory from tape. MWA+47 through MWA+4B must contain the file name to load. If a LOADG is to be done, a Z flag must be on the stack, otherwize an NZ flag. Then if the program name is specified, put NZ in the flags, otherwise Z (ie, load the next file on the tape).
E13A 58,225	MONITOR INPUT: will put the command in the command input buffer at MWA+0. IY must point to the MWA. MWA+43 must contain 0 (not Batch).
E1A2 162,225	Will find MWA and put the address in IY without causing screen flicker (only does so during vertical retrace on the TV to avoid DMA conflicts)
E1BA 186,225	SENDLINE: sends an entire line to the SEND device. HL points to the line, which must end in a 00. LF's are always sent when CR's are found

ADDRESS	DESCRIPTION
E1C9 201,225	ERROR: Sends "ERROR" followed by the diagnostic message (which is pointed to by HL).
E1D4 212,225	OVER command processor (CP). Handles all work necessary for the OVER command
E1E8 232,225	Sends 4-byte ASCII equivalent of the 2-byte integer in DE. If DE=3F29, then "3F29" is sent.
E1ED 237,225	Send 2-byte ASCII of byte in A
E205 5,226	Send a CR followed by a LF, CRLF
E23D 61,226	Convert a 1-4 byte ASCII hex number (pointed to by HL) into DE. If HL points to A93 followed by a "Monitor Delimiter" (eg, blank, CR, etc), then DE will contain 0A93. This is the reverse process of the routine at E1E8.
E2D2 210,226	Send as many blanks as the number in B
E4D3 211,228	DUMP CP
E538 56,229	ENTER CP
E562 98,229	MOVE CP
E597 151,229	GO CP
E5A2 162,229	SET CP
E638 56,230	SAVE CP
E6B9 185,230	FILES CP
E78A 138,231	LOAD CP
E845 69,232	PROMPT CP
E858 88,232	BATCH CP
E85C 92,232	CREATE CP
E884 132,232	LIST CP
E8A1 161,232	TEST CP
E98A 138,233	PP CP
E9B1 177,233	Clear the video screen and refresh/rewrite the graphics character set at FC00
E9CC 204,233	Move the cursor to line/column specified in the MWA. See cursor positioning described previously.
E9D6 214,233	Find the cursor. HL is set to the screen address (which starts at F080) and DE is set to the column number.
EB10 16,235	Refresh character set at FC00
EC1E 30,236	Keyboard input tables (to EDFD). See keyboard section.
EDFE 254,237	Character set for the 64 standard graphics 80-BF to be copied to FC00.

SUMMARY

I hope the information found in this manual will help you use the wonderful features of the Exidy Sorcerer to their fullest extent. Exidy obviously went through a lot of design work on both the hardware and software of this machine, and it would be a shame not to take advantage of that effort. Enjoy!

DISCLAIMER

This information obtained for this document was found the hard way, usually by wading through my personal disassemblies (I've written and sell a Z80 Disassembler among other program products for the Exidy) of the Monitor and BASIC trying to figure out what was going on. This was as much a labor of love and for personal enlightenment as for publication.

With this in mind, there is no guarantee or warantee either expressed or implied associated with this manual or the information it contains. The material contained herein proved very useful for my work on the Exidy, and I sincerely hope the same holds true for you. Good luck in your programming!