Exidy Devices and Ports

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

Table 2. Sorcerer I/O Port Assignment

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 S-100 bus expansion facility.

The keyboard is implemented as part of the Z80 I/O port number FE hex (254), input bits 0-4, output bits 0-3. The video screen needs no port but uses the 1920-byte RAM area at address F080 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 preceded by a bit called a start bit, and must be followed by one or usually two 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 forerunners 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 the 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 cabling 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 connector 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 O=S, and all output which would have gone to the screen will go to the printer, until Reset or SET O=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 (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 O=L, and from that point on all output will be routed to the screen and the printer, until Reset occurs or until another SET O=x command is entered.

CASSETTE TAPE FILE FORMAT

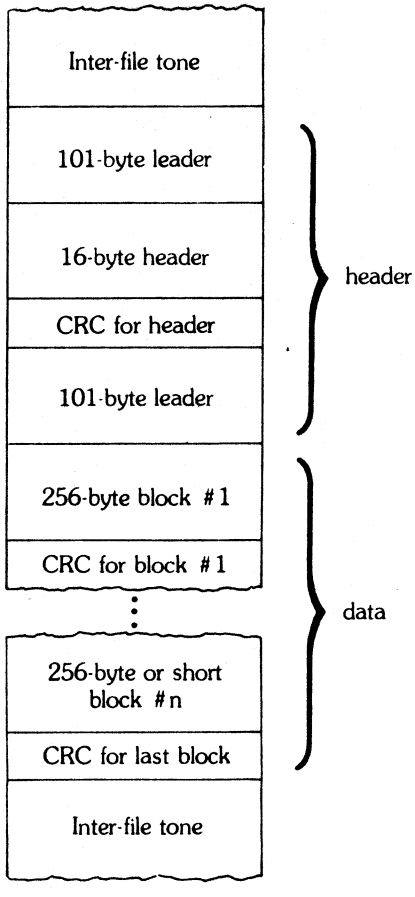
When a SAVE, LOAD, or FILES command is done from the Monitor, or when a CSAVE or CLOAD is done from BASIC, files are processed from the cassette tape device on the serial interface. This applies to both cassette #1 and #2. Cassette tape motor-on routine can be found at E024 (−8156), motor-off at E027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

Table 3. Cassette Tape File Format



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 CRCs 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 four 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 S-100 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 (i.e., 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 (i.e., 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 writen 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.

Performing Keyboard Input

To get keyboard input from the user from BASIC or Z80 Assembly Language without INPUT statements, a very useful 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 (**N**ot **Z**ero) back to READY level. If not, the current RECEIVE device (usually keyboard) is scanned for a character. If none (**Z**ero), 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.

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 (e.g., 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, an 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 "1".
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 64x1. 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 six 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=64x1
```

   Or if 1 were in register pair HL, just execute the ADD HL,HL instruction six times in a row to double 1 six 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 64x1 is to be stored:

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

   or in BASIC, POKE the low part (low byte) of the number 64x1 (64x1 MOD 256) into AD+104, and POKE the high part (byte) of 64x1 (INT(64x1/256)) at AD+105. Now, 64x1 MOD 256 is just the remainder when 64x1 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),O
```

   or in BASIC

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

   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
```

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.

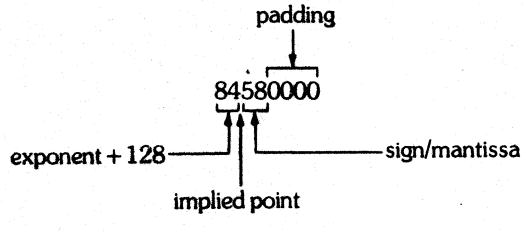
These numbers are stored by BASIC in four bytes of memory. Each number has three parts:

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 four 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 four 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 (four 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 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 four 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***.

.11111111010000000000000

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	=	0061000

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:

00E628B

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 address 100 (256), and has an overall appearance like

Table 5. BASIC Control Area

100	BASIC Control Information
1D5	BASIC Program Source
a	BASIC Program Variables
b	BASIC Program Arrays
c	FREE SPACE
d	STACK
e	BASIC String Space
	Monitor Stack
	MWA

In detail, RAM locations 100-14E (256-334) are copied from the BASIC ROM (address C258) when a BASIC Cold Start occurs (i.e., 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	Three-byte JUMP instruction to C06B (Warm Start). Done when PP command is entered without operands.
103/259	Three-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	Two-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	Two-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	Two-byte BASIC line number of current line.
1B5/437	Two-byte address of the next ***full*** line to execute from the link pointer of the current line (see below).
1B7/439	Two-byte address of the end of the program and the beginning of the BASIC Program Variable Area (letter "a" above).
1B9/441	Two-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	Two-byte address of the end of the Array Area and the pointer to free space (room for expansion — letter "C").
1BD/445	Two-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	Four-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 String Variables and Arrays

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

Offset	Description
+0	Two-byte variable name. The high order bit is always 1.
+2	One-byte current length of the variable length string value.
+3	00
+4	Two-byte address of the string itself. It resides either in the string space or in the program statement itself (e.g., 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 four byte value is not floating point format but the length/00/ stringaddress fields described above. All dimensioning remains the same.

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	Two-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	Two-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 one-byte character not belonging to the ASCII character set (i.e., 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 Variable Area. Each one takes a constant six bytes:

Offset	Description
+0	Two-byte ASCII variable name. The high order bit is always 0. The letters are also reversed as usual.
+2	Four-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 seven 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	Two-byte array name. The high order bit is always 0. The letters are reversed.
+2	Two-byte total array length minus four (i.e., the length of the array starting after these two bytes). This is used to find the next array in the area quickly.
+4	One-byte number of dimensions (we called it n).
+5	Two-byte size (number of elements) in the first dimension.
+7	Two-bytes sie of the second dimension (if any).
•	•
•	•
•	•
+5+2(n-1)	Two-byte size of the nth dimension.
+5+2n	Beginning of a list of contiguous four-byte floating point array elements. These are in Row order.

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. Example:

```
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.

Exidy Monitor Memory Map

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

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

Be aware also that this is an ***overall*** wide angle view of m Detailed maps of certain areas (such as the Monitor Workar the BASIC Control Area) are included.

Table 6. Monitor Memory Map

Address		Description
0000	0	256-byte Z80 Restart space (RAM)
0100	256	User RAM start, begin BASIC Control (RAM)
1F00	7936	8K Monitor Stack end (8K machines) (R/
3F00	16128	16K
7F00	32512	32K
1F90	8080	8K Monitor Stack start (8K machines) (R
3F90	16272	16K
7F90	32656	32K
1F91	8081	8K Monitor Workarea start (8K mach (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 (e.g., begin BASIC) (R
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 alphanum (00-7F) (PROM)
FC00	−1024	512-byte Exidy keyboard standard grap character set, accessed by depres GRAPHICS key, character codes hex 80 (128-191) (RAM)
FE00	−512	512-byte User Programmable grapl character set, accessed by depressing SH and GRAPHICS keys, codes hex CO (192-255) (RAM)
FFFF	−1	End Exidy address space (64K)

Table 7. Monitor Workarea

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 OD, 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 form 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 EOED, −7955).
+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 (**C**yclic **R**edundancy **C**heck). 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.
+6C	FD	Last character entered from the keyboard. This 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.

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. A
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 ther
Here the address will be given in hex of course, but will also be given as a two-part decimal number in the order necessary to POKE into the US
JUMP vector at locations 260-261.

Table 8. Monitor Subroutines

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.
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 0D). 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, otherwise an NZ flag. Then if the program name is specified, put NZ in the flags, otherwise Z (i.e., 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. LFs are always sent when CRs are found.
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" (e.g., 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.

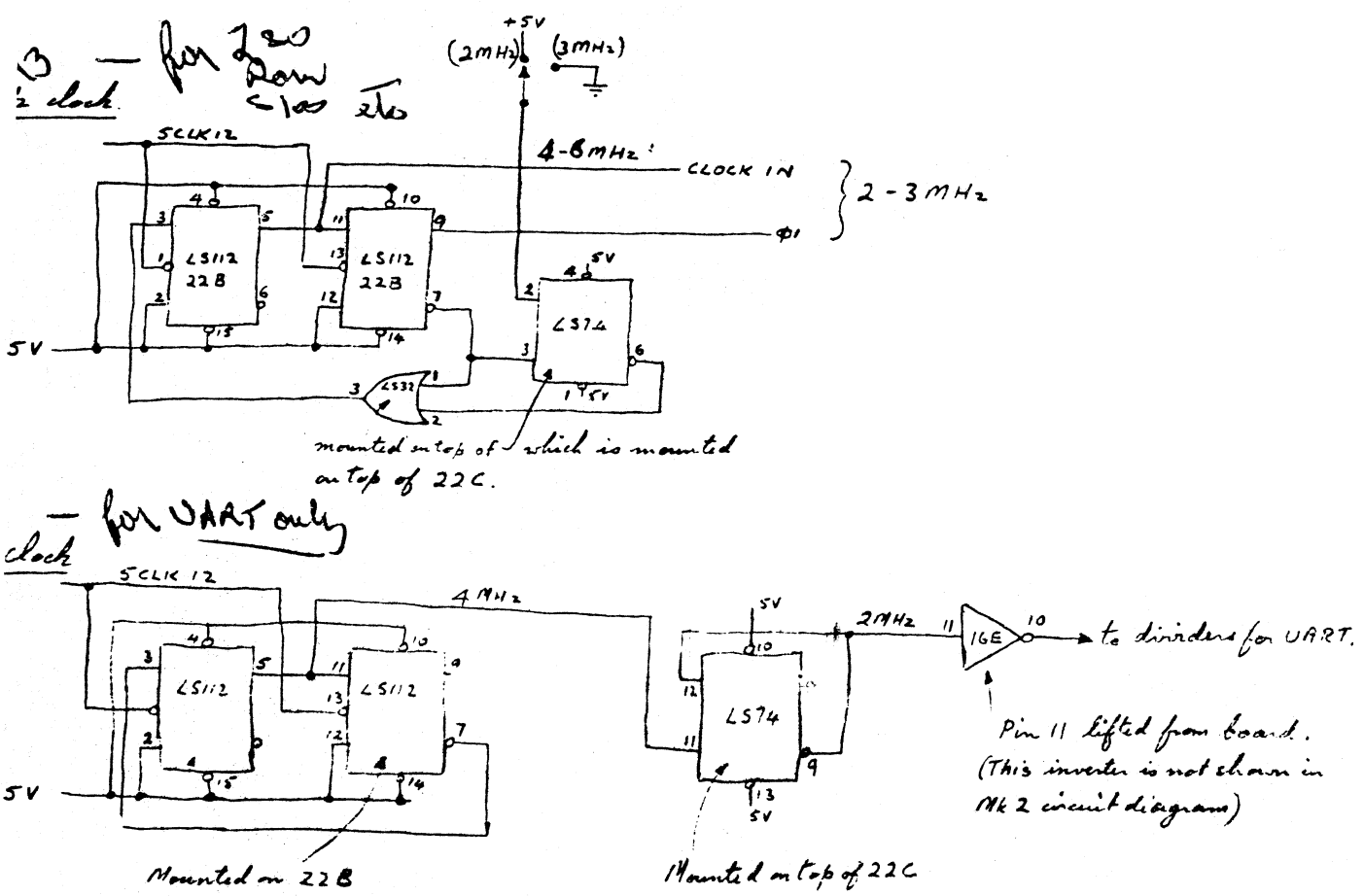
11. Have a cuppa.

I have not had to junk any Z80 nor RAM chips (yes, even 350 nS) in the half dozen machines that this mod has been done to. RAM timing in the Mark 1 and Mark 2B boards has occasionally needed adjustment though. One 2B needed a C4 mod to run reliably at 2 MHz, let alone 3MHz, but the chips stayed unchanged. I intend to publish my experiences and circuit diagrams of the differences that I have found between the various models soon.

Finally, a caveat. At 3MHz the 12 volt regulator may need to supply >200mA for the RAM, cf. about 140mA at 2MHz, depending also on RAM chip speed, the faster chips usually drawing more current. In the Mark 2 with an unmodified power supply, the regulator gets very hot due to the proximity of the resistor (radiator!!), and may fail. The easiest fix is to raise the 5 ohm resistor about 1.5 to 2 cm above the board. In doing so the board and regulator is not heated by the resistor. No additional heatsinking is then needed.

Thanks to David Woodberry for his original ideas out of which this article arose.

Rick Millicer.



UPDATED DESCRIPTION OF ROM PAC BASIC WORK AREA 59

The following was sent to me by Vic Tolomei in response to an inquiry I made of Exidy asking for more information on the ROM PAC Basic work area. This supplements the information given in the Software Manual.

Starting Address	Length (Bytes)	Description of location
100	3	Z80 "jump" instruction to warm start Basic - used if Basic work area is "SAVEed" as a CP/M transient program
103	3	Z80 "jump" to USR function address- default value jumps to "FC ERROR" routine
106	3	Variable Z80 "OUT n" instruction - used by Basic "OUT" command
109	14	Fast 4 byte subtract routine - used by floating point divide tor speed
117	35	Pseudo-random number data - used as tables and counters by the Basic "RND" function
13A	4	Last pseudo-random number generated by "RND", in floating point for RND(0)
13E	3	Variable Z80 "IN n" instruction - used by Basic "INP" command
141	1	Number of ASCII nulls (00 hex) to print after a carriage return - default 01
142	1	Terminal line length - default 64
143	1	Column number of last "PRINT" with comma field, start of last 14 byte field
144	1	Suppress output flag: 0Ch= output, 0FFh= suppress from control O, until second Control O input
145	2	Pointer to top of Basic stack
147	2	Current line number (FFFEH during initialization, FFFF in dir ect mode)
149	2	Pointer to start of Basic program text (currently 01D5H)
14B	67	Terminal input buffer - starts with a comma (",") - also dir ect command line
18E	1	Current terminal column position
18F	3	Internal flags and indicators
192	2	Pointer to highest RAM location - HIMEM
1A6	2	Pointer to top of free string space - set by "CLEAR n" command
1A8	2	Internal pointer used in string space garbage collection
1AA	2	Current "DATA" line number
1AC	7	Internal flags and temporary areas

1B3	2	Line number in program set by Control C or (STOP).
1B5	2	Pointer to current statement in program - to be executed next
1B7	2	Pointer to start of variable space (numeric and string) - after end of program space
1B9	2	Pointer to start of array space (numeric and string) - after end of variable space
1BB	2	Pointer to end of RAM storage in use - after end of variable space
1BD	2	Pointer to current item in "DATA" lists
1BF	8	Floating point numeric accumulator for USR routines
1C3	18	Internal storage used for floating point printout and multiplication
1D4	1	Zero to signify end of imaginary first program line

Devin Trussell

EXIDY EXTENDED CASSETTE BASIC....preliminary information. 60

Exidy cassette basic on latest information (July 23) will still not be available until at least September 10. A price has not yet been set for the Basic but "is expected to be significantly higher than that announced a year ago. Work on the program has apparently been completed and allegedly the only hold up is final drafting and printing of the documentation. During my visit to Exidy, Vic Tolomei (head of software development) gave me some information on the new Basic. It is 19K long so would be a little expensive to ROM even if it did not use internal work areas (which I believe it does). The new Basic has all the features of the present ROM PAC basic plus many new abilities. In fact it is supposed to be equivalent to Microsoft Extended Disk Basic without the disk related instructions. However cassette basic tokens are not compatible with those used in the ROM PAC so some sort of program conversion process will be necessary if you wish to run old programs under the new interpreter. High on the list of new capabilities is 17 digit floating point numerical accuracy. Basic lines up to a length of 255 characters are permissible. A complete list of the tokens in the cassette basic is given below. Virtually all the facilities of TRS80 Level II basic are supported, including a screen editor and "PRINT USING"

ABS	AND	ASC	ATN	AUTO	BAUD	BYE
CALL	CDBL	CHR$	CINT	CLEAR	CLOAD	CONT
COS	CSAVE	CSNG	CURSOR	DATA	DEF	DEFDBL
DEFINT	DEFSNG	DEFSTR	DELETE	DIM	EDIT	ELSE
END	EQV	ERASE	ERL	ERR	ERROR	EXP
FIX	FN	FOR	FRE	GO TO	GOSUB	GOTO
HEX$	IF	IMP	INKEY$	INP	INPUT	INT
LEFT$	LEN	LET	LINE	LIST	LLIST	LOG
LPOS	LPRINT	MID$	MOD	NEW	NEXT	NOT
NULL	OCT$	ON	OPTION	OR	OUT	PEEK
POKE	POS	PRINT	RANDOMIZE	READ	REM	RENUM
RESTORE	RESUME	RETURN	RIGHT$	RND	RUN	SERIAL
SGN	SIN	SPACE$	SPC(	SQR	STEP	STOP
STR$	STRING$	SWAP	TAB(	TAN	THEN	TO
TROFF	TRON	USING	USR	VAL	VARPTR	WAIT
WEND	WHILE	WIDTH	XOR	;	↑	'
*	+	-	/	<	>	=

NON CRC TAPE READER 61

In the standard Exidy monitor is a tape reader subroutine. This is tailored by editing part of the monitor PROM, relocated into RAM, to meet specific requirements. Since the assembly listing of the monitor is available in the software manual it is convenient to do a 'move' in such a way that most of the address is preserved (i.e. the first hex digit is changed from 'E' to '6'). The steps required are given below:

(1) Turn off computer, wait 10 sec and then turn back on (to clear memory).

(2) MO E74B E82E 674B (CR)

(3) EN 6755 (CR)

(4) 00 00 00 / (CR)

(5) EN 681F (CR)

(6) 67 / (CR)

(7) EN 682E (CR)

(8) C9 / (CR)

(9) GO 67B8 (CR)

Now play your tape in the usual manner and the bad part of the tape will not cause the Sorcerer monitor to stop the tape reading session.

A good understanding of the structure of BASIC is necessary to perform the next steps because a jump back to the BASIC ROM is not executed and some of the control bytes must be edited in by hand. First the start of dynamic variable, arrays, and strings must be inserted. By entering the address (in INTEL format) of the third zero of the end of the source file three times starting at the address, 0187H (recall that this is marked by the bytes: 00 00 00), this first step is accomplished. Next, check all line numbers (recall these are in HEX and in INTEL format). Then check all linking addresses found just after each line number (again in INTEL format) and finally make sure that a 00 marks the end of each BASIC source statement line. If the tape error is within the source statement itself you can easily edit this with the BASIC interpreter by RUNning the program.

Explanation of the listed steps:

(4) This replaces the jump to CRC error routine with NOPs.

(6) This changes the 'check CRC' routine to its new relocated location at 674EH

(8) To end the loading session and return to monitor.

Alexander and Leon Travis

SORCERER CURVE TRACING PROGRAM 62

This program uses the finest graphic detail available from the Sorcerer

```
100 DIM SC(64,30),D(128),M(1024)
110 FOR J=0 TO 128:D(J)=1024-8*J:NEXT J
120 FOR N=0 TO 7:B(N)=2E(7-N):NEXT
130 FOR U=0 TO 29:SC(0,U)=-2112-64*U:NEXT U
```

BASIC TOKENS THE INSTRUCTION CODES 80

The following, by Denis Wong appeared in the January 1980 Newsletter of the Sorcerer Program Exchange Club

All basic programs are stored from location 1D5 as follows:

01D5 - Low byte
--Next line number address location.
01D6 - Hi byte

01D7 - Low byte
--Line number
01D8 - Hi byte

01D9 - Start of program coded in tokens

Table of Exidy / Microsoft Tokens

HEX	TOKEN	HEX	TOKEN	HEX	TOKEN	HEX	TOKEN
80	END	81	FOR	82	NEXT	83	DATA
84	BYE	85	INPUT	86	DIM	87	READ
88	LET	89	GOTO	8A	RUN	8B	IF
8C	RESTORE	8D	GOSUB	8E	RETURN	8F	REM
90	STOP	91	OUT	92	ON	93	NULL
94	WAIT	95	DEF	96	POKE	97	PRINT
98	CONT	99	LIST	9A	CLEAR	9B	CLOAD
9C	CSAVE	9D	NEW	9E	TAB	9F	TO
A0	FN	A1	SPC	A2	THEN	A3	NOT
A4	STEP	A5	+	A6	-	A7	*
A8	/	A9	↑	AA	AND	AB	OR
AC	>	AD	=	AE	<	AF	SGN
B0	INT	B1	ADS	B2	USR	B3	FRE
B4	INP	B5	POS	B6	SQR	B7	RND
B8	LOG	B9	EXP	BA	COS	BB	SIN
BC	TAN	BD	ATN	BE	PEEK	BF	LEN
C0	STR$	C1	VAL	C2	ASC	C3	CHR$
C4	LEFT$	C5	RIGHT$	C6	MID$		

In addition, punctuation (ie, :;,") numbers leters and spaces are coded in ASCII

Here is a short program:-

```
10 LET A=12:LET B=13
20 PRINT A+B
30 END
```

This program will be stored as follows:-

```
E7  01  0A  00  88  20  41  AD  31  32  3A  88  20  42  AD  31  33  00  F1  01
 NLN     10    LET SP  A   =   1   2   :   LET SP  B   =   1   3   EOL     NLN

14  00  97  20  41  A5  42   00  F7  01  1E  00  80  00  00  00  0A
 20   PRINT SP  A  +   B   EOL  NLN    30      END   EOL       END OF PROGRAM
```

CLEARING MEMORY 85

Bert King (W.A.) wrote and pointed out that the whole memory can be filled with "0"s by using the following procedure:

(a) BYE (CR)
(b) TE 0 FFFF (CR) (RUN-STOP)
(c) RESET (to re-establish BASIC work area.

Bert mentions references to clearing memory by switching the machine off for 10 seconds, and suggests the above procedure as a substitute, but note that this is usually done to protect ROM-PACs, which may be damaged if removed or inserted under power, including charged capacitors.

D. Trussell

RELOCATING BASIC PROGRAMS 86

Basic programs may be run at addresses other than the normal 1D5hex., providing that the normal Basic Work Area (0100-01D4hex) is left undisturbed.

In order to achieve this, the pointer at 0149hex must be altered to the new start address; in this example, 0300hex.

BYE (CR) EN 0149 00 03 / PP (CR). You will now have READY.

Next, enter NEW (CR). This will re-establish the other pointers.

Now, enter your program from the keyboard in the normal way, RUN it, and CSAVE it. Note, however, that before loading a program CSAVED from the altered location, you will need to alter location 0149 as described above.

Editors Note — Peter Helmick

I think that it should be possible to convert existing (1D5) programs to an alternative starting point by loading them from the Monitor with the LO command to the new location, then setting 1B7-8 to the new end-of-program address, which should be New Start Address + Block (from the header). I feel that this should be a valuable technique for creating "protected memory" for Basic, so often needed for fancy printer drivers, and other Z-80 code utilities. A follow-up article on this, Peter?

Ian Macmillan

"HEXPAD"...or is it "MEMORY CHANGE"? 87

This is a really first class program! Everyone who uses it wants it, and if you do any Z-80 coding at all, not only do you want it, you need it!

Paul Grimshaw, of SPEC wrote the program, calling it "Memory Change", but somehow it has become known as HEXPAD, so you can make up your own mind as to what it should be called.

Peter Lyons typed it in as a Z-80 file from the listing in SPEC, and made sure that it runs. The Hex. dump in this article was converted from an actual memory dump directly into a Word Processor file, so that the listing has a good chance of being error free.

I had considered publishing a dis-assembly, but for reasons of space decided against it, especially as most people who will need this program this will probably have a disassembler. If not, get Ivan Reid's DISAS from the library.

The only point not covered in the instructions included in the program, is that (CR) returns you to the first entry. The program is run (in Monitor) by entering GO 6E00.

have on leaving the routine.

I.M.

WHAT IS PORT FD? 104

Peter Wong of Warrnambool (Victoria), wrote saying that he had examined a disassembly of the Monitor, and having found the statements LD A,FF and OUT (FD),A at E062 and E064, wanted to know:

1. What are the bits of Port FD used for?

2. Why does the monitor, on starting, output 0FFH to Port FD?.

The answer Peter, is that the Sorcerer is equipped with a software controllable UART, which means that the format of the characters sent to tape or RS-232 output, and those acceptable on input, are under the control of the user.

The UART also makes certain information available to the user at Port FD Input.

The Universal Asynchronous Receiver And Transmitter takes a character in parralel form and sends it out in serial form, as a group having a fixed number of bits in a given time.

The number of bits per second in a character sent is the Baud rate, and this is not affected by spaces between characters.

The Sorcerer serial "word" normally has a Start bit (always "0"), eight Data bits, and two Stop bits (always "1")

This format is controllable by sending a byte to Port FD, with the bits having the following significance:

BIT 0: No. of data bits (1)... see table below.
BIT 1: No. of data bits (2)... see table below.
BIT 2: No. of Stop bits...if "0", is 1; if "1", is 2 stop bits.
BIT 3: Parity Select...if "0", is Odd; if "1", is Even.
BIT 4: Parity On/Off...if "0", is On; if "1", is Off.
BIT 5: Not used...value unimportant.
BIT 6: Not used...value unimportant.
BIT 7: Not used...value unimportant.

TABLE...No. of Data Bits Selected by Bits 0 and 1.

BIT 0	BIT 1	No. of Data Bits.
0	0	5
1	0	6
0	1	7
1	1	8

PARITY BIT

The parity bit in the transmitted character will, if selected "On", be automatically set (or reset) to make the total number of "1"s in the group an even number, if even parity is selected, or an odd number, if odd parity is selected. This is an error detecting system that can be used to detect when the received character differs from that sent.

The Sorcerer I/O routines do not use the parity error detection system, as they use Cyclic Redundancy Checking.

SORCERER INITIALISATION

By means of the instructions 'LD A,FF', and 'OUT (FD),A', the Monitor sends FF (11111111) to Port FD. This sets the serial format to eight data bits, no parity, and two stop bits. If this format is changed, as fov example by setting only one stop bit, the tape made will be unreadable unless the reading Sorcerer's UART is also set for one stop bit.

INPUT PORT FD

If the input port is interrogated by the software, e.g. IN A, (FD) , the bits have the following significance:

BIT 0: Transmitter Buffer Empty..."1" when the data is all sent.
BIT 1: Receiver Data Available..."1" when a complete data word is ready.
BIT 2: Over-run Error..."1" if next character here before last taken.
BIT 3: Framing Error..."1" if received character has no valid stop bit.
BIT 4: Parity Error..."1" if character does not agree with the selected parity.
BIT 5: Not used.
BIT 6: Not used.
BIT 7: Not used.

USEFUL FOR WHAT?

Other machines using similar tape interfaces (e.g. SORD) may differ in the serial bit format, so that reading "foreign" tapes may require that the UART be reset via Port FD Output. This sort of problem may be diagnosed by examining the bits of Port FD Input.

Ian Macmillan

SUPER RAM TEST...a bug, and enhancements [105]

Having a consistant fault in my Sorcerer Mk.2 whenever I attempt to select the full 48K, I found the publication of the Super RAM Test in November most timely.

However, as presented, the test cannot output the BAD address (0044 to 0049 in the listing) because the Monitor routine ADDOUT requires the naughty bits to be in the DE register on entry ... not in 'HL'.

I have modified the program to allow correct entry into ADDOUT.

The bytes to change for various memory sizes are 0F and 1E, which contain BF for 48K, 7F for 32K, and 3F for 16K memories.

As an exercise in masochism, I changed the absolute jumps in the original to relative jumps.

Versions of FORTH have been made available from a number of sources including; Forth Inc. (Charles Moore's company); Miller Microcomputer Services (MMSFORTH for TRS80) and the FORTH Interest Group whose FIG-FORTH has made possible the wide distribution of low cost versions of the language.

For more information about FORTH, refer to BYTE, August 1980, which devotes most of the issue to the topic.

S.C.U.A. FIG-FORTH.

This version of FORTH has been implemented by Bob Stafford for members of 'Sorcerer Computer Users of Australia'. It owes its source to the FORTH INTEREST GROUP (F.I.G.), San Carlos, California, U.S.A., which makes available public-domain listings and documentation for FORTH, implemented for 8080, 6800, 6502, 9900, PACE, and LSI-11. They require only that the model documentation should be closely followed, and that credit should be given to them in any published material.

'FIG-FORTH for 8080' is normally a CP/M 8" Floppy Disc based program, but the S.C.U.A. version has been modified to run in a RAM based system with cassette tape program storage, as the majority of members do not have disc systems.

SCUA FIG-FORTH can be converted back for disc operation, but more information is needed than can be found in the documentation supplied with the tape. This, and MUCH more, may be found in two excellent manuals available from F.I.G.:

(a) FIG-FORTH FOR 8080...Assembly Listing.

(b) FIG-FORTH INSTALLATION MANUAL AND GLOSSARY

These are available for US$13.00 each, airmail paid. In addition, membership of F.I.G., which includes its Bi-monthly newsletter "FORTH DIMENSIONS" , costs US$15-00. All this from:

FORTH INTEREST GROUP, P.O.Box 1105, SAN CARLOS, CALIFORNIA 94070, U.S.A.

THE BOTTOM LINE

SCUA FIG-FORTH, For further information members should write to The Librarian, S.C.U.A., Box 144, Doncaster 3108, Victoria, Australia.

Reading TRS-80 tapes with a Sorcerer

108

Introduction

It is possible to read TRS-80 tapes with a Sorcerer with the help of a small amount of hardware and software. This article will show how to do it, concentrating on BASIC tapes, although TRS-80 "System Format" tapes can also be read.

First, though, there are a few caveats to be aware of, as follows:

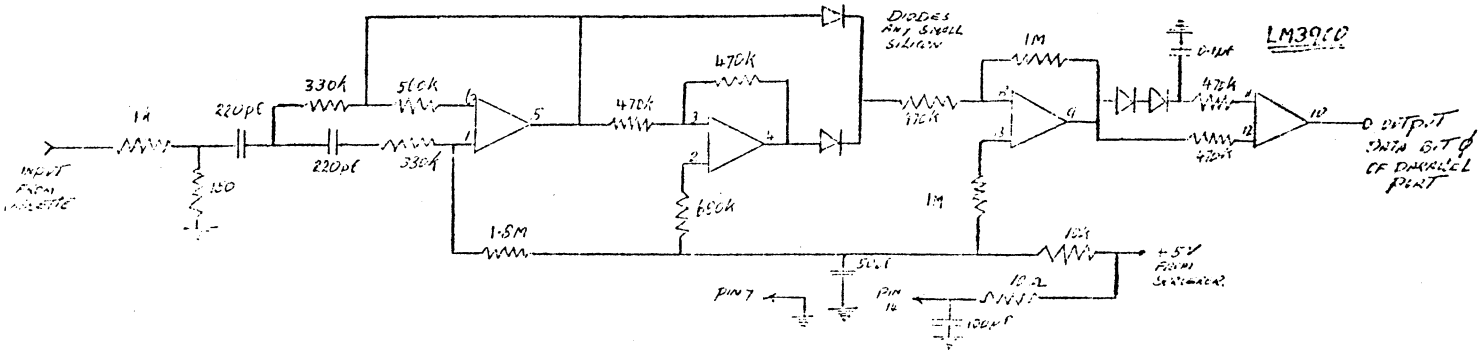
(1) The method handles only Level 2 BASIC tapes. Tapes for Level 1 BASIC are written at a different Baud rate.

(2) The recording method used by the TRS-80 is much more critical than that used in the Sorcerer, which uses the phase-locked tone method. I have found playing levels on the cassette to be extremely critical when reading TRS-80 tapes.

(3) Not all commands and syntax convert exactly. The most obvious ones are : SET, RESET, CLS, ON ERROR, PRINT@, etc. These can all be "coded around", but it has to be done manually. This can be quite laborious in a big BASIC program that uses Graphics!

Part1 - Hardware Considerations.

The cassette recording method used by the TRS-80 consists of recording clock pulses at regular intervals of 1 milli-second; with 1 or 0 represented by the presence or absence respectively of another pulse halfway between the clock pulses.



The circuit shown employs an LM3900 Quad. Op. Amp., in which the four sections are used as a tuned filter to remove hum and noise, an active full-wave rectifier, an inverter and a Schmitt trigger, to process the pulses and feed them to the Sorcerer's parallel input port.

The 1k resistor immediately on the input was used to suit the output of my recorder, and may need to be varied to suit others.

Part 2 - Reading the Data

This code reads the data and stores it in memory, but does not translate the Basic tokens, or correct the line address pointers. This code can also be used to read TRS-80 "System Format" files, but the resultant data in memory has embedded error checking and block control characters, which could be removed in a separate program, which I have not yet written.

The code consists of a main calling section, which performs successive calls to a subroutine which returns one bit of data on each call. After the main program encounters the sequence of the sync. character A5hex, groups of eight calls to the subroutine are repeated to get the data byte by byte.

Control-C is used to interrupt the process. After hearing the end of file on the tape monitor it is necessary to rewind the tape a little, press Control-C, and play some data.

READ TAPE PROGRAM

```
3500 21 D1 01       LD HL,01D1     SET START ADDRESS
3503 00             NOP
3504 CD 50 35       CALL 3550
3507 FE A5          CP A5          SEE IF SYNC BYTE
3509 20 F9          JR NZ,F9(3504) NO,LOOP AGAIN
350B 00             NOP
350C CD 50 35       CALL 3550      PERFORM 8 CALLS TO ....
350F CD 50 35       CALL 3550           GET 8 BITS OF DATA
3512 CD 50 35       CALL 3550
```

```
3515 CD 50 35        CALL 3550
3518 CD 50 35        CALL 3550
351B CD 50 35        CALL 3550
351E CD 50 35        CALL 3550
3521 CD 50 35        CALL 3550
3524 77            w LD (HL),A       STORE THE BYTE
3525 23              INC HL          INCREMENT THE ADDRESS
3526 00              NOP
3527 18 E3           JR E3(350C)     GO AGAIN


SUBROUTINE TO GET A BIT OF DATA

3550 C5              PUSH BC
3551 F5              PUSH AF
3552 00              NOP
3553 06 04           LD B,04         SET DELAY
3555 10 FE           DJNZ FE(3555)       AND WAIT
3557 DB FF           IN A,(FF)       CHECK PARALLEL PORT
3559 EE FF           XOR FF          INVERT IT
355B 1F              RRA             SET FLAG
355C 30 F9           JR NC,F9(3557)  LOOP AGAIN IF NO CLOCK YET
355E DB FF           IN A,(FF)       DO IT AGAIN TO MAKE SURE...
3560 EE FF           XOR FF          ...THAT ITS NOT A TRANSIENT
3562 1F              RRA
3563 30 F2           JR NC,F2(3557)  TRANSIENT,SO WAIT AGAIN
3565 06 3F           LD B,3F         HAVE A CLOCK, SO SET...
3567 10 FE           DJNZ FE(3567)   ...DELAY AND WAIT .....
3569 00              NOP             ...BEFORE LOOKING FOR..
356A 00              NOP             ...DATA PULSE
356B 00              NOP
356C 06 22           LD B,22         SET SEARCH "WINDOW" FOR
356E DB FF           IN A,(FF)       DATA PULSE,THEN GET DATA
3570 EE FF           XOR FF
3572 1F              RRA
3573 38 05           JR C,05(357A)   PULSE FOUND - CHECK AGAIN
3575 10 F7           DJNZ F7(356E)   NO PULSE-LOOK AGAIN IF
                                     STILL IN SEARCH WINDOW
3577 18 17           JR 17(3590)     WINDOW EXPIRED, SO A ZERO
3579 00              NOP
357A DB FF           IN A,(FF)       FOUND A 1, CHECK AGAIN
357C EE FF           XOR FF
357E 1F              RRA
357F 38 05           JR C,05(3586)   DEFINITELY A 1
3581 10 EB           DJNZ EB(356E)   TRANSIENT,LOOK AGAIN IF
                                     STILL IN SEARCH WINDOW
3583 18 0B           JR 0B(3590)     WINDOW EXPIRED, SO A ZERO
3585 00              NOP
3586 F1              POP AF          RESTORE REGS
3587 C1              POP BC
3588 07              RLCA            SHIFT ACCUM
3589 CB 87           RES  0,A        ZERO AND.....
358B 3C              INC A           SET BIT ZERO TO 1
358C C9              RET             DONE!
358D 00              NOP
358E 00              NOP
358F 00              NOP
3590 F1              POP AF          RESTORE REGS
3591 C1              POP BC
3592 07              RLCA            SHIFT ACCUM
3593 CB 87           RES  0,A        SET BIT ZERO TO 0
```

```
3595 F5          PUSH AF        CHECK FOR CONTROL-C
3596 CD 15 E0    CALL E015
3599 C2 03 E0    JP NZ,E003     YES,EXIT PROGRAM
359C F1          POP AF
359D C9          RET            DONE!
```

Part 3 - Translating the Tokens etc.

This program translates the Tokens used for key words. Any byte not able to be translated as a Token is assumed to be a remark. The program is run after using the Data Reading program in Part 2. After running this, it is still necessary to reset the line address pointers. This is done by entering a line (e.g. a REM, or :) at line 0, and then deleting it:

PP (cr) 0 REM (CR) 0 (CR)

Note that this will delete line 0 if it previously existed in the TRS-80 program, so that it is best to check for this by dumping the start of the Basic program (as described in earlier issues of the newsletter), and if line 0 exists, re-enter it instead of the dummy line 0 shown above, and do not delete it.

This is the program, which is run by entering: GO 0(CR)

TOKEN TRANSLATOR

```
0000 E5          PUSH HL
0001 21 D5 01    LD HL,01D5     SET START OF BASIC
0004 23          INC HL         BYPASS LINE ADDRESS
0005 23          INC HL
0006 23          INC HL         BYPASS LINE NUMBER
0007 23          INC HL
0008 7E          LD A,(HL)      LOAD THE BYTE OF BASIC
0009 D6 00       SUB 00
000B 28 05       JR Z,05(0012)  IF EOL,CHECKIF PROG END?
000D FC 30 00    CALL M,0030    IF HEX 80 OR
                                MORE,TRANSLATE THE TOKEN
0010 18 F5       JR F5(0007)    LOOP AGAIN
0012 23          INC HL         BYPASS LINE ADDRESS
0013 23          INC HL
0014 7E          LD A,(HL)      LOAD THE BYTE
0015 D6 00       SUB 00
0017 28 02       JR Z,02(001B)  IF ZERO,ASSUME END OF
                                PROGRAM
0019 18 EA       JR EA(0005)    NOT ZERO,SO GO AGAIN
001B 23          INC HL
001C 22 B7 01    LD(01B7),HL    LOAD END OF PROG ADDR
001F 00          NOP
0020 00          NOP
0021 21 D4 01    LD HL,01D4     RESTORE HEX 00 AT 1D4
0024 36 00       LD (HL),00
0026 E1          POP HL
0027 C3 03 E0    JP E003        DONE!
```

THE TRANSLATION SUBROUTINE

```
0030 E5          PUSH HL
0031 21 50 00    LD HL,0050     LOAD START OF TABLE
0034 F5          PUSH AF        SAVE DATA BYTE
0035 7E          LD A,(HL)      LOAD BYTE FROM TABLE
0036 D6 00       SUB 00
0038 28 0D       JR Z,0D(0047)  END OF TABLE?
003A F1          POP AF         NO,SO RESTORE DATA BYTE
```

```
003B BE          CP (HL)          COMPARE DATA AND TABLE
003C 28 04       JR Z,04(0042)    EQUAL?
003E 23          INC HL           NO,SO LOOK AT NEXT ENTRY
003F 23          INC HL
0040 18 F2       JR F2(0034)      GO AROUND AGAIN
0042 23          INC HL           MATCH,SO POINT TO NEW...
0043 7E          LD A,(HL)        ...TOKEN AND SUBSTITUTE
0044 E1          POP HL           RESTORE BASIC PROG ADDR
0045 77          LD (HL),         SUBSTITUTE FROM ACCUM
0046 C9          RET              DONE±
0047 F1          POP AF           NO MATCH,SO RESTORE....
0048 E1          POP HL
0049 3E 8F       LD A,8F          AND TREAT AS A REMARK
004B 77          LD (HL),A
004C C9          RET
```

Translation Table

The translation table for Basic Tokens is listed below. It consists of pairs of bytes, the first of which is the TRS-80 Token, and the second the equivalent Sorcerer Token. The Table is terminated by a 00hex in the first byte of the last pair.

```
ADDR   0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F

0050:  80 80 81 81  86 B7 87 82  88 83 89 85  8A 86 8B 87
0060:  8C 88 8D 89  8E 8A 8F 8B  90 8C 91 8D  92 8E 93 8F
0070:  94 90 A0 91  A1 92 B1 96  B2 97 B8 9A  BC 9E BD 9F
0080:  BE A0 CA A2  CB A3 CC A4  CD A5 CE A6  CF A7 D0 A8
0090:  D1 A9 D2 AA  D3 AB D4 AC  D5 AD D6 AE  D7 AF D8 B0
00A0:  D9 B1 DA B3  DB B4 DC B5  D0 B6 DE B8  E0 B9 E1 BA
00B0:  E2 BB E3 BC  E4 BD E5 BE  F3 BF F4 C0  F5 C1 F6 C2
00C0:  F7 C3 F8 C4  F9 C5 F8 C6  00 00
```

Note... by altering the Translation Table, this progran could probably be used to translate the tokens to be used by Cassette Basic (when it arrives), as apparently these will be different to those used in the ROM-PAC Basic.

Part 4 - Setting Up and Running TRS-80 Tapes.

A little trial and error is required to set the playback level correctly. If the level cannot be satisfactorily set, alter the data address value loaded in the data reading program to F080 heX, run the program, and you will be able to see what is being read on the screen. Tokens and line numbers will print as graphics, but numbers, literals etc will show up clearly.

Steve Fraser

RS-232 DRIVER...problems, problems. 109

I must inform other users that my RS-232 driver, published in the December 1980 newsletter has problems and can be replaced with a much smaller routine. This new routine will work ONLY if BOTH input and output are set to serial. This will prevent the Keyboard and QUIKCK routines from resetting the offending bit. This routine is, of course, not necessary with the new monitor.

```
0000     F5        SEROUT   PUSH AF
0001     FDE5               PUSH IY          ;SAVE REGISTERS
0003     FD7E3D             LD A,(IY+TAPES)
0006     CBFF               SET 7,A          ;SET RS-232 BIT
```

```
200 B$=RIGHT$("00000"+B$,5)
210 FOR A=0 TO 4:POKE(474+A),ASC(MID$(B$,A+1,1)):NEXT
215 REM
          OPERATOR ROUTINE

220 POKE -3988,16:POKE-3987,240
225 POKE 32720,22:POKE 32721,192
230 POKE 32722,96:POKE 32723,240
250 INPUT A$
270 POKE 32722,28:POKE 32723,235
275 POKE 32720,240:POKE 32721,233
280 LIST 1000
290 REM
          DATA LINES

1000 DATA1234
1010 DATAPOO TO YOU
1020 DATA12
1030 DATA12,13,14,15
```

Ian Macmillan

CURSOR DESTROY! 114

This is really just an improvement on John Quirk's routine described in the July Newsletter (No.14) which works pretty well, but can still leave the cursor on the screen for a few microseconds, leaving occasional 'streaking' on the screen.

My version, like John's, can be located at any address, and is used by changing the output vector to the starting address of the routine. This is done from Monitor by SE O=xxxx where xxxx is the start address, which in the example is 0000.

```
0000    FD E5      PUSH IY              ;
0002    CD A2 E1   CALL GETIY           ;
0005    E5         PUSH HL              ;
0006    D5         PUSH DE              ;
0007    F5         PUSH AF              ;
0008    C5         PUSH BC              ;
0009    CD 1B E0   CALL VIDEO           ;
000C    CD D6 E9   CALL PTRSET          ;
000F    FD 7E 67   LD A,(IY+PTRSET)     ;
0012    77         LD (HL),A            ;
0013    C1         POP BC               ;
0014    F1         POP AF               ;
0015    D1         POP DE               ;
0016    E1         POP HL               ;
0017    FD E1      POP IY               ;
0019    C9         RET                  ;
```

Craig McFarlane

SORD TO SORCERER, AND SORCERER TO SORD MACHINE CODE TRANSFER. 115

Here is a simple way in which a SORD can be connected to a Sorcerer. It uses only 16 lines (8 input and 8 output data lines) and a minimal amount of software.

On the SORD the PIO outputs are open collector, so instead of adding pull-up resistors I have decided to cheat and use the printer ouputs instead. If 4

however, you wish to use the PIO outputs there are only slight modifications to the software.

The Sorcerer's PIO outputs are already provided with their own TTL outputs so these are okay to use without pull-ups.

Instructions

To accomplish transfer of machine code from SORD to Sorceber or Sorcerer to SORD. (Programs for input and output can go into either machine since the programs are designed for use in both machines).

1. Place the input program anywhere in one computer. (The programs will run anywhere since they are completely relocatable).

2. Place the output program anywhere in the other computer.

3. Place the address of the first byte to be inputted in the spots indicated, and in the output program place the address of the first byte to be outputted.

4. Place the high order byte of the program to be inputted in the spots indicated, and in the other program insert the high order byte of the progam you are outputting.

5. Now run the output program, then go over to the other computer and run the input program. Nothing will happen until you have run both programs, but you must run the output program first. The operation is very fast (almost instantaneous for short transfers!)

NOTE:- near the end of each program you will see a JP Z,MON instruction, be sure to put in the correct jump address in each program. In the SORD the monitor jump address is C013. In the Sorcerer the address is E000. Remember the ouput computer goes first. Then the input computer runs its program. If everything goes correctly the computers should both stop at exactly the same time. If not the output computer did not go first or the program timing loop is off. If the memory is not transferred corectly check all connections thoroughly (eg. check they are plugged in!). If the timing loop is off change the delays where indicated in the program. You cannot make the loops too long and make it malfunction, this only happens if it is too short. Remember though, shorter delays give a higher data transfer rate.

Hardware required

One DB-25 male plug, and sixteen wires.

Make the following connections

SORD PIO connections (INPUT)			SORCERER PIO (OUTPUT)	
IN 0	pin 34	to	OUT 0	pin 16
IN 1	32		OUT 1	17
IN 2	30		OUT 2	18
IN 3	28		OUT 3	19
IN 4	26		OUT 4	7
IN 5	24		OUT 5	6
IN 6	22		OUT 6	5
IN 7	20		OUT 7	4

SORD PRINTER (OUTPUT)			SORCERER PIO (INPUT)	
OUT 0	pin 9	to	IN 0	pin 10
OUT 1	10		IN 1	22
OUT 2	11		IN 2	11
OUT 3	12		IN 3	23
OUT 4	13		IN 4	12
OUT 5	14		IN 5	24
OUT 6	15		IN 6	13
OUT 7	16		IN 7	25

Software

```
                              INPUT COMPUTER
                              --------------
21 XX XX                      LD HL,START        ;HL=START OF PROGRAM TO
2B                            DEC HL             ;PROGRAM TO BE OUTPUT
AF                 RESTART    XOR A
D3 61                         OUT (61H),A
06 0C                         LD B,0CH
10 FE                         DJNZ $
3E 80                         LD A,80H
D3 61                         OUT (61H),A
06 0C                         LD B,0CH
10 FE                         DJNZ $
DB 41                         IN A, (41H)
77                            LD (HL),A
23                            INC HL
3E FF                         LD A,0FFH
D3 61                         OUT (61H),A
06 05                         LD B,5             ;CHANGE LENGTH OF DELAY HERE
10 FE                         DJNZ $             ;AT MOMENT DELAY SET FOR A
7C                            LD A,H             ;4 MHZ SORD
3D                            DEC A
FE ZZ                         CP PAGE            ;HIGH ORDER BYTE OF LAST
CA 13 C0                      JP Z,MON           ;ADDRESS OF PROGRAM HERE
18 DC                         JR RESTART


                              OUTPUT COMPUTER
                              ---------------
21 YY YY                      LD HL,START          ;LOAD HL WITH START OF AREA
E5                            PUSH HL            ;TO BE INPUT TO XX XX
DB 41              LOOP       IN A,(41H)
B7                            OR A
20 FB                         JR NZ,LOOP
78                            LD A,B
D3 61                         OUT (61H),A
DB 41              TEST       IN A,(41H)
FE FF                         CP 0FFH
20 FA                         JR NZ,TEST
AF                            XOR A
D3 61                         OUT (61H),A
E1                            POP HL
7E                            LD A,(HL)
23                            INC HL
E5                            PUSH HL
47                            LD B,A
7C                            LD A,H
3D                            DEC A
FE ZZ                         CP PAGE            ;HIGH ORDER BYTE OF LAST
CA 13 C0                      JP Z,MON           ;ADDRESS OF PROGRAM = ZZ
18 E1                         JR LOOP
```

SORD Software Conversion

To convert software written for the SORD (only machine code) to Sorcerer, here are some helpful hints.

INCREASE LINE LENGTH 131

If you want to print 132 characters on a line, it is necessary to do a bit of POKEing. Maximum line length resides in 322(decimal). Since it is only one byte, the maximum line length you can achieve by POKEing it is 255 characters. To set line length to X characters:-

```
  8 POKE 322,X
```

JIM MALCOLM

KEY-TOUCH RESPONSE 132

I am impressed by those programs which dash off and do something as soon as you touch a key, without waiting for RETURN. Back in the August, 1980 issue, David Woodberry explained this simple way of doing it:-

```
  9 POKE 318,195:POKE 320,224
  :
120 A=INP(9):IF A=0 THEN 120
130 REM  CYCLE TILL A CHARACTER IS PRESSED
140 REM  A HOLDS THE ASCII CODE OF THE CHARACTER PRESSED
150 IF A=89 OR A=121 THEN 250:REM BRANCH ON Y OR y (e.g.)
  :
```

JIM MALCOLM

RIGHT JUSTIFICATION OF A COLUMN OF FIGURES 133

Greg Boot's note in the April, 1981 issue is perhaps the 'Rolls Royce' - trailing zeros and all - but for simple alignment this 'Mini' works well:-

```
  :
160 FOR I=1TO10:X=P(I):GOSUB 400
170 REM P(I) IS AN ARRAY OF NUMBERS TO BE TABULATED
180 PRINT "P(";I;") =";TAB(20-J);P(I):NEXT
  :
  :
400 FOR J=1TO10:X=X/10:IF X<1 THEN:RETURN
410 NEXT:RETURN
```

JIM MALCOLM

CENTRING TEXT 134

This little routine will print the string A$ centred (+/- one character) around C, where C is a character position number across the screen:-

```
  :
220 T=C-INT(LEN(A$))/2
230 PRINT TAB(T);A$
```

JIM MALCOLM

INCLUDING COMMAS IN STRING INPUT 135

If you put a comma in string input, (e.g. YES, THANKS instead of just YES) you get the reply:-

```
EXTRA IGNORED
```

- unless you enclose the whole string in quotes.
"YES, THANKS" is accepted in full.

JIM MALCOLM

CONQUERING TAPE CRC ERRORS 136

Remember those tapes which you have tried in vain to load, but have been defeated by those nasty CRC ERRORs? Well, here may be the answer to your problems.

The way to load these lost programs is to turn off the Cyclic Redundancy Check (CRC). The idea is not new, but is a feature of the SOL-20 computer.

Unfortunately we cannot do it quite as easily as the owners of that machine, but it can be done. The secret is to move the loading routines into RAM, modify some addresses, and change the CRC error jump into the NOP no operation op-code.

I moved the loading routines to high memory for two reasons; most programs load into low memory, and it is out of the way of most of the programs I wanted to run, leaving more working room.

Use the following procedure in Monitor...BYE (CR)

(1) MO E71B E845 671B (CR)

(2) EN 671D (CR) 67 / (CR)

(3) Using the procedure in (2), enter 67 into the following addresses: 6733, 67C1, 6806, 681F, 6841, and 6844.

(4) Use the same procedure to enter 00 into the following addresses: 6755, 6756, and 6757.

(5) Your new NOCRC loader may be saved on tape for future use:
SA NOCRC 671B 6845 (CR)

To use the loader, position your tape to the 'lost' program, type GO 678A and start your tape. The program will load almost like a normal LOad. It is important to remember that the data is not being checked for accuracy, so the program may not work even after you have loaded it. My experience to date with machine language programs has been very good. They have run very well, although some of the text has been misspelled.

Mark Northrup.
(Reprinted from the Sorcerer's Apprentice, March 1981.)

CLEAR n,N...another undocumented BASIC command [137]

H.A.Lautenbach, writing in the January 1981 issue of the excellent Canadian newsletter PORT FE, casually mentions CLEAR n,N ...yet another undocumented and valuable feature in Exidy Basic

As an example, the command CLEAR 100,6000 will clear 100 bytes of string space, and set the Top of Memory (as far as BASIC is concerned) to Decimal 6000; that is 1770hex. This means that you can create a memory protected area above all normal BASIC operations to contain all those extensive Z-80 utilities, graphics blocks, editors, printer drivers, and similar curiosities, for which we often have trouble finding homes

10 LINE VIDEO ROUTINE. [138]

A 10 line video routine may seem a little strange at first, but it can be quite useful. Consider a BASIC program that requires a lot of graphics as well as text input. Do you input all data before the graphics start, complicate your program with a lot of cursor control to prevent scrolling or do you use this routine which allows the text to be done on the bottom 10 lines and leaves the the top 20 for graphics?

This routine is the same as the Monitor 1.0 routine except for two items, (1) Only the bottom 10 lines are used and (2) The Form Feed does not reset the graphics (could be annoying if you're using non-standard graphics±)

VIDEO2 uses parts of the monitor to allow it to fit in the space 10 to FFH. A sample program in BASIC (for a 32K machine) is included to give an example of the use of VIDEO2.

Demonstration Program in BASIC.

```
10 PRINT CHR$(12):POKE 32720,57:POKE 32721,0:POKE -3986,32
20 REM SET UP FOR 10 LINE VIDEO ROUTINE & REMOVE OLD CURSOR
30 PRINT"10 LINE VIDEO DRIVER    by   Mark Little"
40 PRINT:PRINT"THIS IS A DEMO TO SHOW THAT TEXT SCROLLING WILL NOT"
50 PRINT"AFFECT THE GRAPHICS ON THE TOP PART OF THE SCREEN."
60 Q=-3968:FORN=0TO19:FORM=0TO63:POKE Q+N*64+M,42:NEXT M,N
70 PRINT"AS YOU WILL SEE THE SCROLLING WILL NOT AFFECT THE TOP"
80 PRINT"PART OF THE SCREEN."
90 FOR N=0 TO 2000:NEXT N:FOR N=0TO 19:PRINT:NEXT N
100 PRINT "THIS PROGRAM WILL LEAVE THE OUTPUT IN THE 10 LINE MODE"
110 FOR N=0 TO 400:NEXT N:PRINT CHR$(12)
```

MARK LITTLE

MODIFICATION TO PROVIDE A 2400 BAUD CASSETTE INTERFACE 139

It is quite practicable to modify the Sorcerer to provide a recording rate of 2400 baud, although the ability of the cassette recorder in use may present a problem, although . to date I have successfully recorded on a medium price stereo tape deck, a bottom price, modified, Dick Smith unit, and a moderate price Sanyo cassette recorder type M2533B.

A summary of the essential requirements for the increased rate follows, the references being to chip/pin as per the technical manual.

Tape Write

1. Clock is 5C/3 - change to 2400 Hertz. This is acheived by using a changeover switch going from 5C/3 to 5D/14 (1200 baud) or to 3B/12 (2400 baud).

2. Clock to 2C/1 (and 2C/13) - must be 4800 Hz. A changeover switch connects from 3B/13 to 3B/7 (='0') - or to 3A/14 (follows normal logic changes)

3. UART WR - must be 38K4 Hz. A changeover switch connects from 4C/6 to 4C/11 (2400), or to 4D/14 (1200).

Tape Read

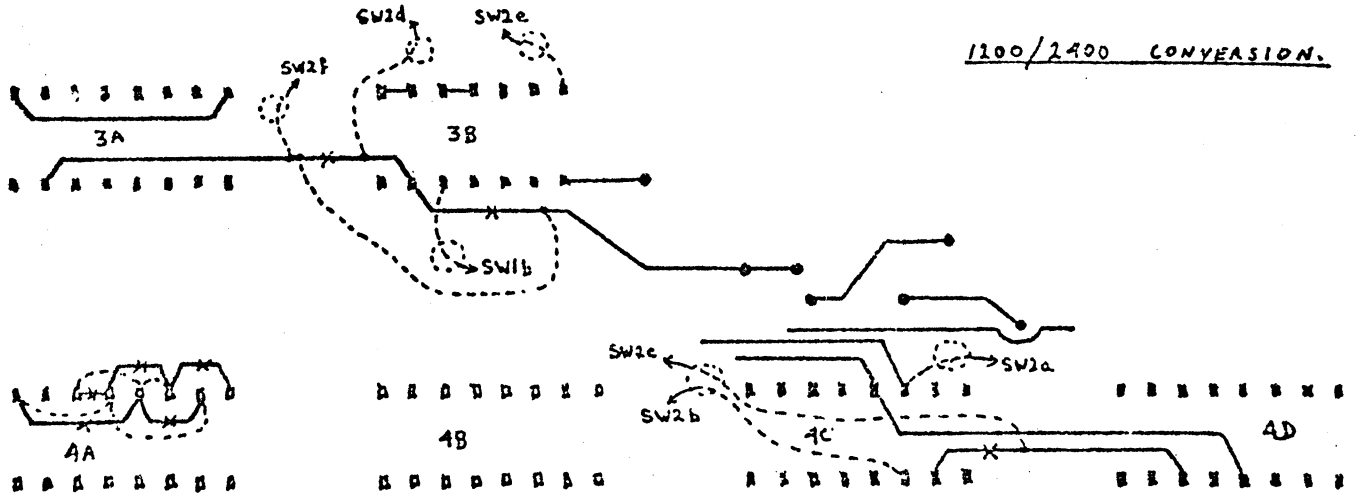
1. Clock pulses to 4A must be 38K4 Hz. To get this, C16 has 1000 picofarads in series - controlled by a shorting switch.

2. The 3C oscillator must be around 38K4 Hz, so that the rate into 3C/3 becomes 38K4/16 = 2400 baud. RD is given by C16 rate - i.e. approximately 38K4 Hz.

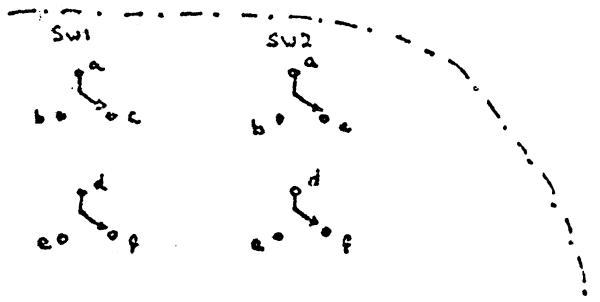
3. Cassette RD must be 38K4 Hz. In the tape read mode everything revolves round the C16 frequency except the clock rate to 4A which is crystal controlled, i.e. the timed interval is independent of incoming data - except for its start.

The circuit changes shown in the drawings include two optional modifications. The first is a cut track bridged by a 1000 ohm resistor plus a shunt 680 picofarad capacitor as shown. This is worth doing even for 1200 Baud machines since it enforces a small delay in the clocking pulse between 5C/6 and 2C/2. Lack of this delay can result in occasional record errors.

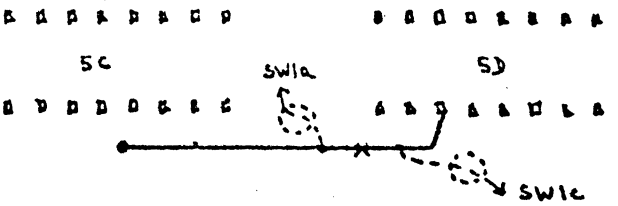
The second modification involves a number of cut tracks plus some additional wires around 4A. This modification changes the timing interval provided by the divider chip 4A and provides a slightly better central position for the sampling pulse which determines whether the incoming tape data is a '1' or a '0'.

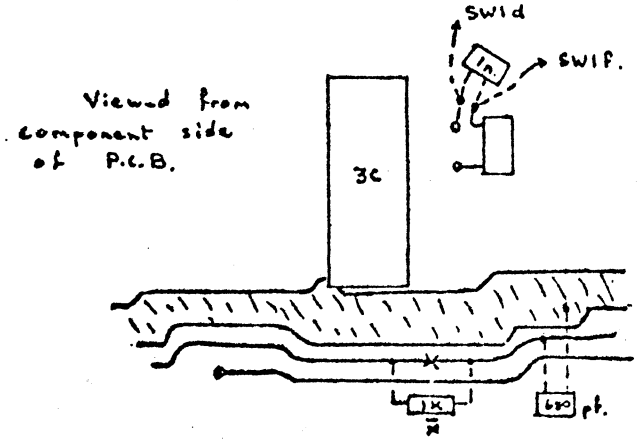


1200/2400 CONVERSION.



Viewed from track side.





Viewed from component side of P.C.B.

= Cut track.

```
)GO 0
TAPE CHECK PROGRAM (1200 BAUD)
Heerlen, 1980  by A. V. Montfoort
NO DATA:  ********
FRAMING:  ***
```

To SAve the program: SA CRC 0000 00BF (CR).

To LOad the program: LOG CRC (CR)

NO DATA: Indicates whether the computer is receiving data.
If the asterisks remain stationery - means no data.
If the asterisks are flashing - means data present.

FRAMING: Indicates if the received data contains errors.
If the asterisks remain stationery - means no errors.
If the asterisks are flashing - means errors present.

A.V.Montfoort (Holland)
(Translated by F.Schuffelen)

SOME BASIC ERRORS OVERCOME.....BASIC 1.1 [147]

During my visit to Exidy, Vic Tolomei provided me with information on Basic 1.1. It is a new version of the ROM PAC Basic. No new features are added, but all known errors in Basic 1.0 are corrected. Those of you who have been clever enought to utilize routines within the Basic ROM will be relieved to know these corrections have been accomplished without moving any routines. This has been accomplished by adding patches at the top of basic. There are still over 950 bytes of free space at the end of the Basic 1.1 so those who wish to add such useful extra utilies such as a video editor, renumberer and program merge should be able to do so. (especially now that an inexpensive Eprom burner is available for Sorcerer see July/Aug80 issue of Electronics Australia). No decision to produce the 1.1 ROMS has been taken yet so availability from EXIDY is unlike to occur until 1981. There is a recommendition that 1.1 ROMS should also be made available to old customers as an inexpensive upgrade kit to replace old chips on older Sorceres. SCUA by an indirect route has come into possession of a tape version of Basic 1.1. An approach will be made to EXIDY to see if they have any objection to it being made available to members so that they may put it into EPROMS.

A list of known problems in Basic 1.0 and their remedy follows.

1.SYMPTOM:(CLEAR), (ESC) and (HOME) produce syntax errors when entered before a command or statement in basic, (CLEAR) or (HOME) placed in a PRINT command will garble the listing during a LIST command,
BUG: The keyboard input routine incorrectly put all entered characters into the Basic input buffer
FIX: The 1.1 keyboard input routine performs the action required by (CLEAR) etc and then throws these characters away. The second symptom is eliminated by using using CHR$(12) and CHR$(17) respectively for (CLEAR) and (HOME).

2.SYMTOM: The cursor "flips" onto the next line before 64.
BUG: The 1.0 input routine counted (RUB) as characters on the screen. The buffer contents are correct but the display is disturbed.
FIX: The 1.1 input routine increments the cursor position count for all characters except (RUB) which decrements the count.

3.SYMTON: Cannot CLOAD a basic tape file program whose filename begins with the letter "G".
BUG: The 1.0 CLOAD routine does checking for a "G" after the CLOAD command. This checking throws away the first character of any filename beginning with "G".
FIX: 1.1 cleans up the check for "G" in the LOG context without wiping the filename.

4.SYMTOM: If lower case is used in the CSAVE and CLOAD program tape file commands they act unpredictably (throw the user into monitor with weird messages etc). Basic is upposed to be transparent to upper and distinctions.
FIX: 1.1 CLOAD/CSAVE code folds all lower case to upper case.

5.SYMTOM: The 1.0 keyboard routine allows the user to type more input characters than there is buffer space, this can cause program destruction.
BUG: The 1.0 keyboard routine has a tast for buffer overflow, but does not branch on it.
FIX: 1.1 has the branch. Also when the buffer is full the input routine will stop echoing characters.

6.SYMTOM: CLOAD* and CSAVE* are so unreliable that they are useless. This is by far the worst problem with ROM PAC Basic.
BUG: This problem is caused by two independant bugs.
a) CSAVE*/CLOAD* do their own I/O to the serial Z80 ports (FC/FD). They did not use monitor TAPEIN and TAPEOUT routine which work correctly.
b) CSAVE:/CLOAD* used had insignifacant header of 4 bytes of hex B2. Also no CRC checking was done.
FIX: 1.0 CSAVE*/CLOAD* routine were completely thrown out. Entirely new routines were used in 1.1. Lower case is handled properly. The array file format was redesigned to look very similar to program file format (100 of 00hex)01(16-byte header+crc)(100 0f 00hex)01(data+crcs). The array file name which can be only 2 characters long is padded to 5 characters with asterisks. Such a file (with nonalphanumerics characters) is not loadable by standard LOAD or CLOAD commands. Thus only CLOAD* can load an array file. An assembly language program called ARRAY has been developed to enable old 1.0 array files to be put into the new 1.1 format.

7. FIX All errors reported by basic (ie the message ?xx ERROR) will also cause the tape motors to be switched off. This is especially needed by the 1.1 array processing.

8. Olgo updated to indicate Version 1.1 and new copyright of 1080.

Vic Tolomei via D.Trussell

AN X-Y DISPLAY PROGRAMMING METHOD [148]

The main aim of any screen display routine should be to make it as easy as possible to place the desired object into position. It is also important that the testing of the screen boundaries be as simple as it can be made. For example, in a system that uses only the screen addresses it is not a simple task to check if the displayed object is past any of the edges of the screen. The X-Y system, on the other hand, allows the edge limits to be checked by one simple test. Say the coordinates range from 0H to 10H, testing that the object's position is not greater than 10H not only checks that it is not greater than 10H, but also that it is not less than 0H. This comes about because the test assumes an unsigned number and so -1H (FF) is taken to be greater than 10H. It is for these reasons that I chose the X-Y for my routine.

A straight X-Y method is satisfactory for displaying single characters, but as the SORCERER uses a programmable character generator for graphics, in most uses a group of characters is used to display the graphic object. The characters in the SORCERER are of an 8x8 matrix so a graphic object is usually adequate for most games if it is a 3x3 character set, that is 24x24 pixels. So with the aim of easy use in mind, this routine outputs a 3x3 character block to the defined X-Y position.

The one remaining problem is to replace the characters in the old position with spaces. Blanking the old position is done before updating the screen position by using the old coordinates to shift in a 3x3 block of spaces.

SORCERER MONITOR VERSION 1.1 [149]

During my visit to Exidy, Vic Tolomei gave me the following information on the new Sorcerer monitor program. Modifications were made which cure all known bugs in the original monitor program. These changes were made without moving any major routine or subroutine. All code which was added had a corresponding compression of existing code. The only exception to this is the subprocessor which supports the "T=n" parameter of the "SET" command, which was moved two bytes. Thus the new monitor should be totally upward compatible with existing software such as ROM PACS, CP/M etc. Described in the following are the bugs in Monitor 1.0 and how they were fixed.

1. SYMPTOM - Neither Sorcerer I or II seem to work with an RS232 device on the serial port. For example, serial printers appear to hang up or print garbage at any baud rate.

BUG - The 1.0 keyboard scan routine reset the serial port from RS232 mode to cassette mode every time it was invoked to get a character. Thus the RS232 signal was immediately set to idle state and the UART shut down. In Sorcerer I, due to a problem in UART control hardware which caused RS232 idle to be a space (+9V) and not a mark (-9V), the serial device was sent a long string of invalid inverted bits, hanging it up. Sorcerer I required a hardware mod to reverse this idle state logic. In Sorcerer II the above hardware problem was fixed but the reset of the port to cassette still shut the UART down before a byte could be completely sent, causing lost characters and framing errore. This occurred at 300 or 1200 Baud. Note programs which use their own keyboard routine (eg Word Processor PAC).

FIX - The 1.1 keyboard routine maintains the RS232/cassette mode in its proper state. With this Sorcerer I no longer requires the above hardware modification.

2. SYMPTOM - 1200 baud sometimes loses characters or does not work at all even with the above hardware mod. to Sorcerer I.

BUG - The 1.0 keyboard routine, error processing, motor control and elsewhere always resets to 300 baud when invoked. Programs with their own keyboard routines will operate correctly.

FIX - The 1.1 routines maintain baud rate at all times. Note: 1200 baud still cannot work with printers with slow carriages due to data overruns, unless the application program sends nulls at the end of a line or Sorcerer is redesigned to handshake on the serial port. Using the SET S=6 command can be an expediant circumvention of the problem.

3 SYMPTOM - There is no convenient way to set RS232 mode from the monitor command level to use serial devices assuming serial devices work correctly.

BUG - The 1.0 SET T=n command only sets baud rate. There is no commmand for chosing RS232 or cassette mode.

FIX - The 1.1 SET command has two new parameters for this purpose, as follows: SET T=2 - 1200 baud RS232 and SET T=3 - 300 baud RS232. SET T=1 and 2 are unchanged in monitor 1.1. Note that the baud rate and the mode is now set immediately upon completion of these commands, not late when the motor is turned on as in 1.0.

4. SYMPTOM - SET O=S (serial) works only at 300 baud, no matter what SET T=n parameter was used.

BUG - SET T=n did not set the baud rate, just "remembered" it; motor on did it. Motor is never turned on unless cassette is used.

FIX - see #3 above. NEW SET T=n now actually sets the mode and baud rate. Thus to use a serial printer from the new monitor only the following sequence of commands is necessary: SET T=2 and SET O=S. Note that RS2323 and cassette operations are mutually exclusive. Note also that the 1.1 monitor together with Exidy CP/M versions 1.42 and higher will allow CP/M communication with the serial RS232 port. Use the Monitor SET T=2 or 3 command and then simply use the CP/M standard punch device PUN:.eg to list a file on CP/M, SET T =2 or 3, boot CP/M and issue A)PIP PUN:=FILENAME.EXT

5. SYMPTOM - A user program initiated by the GO command hangs 1.0 monitor on RETURN if register IY is modified.
BUG - Command completion processing assumes IY is OK.
FIX - 1.1 will do a full warm start after all command completion (including GO) to reset IY correctly.

6.FIX- Logo updated to Monitor 1.1 Copyright 1979.

7. SYMPTOM - Tape read with tape positioned in middle of file containing a string of 00's incorrectly gives CRC ERROR instead of skipping file to next file header.
BUG- "TAPWT" file leader finder looks for at least ten 00s, but allows any character following as an indicator of the file leader (not just 01).
FIX- 1.1 routine only allows a 01 after at least ten 00s.

8. SYMPTOM- Standard Graphics refreshed when CLEAR key depressed: e.g. this incorrectly destroys any programmed graphics in these positions when the screen is cleared.
BUG- Refresh should only occur from a Cold Start, not for a FORMFEED (CLEAR key).
FIX- Refresh fixed to occur only on cold start.

9. FIX- "PR" command removed from Monitor (Allows change to prompt }) Replaced by subroutine 'FOLD' at E845hex. This takes a character in Register A, and if lower case alpha, converts it to upper case.

10. SYMPTOM- 1.0 command level keyboard input does not allow lower case. Very inconvenient when using W.P.
BUG- 1.0 line input does not fold lower case.
FIX- 1.1 routine now uses FOLD routine. Note lower case now echoes as upper case also.

11. SYMPTOM- "TAB/SKIP" key only generates horizontal tab character (HT, Control I, 09hex.) if shifted. Lower case use incorrectly generates vertical tab (VT, Control K, 0Bhex.). Very inconvenient since VT is rarely used, while HT is heavily used in editors.
BUG- 1.0 key font definition table incorrect for non shifted TAB/SKIP position.
FIX- 1.1 table should update from 0B to 09 codes, but cannot, because WP PACs which look for Control K will no longer work. Bug not fixed.

12. FIX- 1.1 Cassette motor on/off processing fixed to maintain serial Baud rate, and RS232/cassette modes.

13. SYMPTOM- Cannot rub out graphics on monitor control line.
FIX- none needed as monitor cannot 'see' them.

Devin Trussell

READING TRS80 TAPES - A FOLLOWUP *refer item* 108

I have found the TRS80 tape read program which appeared in the February issue of the newsletter to be very difficult to use. The principal problem is the various delays used in the program seem to need adjustment depending on what tape recorder you use. Correct loading of TRS80 tapes is also critically dependent on playback level. The time delays I currently use with a fair degree of sucess are:

ADDR:	3554	50 (not 4C or 04)
	3566	58 (not 3F)
	356D	17 (not 22)

These values were established by Adrian Pett after examination of the signal waveform going into the Sorcerer.

Most users will find it advantageous to move the whole tape reading program to a higher memory location. Some 16K TRS-80 tapes will overwrite the tape reader if it is located at 3500hex. I use 6500hex. as the starting point for the program. It is unfortunate that the hardware-software combination mentioned in the

March issue of the newsletter is no longer offered for sale. I have had much less difficulty using it. I would be interested in the experiences of others in using the tape read program so that when the article is reprinted in the Yearbook it will have maximum utility.

I have also noticed the token translation table for conversion contained several errors (SQR, RND, MID$, USR, and DEF). A correct table is given below. In it I have included several extensions so that fewer TRS-80 tokens are translated as REM. These additions translate RESET as # , SET as ! , CLS as) , ELSE as \ , USING as & , INKEY$ as 1, POINT as (, and FIX as . The added symbols are all standard ASCII characters and they will not occur in BASIC programs except within print statements so they cannot cause confusion.

The translation table is correct for locations 50 through 9F after that the following should be used:

ADDR

```
00A0:  D9 B1 DA B3  DB B4 DC B5  DD B6 DE B7  E0 B9 E1 BA
00B0:  E2 BB E3 BC  E4 BD E5 BE  F3 BF F4 C0  F5 C1 F6 C2
00C0:  F7 C3 F8 C4  F9 C5 FA C6  C1 B2 B0 95  DF B8 82 23
00D0:  83 21 84 5D  95 5C BF 26  C9 7C C6 7B  F2 7E
```

As a final note I would like to mention that the token translator program would have to be modified for use with cassette basic since it has two instructions for each token. Only one of the two instructions is a function (i.e. left parentheses "(" follows the token). There are rumours (denied by Exidy) that Exidy has a Cassette BASIC to ROM-PAC BASIC translator program. However it should not be too difficult to modify Steve Fraser's excellent program to do this.

Devin Trussell

* * * * * * * * * * * * * * * * *

ADDITIONAL NOTES

the IN and OUT lines of the Tandy TRS-80.

So much for the construction of the second version of the programmer. Let us now take a look at the Sorcerer driver software. The software consists of five routines, each of which can be called up at any time, either individually, or by further addition to the program. These routines are called by entering "GOSUB $", where $ represents the number of the routine being called. For example, if you wanted to call routine one then you would enter GOSUB 1 followed by a carriage return.

Routine one checks to see that the EPROM is erased. It does this by reading each address and comparing the contents against FF hex. If any of the data bytes do not equal FF hex then an error message is printed, informing you that the EPROM is not erased.

Routine two reads the data stored in the EPROM into a memory buffer located at 4000. When all the data has been loaded, the message "BYE FROM BASIC TO REVIEW" appears. This means that to review the data, you type "BYE" to exit BASIC and fall under control of the monitor. To see what data is contained in the EPROM, simply enter "DU 4000 40FF". The first 256 bytes will now appear on the screen for review. To see the next 256 bytes change the addresses in the DU command to 4100 and 41FF. I suggest this method since 256 bytes of hex is enough to fill the screen. You can, of course, review the whole 1K block at once by entering the appropriate start and end addresses.

To enter data into memory for subsequent burning into a EPROM use the Sorcerer's "EN" command followed by the starting address of the locations to be programmed. Keep in mind that the data to be burned into the EPROM must reside in the buffer starting from 4000 hex. Since the software here is for burning 2708s, the buffer will start at 4000 and end at 43FF.

Routine three fills the buffer with FF hex so that when you have only a short program to burn into the EPROM, the unused locations will remain at FF (erased). This allows the unused locations in the EPROM to be programmed at a later date, a very useful feature.

Routine four compares the data in the EPROM to that in the buffer, and if there are any inconsistencies, these are reported by a message on the screen.

Routine five is used to burn the data held in the memory buffer into the EPROM. You will note that it also makes use of routines one and four. When called, this routine starts by calling up routine one to check that the EPROM to be programmed is in fact erased. Having checked the EPROM it then calls up a machine language routine which burns the data into the EPROM. The decimal equivalent of the routine appears in the DATA lines at the end of the listing. We

(Continued on page 75)

Here is the listing for the SORCERER

```
0 PRINT CHR$(12):GOTO 15
1 GOTO 26
2 GOTO 32
3 GOTO 38
4 GOTO 41
5 GOTO 48
6 REM ************************************************
7 REM
8 REM SOFTWARE DRIVER FOR E.A. EPROM PROGRAMMER FOR
9 REM USE WITH THE EXIDY SORCERER COMPUTER.
10 REM
11 REM WRITTEN BY   GERALD COHN    -    22/05/1980
12 REM
13 REM ************************************************
15 PRINT"THE FOLLOWING ROUTINES ARE CALLED BY ENTERING"
16 PRINT"'GOSUB' AND THE CORRESPONDING ROUTINE NUMBER:"
17 PRINT
18 PRINT"     1. CHECK TO SEE IF EPROM IS ERASED":PRINT
19 PRINT"     2. LOAD DATA IN EPROM INTO MEMORY BUFFER."
20 PRINT"        BUFFER STARTS FROM 4000 HEX":PRINT
21 PRINT"     3. LOAD MEMORY BUFFER WITH FF HEX":PRINT
22 PRINT"     4. CHECK CONTENTS OF EPROM AGAINST BUFFER":PRINT
23 PRINT"     5. BURN CONTENTS OF BUFFER INTO EPROM."
24 PRINT"        THIS ROUTINE MAKES USE OF ROUTINES 1 AND 4"
25 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:END
26 PRINT CHR$(12)
27 PRINT:PRINT:PRINT:PRINT:PRINT"     CHECKING ......."
28 FOR X=0 TO 1023
29 Z=INP(1)
30 Z=INP(0):IF Z<>255 THEN PRINT"EPROM NOT ERASED":PRINT:END
31 Z=INP(2):NEXT X:RETURN
32 PRINT CHR$(12)
33 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT"     LOADING ........."
34 PRINT:PRINT:PRINT:PRINT"     BYE FROM BASIC TO REVIEW"
35 Z=INP(1)
36 FOR X=16384 TO 17407
37 Z=INP(0):POKE X,Z:Z=INP(2):NEXT X:RETURN
38 PRINT CHR$(12)
39 PRINT:PRINT:PRINT:PRINT:PRINT"     FILLING ........."
40 FOR X=16384 TO 17407:POKE X,255:NEXT X:RETURN
41 PRINT CHR$(12)
42 PRINT:PRINT:PRINT     "CHECKING DATA INTEGRITY ........"
43 Z=INP(1)
44 FOR X=16384 TO 17407
45 A=INP(0):B=PEEK(X)
46 IF A<>B THEN PRINT"DATA ERROR - BYTE # ";X-16384
47 Z=INP(2):NEXT X:RETURN
48 PRINT CHR$(12)
56 PRINT CHR$(12):PRINT:PRINT:PRINT:PRINT
57 INPUT"SWITCH TO READ MODE AND PRESS RETURN KEY";A
58 GOSUB 26
59 PRINT CHR$(12)
60 INPUT"SWITCH TO PROGRAM MODE AND PRESS THE RETURN KEY";A
61 RESTORE:PRINT CHR$(12)
62 FOR X=0 TO 48:READ D:POKE X,D:NEXT X
63 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
64 PRINT"PROGRAMMING IN PROGRESS - KEYBOARD IS LOCKED OUT"
65 POKE 260,0:POKE 261,0:S=USR(S)
66 PRINT CHR$(12)
67 INPUT"SWITCH TO READ MODE AND PRESS THE RETURN KEY";A
68 GOSUB 41
69 PRINT CHR$(12)
70 PRINT"PROGRAMMING COMPLETE - DATA CHECKED AND VALIDATED"
71 PRINT:PRINT:PRINT:PRINT:PRINT
72 PRINT"SWITCH OFF PROGRAMMER AND REMOVE EPROM":PRINT:PRINT
73 DATA 6,255,33,0,64,219,1,124,254,68,202,25,0,126,211,0
74 DATA 205,33,0,35,219,2,195,7,0,120,254,0,200,5,195,2
75 DATA 0,211,2,62,0,254,30,202,46,0,60,195,37,0,211,1,201
READY
```