PORT FE

SORCERERS USERS' GROUP

(Toronto)

P.O. Box 1173 Sta. 'B'
Downsview, Ontario,
Canada. M3H 5V6

SORCERER

Newsletter

The Toronto Sorcerer Users' Group was founded in the Spring of 1979, a handful of willing and eager to learn members.

This newsletter shall at all times keep in mind the goal at its conception. To spread the seeds of knowledge.

Articles printed in this newsletter shall be free for all Sorcerer Users' groups to reprint or comment on as they see fit.

Articles submitted for this newsletter must be in no later than the beginning of the 1st of every month.

June/July 1982 ISSUE

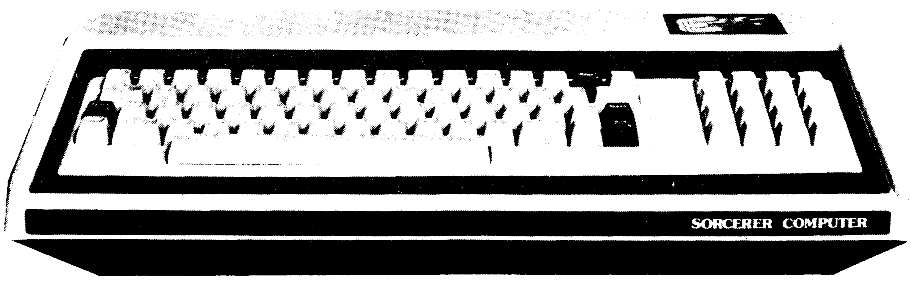
TABLE OF CONTENTS

MEETING PLACE

Location : Bathurst Heights Library - 7:00 PM 3170 Bathurst St.

Wed.June 16 NO Meeting July Wed. Aug. 18 Thur. Sept. 16
Thur. Oct. 14 Thur. Nov. 18 Wed. Dec. 15

One block north of Lawrence on the west side of Bathurst.



'C' LIBRARY ROUTINES by Dereck Gomes

This is our third installment of our 'C' library routines and, this month, we will be looking at two routines that provide us with inverse video of the normal ascii character set. Routine #1 must first be used to create the inverse character set of ascii characters which starts at F800H and put them in the standard and programmable graphics section, then routine #2 will refer tothem simply by adding 128 to the ascii code of the character requested. This is sufficient for most applications, but if you wish to use your graphics section for other uses, then you will have to use another method which will involve creating the inverse characters on the fly and keeping them in a graphics character cell for the duration of its need; a rather more cumbersome method. The method listed in these two routines is more elegant and faster, since you do not have to go through a routine to create it every time you want to use an inverse character; just ask for it.

Note, however, that if you use a routine that uses Exidy's clear screen routine then your graphics section will be re-written and you will have to invoke Invascii() again. It is best to use your own routine to clear the screen.

ROUTINE #1:

COMMENTS: This routine creates an inverse copy of the ascii character set which resides at F800H and puts it in the graphics section for use by routine Inverse().
CALLED ROUTINES: None.
USAGE EXAMPLE: invascii();

```
invascii()            /* setup graphics with inverse of ascii */
{
        char *graph;


        for (graph = 0xfc00; graph ; graph++)
        {
                *graph = ~ ( *( graph - 1024) ),
        }
}
```

ROUTINE #2.

COMMENTS: This routine changes a string of ascii characters pointed to by (ptr) into its inverse video equivalent. Routine Invascii() is a prerequisite for this routine.
CALLED ROUTINES: None.
USAGE EXAMPLE. putdma(2, 3, inverse("this is a string"));

```
inverse( ptr )

        char *ptr;
{
        char *temp;

        temp = ptr;
        while ( *ptr )
        {
                *ptr = *ptr | 0x80;
                ptr++;
        }
        return( temp );
}
```

Tutorial on Standard Rom-Pac.Basic -- Part I

For the next few columns I will concentrate on Rom-Pac basic and how to improve its performance. By the end of the tutorial you will know quite a bit more about what goes on inside that 8 track box.

Quite a few people still don't have disks and are using a 'standard' Sorcerer - this is who I direct these columns to.

First lets find out more about the internal workings of the Pac. Steve Dicker and I had spent two years dissassembling the basic. The following info. comes from this work. I think everyone would like to find out what the Basic Work Area holds and does at 0100H. This table is what we consider its functions to be:

0100	Jump to C06B - Basic warm start (PP)
0103	Jump to C7E5 - USR jump address (C7E5 is FC ERROR vector)
0106	Basic 'OUT' function subroutine
0109	Subtraction routine used by division code
0117	3 byte random number seed
011A	* Unidentified * Looks like floating point data
013A	Current random number in floating point
013E	current 'INP' function routine
0141	Number of nulls+1 to send after CR.
0142	Line length - number of characters before auto return.
0143	Last column of cursor - used for screen formatting
0144	Output suppression flag by Control O (1=suppress output)
0145	Bottom of string space pointer (fills from top down)
0147	Direct/Indirect mode flag (FFFF = direct mode)
0149	Start address for basic text (default is 01D5)
014B	Basic line input buffer
018E	Current print position (column # returned by POS(0))
018F	Flag: 0=Locate named variable, -1=Create entry in table for named variable
0190	String/numeric operation flag (1=string/0=numeric)
0191	Holds intermediate value during expression evaluation
0192	Top of string space (space filled from here down)
0194	Address of next available location in LSPT
0196	LSPT - Literal String Pool
0198	- this area holds pertinent info. on locations
019A	of strings being used by other routines in
019C	the process of manipulation.
019E	
01A0	End of LSPT
01A2	String length - used when printing any string
01A4	String starting location - for printing the string
01A6	Address of where next string should be placed
01A8	Index of last byte executed in current basic statement
01AA	Present line number from which we are reading data
01AC	Used by FOR statement (1=FOR in progress,0=No FOR)
01AD	Last character input into buffer
01AE	Input/Read flag (=0 during INPUT,<>0 during read)
01AF	Address of location of next command in basic to execute
01B1	Adddress of instruction to be executed when ^C hit.
01B3	Current basic line number of line executing
01B5	Address of next full line to execute (link pointer)
01B7	Address of end of program/ start of variable area
01B9	Address of end of variable area/ start of array area
01BB	Address of end of array area/ start of free space
01BD	Address of last used data operand
01BF	Input/Output parameter for USR function and temporary storage area for basic calculations
01C3	?

01C4	ASCII number output-conversion buffer
01D0	Zero byte to indicate end of ASCII buffer
01D1	Temporary storage of MSB of register value
01D2	Temporary storage of next MSB of register value
01D4	Null byte marking start of basic program space
01D5	Start of basic text area

<u>Memory Map of Rom-Pac Version 1.0</u>

C075-C0C5	Opening message
C0C6-C0F5	Command look up tale
C0F6-C1E0	Jump table for basic commands
C1E1-C231	Function look up jump table
C232-C258	Two letter error messages
C258-C2A2	BCA to be moved to 0100H
C2A3-C2A9	'ERROR'
C2AA-C2AE	' IN '
C2AF-C2B6	' READY (cr)'
C2B7-C2BC	' BREAK'
CA1B-CA2D	'?REDO FROM START'
CAFF-CB0F	' EXTRA IGNORED'
C21D-	Precedent tokens

<u>These are basic command jump addresses</u>

C709	END	C62E	FOR	CB34	NEXT	C8B5	DATA
E003	BYE	CA43	INPUT	CD4F	DIM	CA72	READ
C8CC	LET	C872	GOTO	C855	RUN	C944	IF
C6DD	RESTORE	C861	GOSUB	C890	RETURN	C8B7	REM
C707	STOP	D256	OUT	C926	ON	C748	NULL
D25C	WAIT	CF1F	DEF	D705	POKE	C968	PRINT
C735	CONT	C5C6	LIST	C80F	CLEAR	D341	CLOAD
D2C9	CSAVE	C41A	NEW	D7A2	TAB(	D4EA	FN
D54B	SPC(	D8C3	THEN	CD2F	NOT		

STEP and TO have no specific routine but are checked on by the FOR routine itself.

<u>And these are the basic function jump addresses</u>

D606	SGN	D6CA	INT	D61C	ABS	0103	USR
CEE9	FRE	D24A	INP	CF17	POS	D8BA	SQR
D999	RND	D4AB	LOG	D908	EXP	DA0E	COS
DA14	SIN	DA75	TAN	DA8A	ATN	D6FE	PEEK
D18B	LEN	CF9F	STR$	D225	VAL	D19A	ASC
D1AB	CHR$	D1BB	LEFT$	D1EB	RIGHT$	D1F5	MID$

Now I will explain where and how to utilize some very useful routines within the basic for your own basic-machine language combination programs. With this we will be able to write extensions to the rom-pac basic. such examples I will explain next month.

- To display a basic error message:

```
LD   E,ERROR NUMBER     Error numbers (in decimal):
JP   C322H                 16-BS    22-ID    12-OM    02-SN
                           32-CN    00-NF    26-OS    30-ST
                           18-DD    28-LS    10-OV    24-TM
                           08-FC    06-OD    04-RG    34-UF
                           14-UL    20-/0    25-MO
```

- FINDLINE - C3FA - Search for the basic line having the line number held in DE. On return, BC points to the start of the proper line and HL points to the start of the following line. The carry and zero flags are set to indicate the following conditions:

Line was found	(Z,C)	
Line not found	(NZ,NC)	- BC points to the start (link pointer) of a line which has a line number greater than that in DE.
Line not found - end of program reached	(Z,NC)	

- CR/LF - C9BF - send CR/LF and nulls - set 018E to zero

- Print message - D015 - print message pointed to by HL and must be terminated by a null.

- Input line to buffer - C53A - this subroutine accepts input from the current input device and writes it into the basic line input buffer starting at 014CH. An automatic CR occurs after 64 characters have been input. The last character entered is held at location 01ADH. A CR causes the line to be terminated with a null and a CR/LF to be sent to the output device. Hl is left pointing to one byte before the input (014BH) and B holds the number of characters input. IF Control-C is pressed during input, the subroutine returns with the carry flag set.

- Copy - C511 - Move the byte pointed to by HL to the location pointed to by DE and includes both pointers. The move is repeated until HL points to a null or a byte equivalent to that held in B upon entry.

- Translate - C467 - Translates raw input from ASCII codes in buffer to basic tokens and over-writes the input in the input buffer with the compressed instruction. The line is terminated with 3 nulls. On exit, HL=014BH and DE points to the last byte of the packed command string.

- Test sign - D5F7 - Test the floating point variable in the locations 01BF-01C2 and return with the flags as follows:

Number is zero	Z is set
Number is negative	S is set
Number is positive	S is reset

- CMP BC,DE - D672 - Compare the floating point number in B,C,D,E with that held in locations 01BF-01C2(the USR operand space). Returns with; Z set if the numbers are equal.

- Capital? - C7BC - Test the character pointed to by HL. If it is not a capital letter then return with the carry flag set.

- Scan Line - C6CD - This is a very important and useful routine. On entry HL points to the line to be scanned. The line must end in a null. A scan is made until a character other than a space(20H) is found. The subroutine returns with HL pointing to the character, the character in register A and the flags set as follows;

If a null (00) was found - end of line	Z is set
If the character is a digit	C is set
If the character is not a digit	C is reset

- CMP HL/DE - C574 - Compare the contents of HL to that of DE and:

```
    If HL=DE  returns with Z set
    If HL<DE  returns with C set
    If HL>=DE returns with C reset (NC)
```

- ASCII to HEX - C7EA - Converts ASCII code for a number into a two byte hex number in DE. On entry HL points to the first character in the number. If the number exceeds a vlaue of 65529, a SN ERROR is printed. IF HL is pointing to a null on entry, then the subroutine returns with Z set.

- Check Syntax - C57A - this subroutine performs a syntax check. On entry Hl points to a character in the command line which must be a particular character for the instruction to have a proper syntax. The character which it must be is that given by the ASCII code following the CALL SYNTAX instruction. This byte is skipped on return from the subroutine and is therefore not read as an instruction. The process is carried out by the subroutine as follows:

```
    MAIN PROGRAM

    .
XXX        CALL C57A
XXX+3      ","          ; character to be compared to (HL)
XXX+4      --> control returns here
```

1) The CALL places the XXX+3 on the stack.
2) Subroutine loads A with char. pointed to by HL
3) Exchange top of stack with HL (HL=XXX+3)
4) Compare char. in A with that pointed to by HL (A comma "," in this case)
5) Increment HL (the return address) and put it back on the stack, retrieving original value of HL.
6) Now return address is XXX+4
7) Print error message if not equal or return
8) On return, the "," is skipped over.

```
- VARPTR - CDA0 - looks for variable name
                - if found goto CDF5
                - otherwise goes to CDC1 (not found)
                - variable name is in BC
                - on  exit,  DE points to variable value  (4
                  floating point bytes) if found.
```

- Reset Pointers - C426 - this sets up a new stack,resets all the basic pointers and any flags set by basic subroutines.
- Output character - C585 - sends character to current output device
- Input character - C5B4 - gets character from current input device.
- Get 8 bit argument D28D - this subroutine works out an argument pointed to by HL and returns its value in E.

- Get 16 bit argument - CD54 - this subroutine works out an argument pointed to by HL and returns the value in DE.

Ex. POKE 4096*A+1,16/SQR(2*D)

```
    CALL     CD54       ; get 16 bit poke location into DE
    PUSH     DE
    CALL     C57A       ; check for comma
    DEFB     ","
    CALL     D28D       ; get 8 bit argument into E
    LD       A,E
    POP         DE            ; get back poke location
    LD          (DE),A        ; and poke location with 8 bit #
    RET
```

- Work out numeric argument - CB7F - this differs from the above two routines as it works out the argument pointed to by HL then places it into the USROP location at 01C2 in floating point form - not binary. You then 'CALL C7D0' to convert this floating point number to binary. It returns with the argument in DE.

- Work out String Argument - CB93 - This routine will work out a string argument and set up pointers so you will be able to manipulate that string upon return. HL points to the start of the argument in memory. CALL CB93. The argument has been worked out and all pointers placed in the LSPT. PUSH HL then CALL D159. Upon return HL+1 will point to the string located somewhere in memory. HL will point to the length of the string byte. Remember to POP HL.

- Get USR argument into DE - C7CD - this works out the argument in the USR(X) statement and places the value in DE. The value must be between -32767 and 32767

- Save D,A in Z=USR(0) expression - CF0C - this saves a 16 bit value found in A (MSB) and D (LSB) into the USR operand. This value is then placed in the 'Z' of the 'Z=USR(0)' statement upon call to CF0C.

Next month I will show how to turn these routines into useful subroutines for a basic program.

==

THE PREZ ZEZ

The meeting for July is cancelled due to the closing of the Library during the period of our next meeting. This also causes one further problem and that is the August meeting anouncement. This will have to be done in this issue of PORT FE and we hope it hasn't delayed this month's issue too long.

Well I must say that we had a rather good turn out at our June meeting. It seems that quite a number of people turned out to see what the new Monitor was really like.

One of the surprizes that also turned up was one of the New IDS colour printers. One of our members demonstrated what it was like to turn out some multi coloured text, very inpressive indeed. To be sure another person turned up with his entire system as well. So in total we had three systems set up at the meeting. So much for the meeting reports.

I would like to bring your attention to some of the things being neglected by the excecutive of the club.

1. We have some correspondece that has not been answered.
   (please do your best and pick up on some of the things
   that need to be done)
2. Some of the EXCECUTIVE have not shown up for a meeting since
   they have taken office. (this is a disgrace)
   If some of them don't start to show more interest than that
   then we shall have to take appropriate action.
3. I am calling for a complete EXCECUTIVE meeting for Aug 8/82 (Sun)
   my place - 17 Annapearl Ct. Willowdale - 223-9238
   Those that don't show up better have a good excuse. (and phone first)

NOW THAT WE'RE ALL ASSEMBLED

by Joseph R Power

As an assembly language programmer for Systems Research, Inc., I am often required to add new features to a program without increasing the size of the code. This, of course, requires the use of many clever memory-squeezing techniques. Some few of these are so devious they represent the programming equivalent of a pun. Being a punster who can't resist, I decided to share four examples with you. Some of these have been seen before. All are both useful and fiendish.

* Toggling Between Two Values

There are many occasions when your code will need to switch back and forth between two arbitrary values. The normal algorithm for swapping them is:

```
          if LOC = VAL-1
            then LOC := VAL-2
            else LOC := val-1
```

which requires a comparison, two jumps, and two loads. A much smaller, faster, more obscure method is:

```
          LOC := LOC xor (VAL-1 xor VAL-2)
```

where the (VAL-1 xor VAL-2) can often be computed once and used as a constant. If LOC contained VAL-1 then the exclusive or with VAL-1 leaves zero; and the xor of zero and VAL-2 is VAL-2. A corrallry routine allows swapping two values (in A and B) without resorting to temporary variables:

```
          A := A xor B
          B := A xor B
          A := A xor B
```

which leaves the original value of B in A and A in B.

* Carry Bit Extension

There are times when you need to know if a value is zero or non-zero. There is a nifty routine to turn any non-zero value to 0FFh while leaving zero alone. In Z80 mnemonics it goes:

```
          ADD  A,0FFH          ;add FF so anything > 0 will have
                               ;  carry bit set but 0 clears it
          SBC  A,A             ;A := A-A-carry bit
```

The trick of subtracting the carry bit from 0 is also usable for sign extension, provided your instruction set allows you to shift the top bit of the accumulator into the carry bit.

```
          LD   L,A
          RLCA
          SBC  A,A
          LD   H,A
```

puts A into HL and sign extends it (in only 4 bytes).

* Converting Hex Digits to ASCII characters

There are a great number of programs that need a routine to convert hex numbers into their ASCII character representations. The fragment I always use is

```
          ADD  A,090H
          DAA              ;Decimal Arithmetic Adjust
          ADC  A,040H
          DAA
```

to convert a single hex digit in the A register. The DAA instruction is normally only used when doing BCD math (and therefore normally unused). It does some very complex things with both nybbles in the A register. Try tracing through an example or two some time.

* In-Line Parameter Passing

In assembly language, there is usually no way to pass parameters to a routine except by placing them in some agreed upon registers/memory locations. This involves spending a great deal of time and space loading values into these parameter holders. One method that often saves space at the expense of time is to follow the subroutine call with the parameters placed right in the code like:

```
          CALL EXAMPLE
          DEFB parameter-1
          DEFW parameter-2
```

etc. The natural question is how do you use these parameters and not return to them as code? The answer is in the method used to read the parameters:

```
     EXAMPLE:
          EX   (SP),HL    ;POINT HL AT PARAMETERS
          LD   B,(HL)     ;B := PARAMETER-1
          INC  HL
          LD   E,(HL)
          INC  HL
          LD   D,(HL)     ;DE := PARAMETER-2
          INC  HL
          PUSH HL

          main body of subroutine

          POP  HL         ;END OF ROUTINE
          EX   (SP),HL    ;POINT TO CODE AFTER PARMS
          RET
```

All of these methods are tricks (or puns) in the sense that there exist more straightforward methods for accomplishing the same results. But in the real world we often need to play games like these in order to meet some size/speed goal. I would love to have readers of this column submit other examples of programming puns. Dirt must be good; ten trillion worms can't be wrong!

NOW THAT WE'RE ALL ASSEMBLED....   by Joseph R Power

Have you ever considered just how awful the instruction set of the Z80 in your Sorcerer really is? You probably have if you've ever had cause to do any assembly language programming. Well, in this article I'm going to show you some of the more common ways of dealing with the situation of not having all the instructions you need.

Let us start with a specific example. In the Z80 there is an instruction JP (HL) which puts the 16 bit value in register pair HL into the program counter. This has the effect of jumping to the address contained in HL. There is no corresponding CALL (HL) instruction for indirect subroutine access, nor are there any JP <condition>,(HL) or CALL <condition>,(HL) instructions for conditional indirect branching. So as long as you just want to always jump to the address in HL you're fine. All of the other forms, however, are just as useful and, in many cases, much more so. Don't dispair. There is a simple way, with only one byte of overhead, that we can simulate all of these instructions:

```
          GOTO.HL   JP    (HL)
```

Now if we want to do a CALL (HL) we simply use:

```
                    CALL GOTO.HL
```

and the conditional branches ae handled by:

```
                    JP   <condition>,GOTO.HL
                    CALL <condition>,GOTO.HL
```

We now have four sets of instructions where before there was only one.

But this still isn't all that good. We are still limited to just the HL register pair. What about JP (DE) or CALL NZ,(BC)? These too are possible and coding them reveals another technique - finding an equivalent instruction or (more often) sequence of instructions. By using:

```
     GOTO.DE    PUSH DE
                RET
```

we gain the ability to use

```
                JP   GOTO.DE    ; JP   (DE)
                CALL GOTO.DE    ; CALL (DE)
                JP   cond,GOTO.DE    ; JP   cond,(DE)
                CALL cond,GOTO.DE    : CALL cond,(DE)
```

Routines similar to the two above can be coded for BC, IX, and IY. With these five routines we have effectively added 89 new instructions to the Z80!

The last technique I'll discuss for 'extending' your instruction set is by far the most common - using macros to 'create' new instructions. For instance, there is no Z80 instruction to simply compare HL with another register pair. A good macro to perform this function might look something like:

```
;
; CMP.HL  -    Compare HL with another register pair.
;
        MACRO     CMP.HL,RP
        OR    A                 ;CLEAR CARRY FLAG
        SBC   HL,RP             ;SET FLAGS
        ADD   HL,RP             ;DOESN'T CHANGE FLAGS
        ENDM
```

Now you can do a register pair compare by using;

```
        CMP.HL    DE            ;Compare HL with DE
```

Through the judicious use of all three of these methods you can soon have micros with super-powered instruction sets. This makes programming a lot easier.

Quick Reference Sheet: ROM Pac Basic Keyword Tokens

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

WORD PROCESSOR PAC HELPS   by Norm Olsen

Ever wish you could use the word processor pac to produce CP/M compatible files? This would allow you to throw away those awkward editors used for assembler and other program writing. The implementation is simple.

The main difference between PAC files and CP/M files is that the PAC files lack a LINEFEED (0Ah) code. If this code is added to the PAC files, they can be used by CP/M.

The routine for adding the LF is added to the disk driver. First, load the disk driver into memory and then enter the monitor. The driver sits from 100h to 2FFh with the top few bytes not being used. Enter the following code:

```
EN 2E4 CR
02E4: E5 21 0F 08 7E FE 03 20 0A 21 92 01 CR
02F0: 22 F6 07 E1 C3 92 01 FE 0D 23 20 EC 36 0A 18 E8/ CR

EN 112 CR
0112: E4 02/ CR

GO 0 CR    to enter CP/M

A> Save 2 name.COM
```

Make the name different than your regular driver so you can tell them apart. (the above steps can be done in DDT if you wish.)

When you wish a CP/M compatible file, simply load the new driver instead of the old one. When you enter your new text, be sure to <u>double space everything</u> by entering an extra carriage return at the end of every line. (Do not use the word wrap feature. End each line with a CR.) When you issue a disk WRITE command, the new routine goes through the file and changes every byte which follows a carriage return to a 0Ah code. Since everything is doubled spaced, this means the second carriage return will always become the 0Ah code. The CP/M text will be single spaced. DO NOT USE highlighted text as CP/M will not understand it.

When the text comes back on the screen after the WRITE, two things will have happened:
1. Although everything is still double spaced, the carriage return symbols for the blank lines will have disappeared. (This is because the 0Ah codes don't show on the screen.)
2. The next time you write a file, the text will not be converted but will be written as normal word processor text.

To get around problem one, simply give the X command to enter the monitor and immediately PP back to the text. The PAC will convert all the 0Ah codes into carriage return codes and you can see them again. To get around both problems in one easy step, X into the monitor and GO 100. The carriage returns will show up and the next write will do the conversion.

If your assembler or compiler will not use a file with a .WPF extension, REName it with the appropriate extension (.ASM, .BAS, etc.) An existing CP/M file can be edited with the PAC by RENaming the extension to .WPF and reading it as a normal PAC file. This text will already be double spaced.

Even though there are some minor inconveniences, you quickly get used to them and the ease of editing will more than compensate for them.

* *

Lose a word processor pac file and want it back? If it is still in memory, you can get most of it back by doing the following:

```
Press RESET

Command X  into monitor.

EN 80F CR
080F: 20/ CR

MO 80F 8CD 810 CR

PP  back to pac (or boot CP/M and load driver.)
```

You will have three lines of blanks at the beginning which you can delete. You must then reenter the first few lines of text.

When the PAC is booted, it overwrites the first 192 bytes of text with some machine language code. When this code is cleared out, the PAC can then find the end of file and your text is back. If you need to shorten the text, change the end of file by locating the spot where you want it and placing a 03h code there. The first 03h code encountered is used as the end of file.

RCMP utilities - CP/M users' group utility programs.

As some of you are aware, there exists a CP/M user's group in and around the regions of Silicon Valley California. Well it's not just a local sort of thing really, most of the users are spread all over the United States and there also exists a grape vine consisting of Bulletin boards in almost every city in USA. Now wait a minute you say, thats quite a few cities your talking about. Yes I hate to say it but you can find an awful lot of software on those bulletin boards. Within the last year I would say that I personally have gotten about TWO Megabytes worth of FREE software. Everything from games to utilities. Some of which I don't know what I would do without.

Now this leads me to the topic of UTILITIES. For this issue I will list just a few of them for you. In the following issues I will describe them in a little more detail.

CP/M available utilities:

XMODEM	: A modem transfer program
SMODEMX	: A version of XMODEM rewritten for the Sorcerer
SAPX	: Directory sort and re-write program.
SQ	: Squeeze file program. Can save up to 50% disk memory.
USQ	: Unsqueeze file program.
ZCPR	: Z80 CCP for CP/M 2.x supports multi user level.
DU	: CP/M disk doctor.
FMAP	. CP/M disk sector mapping program.
CAT	: Disk cataloguing program.
SWEEP35	: Replaces PIP and has multi user level transfers.
COMPARE	: Compares files for differences. Multiple drive capability.

Here are two examples

SWEEP35.COM	Now this is one of the best known programs on my list. With this program one can rename files, delete files, tag file for transfer from disk to disk or user level to user level, delete all untagged files, check how much memory is left on a particular drive - prior to transferring of files, keeps a running total of how many 'K' are being transferred so that you can fit as much on a disk as is possible, view any text files prior to transfer, retag files for transfers again, sweep all user levels if one wishes, copy files, untag files, complete menu is displayed. A super utility written in PLI and needs about 28K of memory. I wouldn't use anything else for manipulation of files on disk. This will also now verify while transferring files (with CRC check)
ZCPR.MAC	This is CCP for CP/M written in Z80 machine language that allows me to see what user level I am on at all times. The A0> or A8> indicating the user level. Also there is the DIR *.* S command that displays only system files. I think that probably one of the better features is that when, let's say, you're on user level 6 B: drive, and you type in STAT, now STAT is not on user level 6 but only on drive A: level 0, this CCP will search level 6 B:,0 B: and then default to A: level 0 to access STAT. Not bad for a dumb FREE program. It also has other features.

Most of these programs I enjoy using very much. When you're tied into one of these BBS systems you also get the feeling that you're all working together to help each other out. This is why the Sorcerer User's groups were formed all over the world. The more we can participate and ""communicate"" between ourselves the better off we shall all be. Remember what the CP/M user's group is doing, we should be that close knit as well.

by: H.A. Lautenbach

Further Glimpses at EXMON2 by: H.A. Lautenbach

Many are wondering about what's so special about the New Monitor revision by Walter Blady. For many it will be like breathing in new life into your Sorcerer and also your own programming style.

It has not been mentioned before, but another feature that this Monitor has that is a real treat, is that the serial port now becomes a true serial port for 300 or 1200 baud modems and also the SMODEMX program. The toggling of that Bit associated with the keyboard scan routine no longer poses any problem. We are getting many phone calls (long distance no less asking some more complex questions). I would like to take some time to answer some of these in this newsletter.

Questions	Answers
Availability	: No, not though PORT FE.
From whom if not PORT FE	: H.A. Lautenbach same P.O. Box as PORT FE
How much does it cost.	: $65.00 + 5.00 Postage U.S. Funds
Is it compatible with the WPP ROM PAC	:No, direct monitor calls made by the WPP ROM PAC
What happens to the old Monitor chips.	.You can pass them on to never never land OR you can stack the NEW Monitor EPROMS on top of the old ones and have both monitors.
Is it compatible with the ROM PAC BASIC	:Yes fully, no bugs detected.
What terminal does it emulate	:None specific
With the reverse character set do I still have my graphics.	:Yes some of them (not all)
Is the Monitor Jump table compatible with the old Monitor jump table.	:Yes fully (1st 16 are identical) Plus 18 more added.
What have I lost in the Monitor.	:Batch,Test & Files commands
Is it CP/M compatible	. YES (completely, with keyboard status check routine for CP/M)
Will my speed increase in CP/M.	: Depends on how efficient your present status check is. Lifeboat CP/M 1.4 and 2.2 are definitely improved (if original) Also Micropolis CP/M can be improved.
Will all software that requires cursor positioning work.	: Yes, as far as we can determine. As long as it has an install program or system parameter file.
Was anything else changed	: YES many BUGS have been corrected, some not even mentioned ever before.
Has a CP/M boot been provided.	: No, There are too many boot addresses that could and are being used on different disk systems. We thought it wiser not to include this function.
How did you get it all that into the monitor.	: With a lot of squeezing and the making of certain routines more efficient.

Can I control the Sorcerer from a remote ASCII terminal.	: Yes, from both the Sorcerer and the keyboard or terminal that is hooked up via the serial port port on the Sorcerer. (300 or 1200 baud)
Can I control it via a modem at 300 baud.	: Yes, even at 1200 if you have the modems.
Are all the new jump vectors new routines.	: It's about 50/50, some of the jump vectors were brought out to be more accessable to the user.
Is it compatible with MP/M	: We don't know, should be.
Can I use all my graphics if I want to.	: Yes if you want. Reverse ASCII would be overwritten though and would have to be recalled.
Is the reverse ASCII under keyboard control.	: Yes, you use the ESC key and a number.
What about updates. What if there are still bugs.	: 'If' there are any, only a small handling charge will get you revised EPROMS.
Are there discounts for user groups.	: Yes group purchases will be given discounts. 5 to 9 sets 15% ,10 or more 20%

There are probably some that I've missed, but if any of you have more questions, Please direct all enquires to me personally. Most of your ouestions are answered with the documentation that comes with the EPROMS.

For most of you there is a hardware change requiring you to change the ROM/EPROM jumpers to that of the standard 2716 EPROMS and thats all there is to it. In my personal opinion (BIASED OF COURSE) I really think it's the best thing that has happened to the Sorcerer since it was first sold.

I do recommend that everyone needs this capability, for the multitude I'm sure that most of you have done without long enough, The added frills are nice.

PART II ~ The SIO Programming

The Z-80 SIO contains 8 registers that are written to in order.

To initialize: All write registers, with the exception of register '0' are 2 byte instructions. The first byte signifies the register, the second, is the data being sent to it. Upon reset, write register '0' is entered. A single byte here in register '0' takes you through the other registers.

The next page shows all of the registers and the meaning of each bit in each register (0-7). This month please digest this information so that next month, when you are given a simple Hard & Software implementation, you will have a better understanding of how versatile this chip really is.

I am taking this all very slowly so that we don't lose too many people along the way. This can become very involved during the set up procedure.

Next month (August issue this will be continued).

by: Brad Fowles

SIO - REGISTER INFORMATION

WRITE REGISTER 0

D7	D6	D5	D4	D3	D2	D1	D0	
					0	0	0	REGISTER 0
					0	0	1	REGISTER 1
					0	1	0	REGISTER 2
					0	1	1	REGISTER 3
					1	0	0	REGISTER 4
					1	0	1	REGISTER 5
					1	1	0	REGISTER 6
					1	1	1	REGISTER 7
		0	0	0				NULL CODE
		0	0	1				SEND ABORT (SDLC)
		0	1	0				RESET EXT. STATUS INTERRUPTS
		0	1	1				CHANNEL RESET
		1	0	0				RESET RxINT ON FIRST CHARACTER
		1	0	1				RESET TxINT PENDING
		1	1	0				ERROR RESET
		1	1	1				RETURN FROM INT (CH-A-ONLY)
0	0							NULLCODE
0	1							RESET Rx CRC CHECKER
1	0							RESET Tx CRC GENERATOR
1	1							RESET SENDING CRC/SYNC LATCH

WRITE REGISTER 1

D7	D6	D5	D4	D3	D2	D1	D0	
							●	EXT. INT ENABLE
						●		Tx INT ENABLE
					●			STATUS AFFECTS VECTOR (CH-B-ONLY)
			0	0				Rx INT DISABLE
			0	1				Rx INT ON FIRST CHARACTER ONLY
			1	0				INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR)
			1	1				INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR)
		●						WAIT/READY ON R/T
	●							WAIT FN/READY FN
●								WAIT/READY ENABLE

WRITE REGISTER 2*

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	V0

V0–V7: INTERRUPT VECTOR

*CAN ONLY BE WRITTEN INTO CHANNEL B

WRITE REGISTER 3

D7	D6	D5	D4	D3	D2	D1	D0	
							●	Rx ENABLE
						●		SYNC CHARACTER LOAD INHIBIT
					●			ADDRESS SEARCH MODE (SDLC)
				●				Rx CRC ENABLE
			●					ENTER HUNT MODE
		●						AUTO ENABLES
0	0							Rx 5 BITS/CHARACTER
0	1							Rx 7 BITS/CHARACTER
1	0							Rx 6 BITS/CHARACTER
1	1							Rx 8 BITS/CHARACTER

WRITE REGISTER 4

D7	D6	D5	D4	D3	D2	D1	D0	
							●	PARITY ENABLE
						●		PARITY EVEN/ODD
				0	0			SYNC MODES ENABLE
				0	1			1 STOP BIT/CHARACTER
				1	0			1½ STOP BITS/CHARACTER
				1	1			2 STOP BITS/CHARACTER
		0	0					8 BITS SYNC CHARACTER
		0	1					16 BIT SYNC CHARACTER
		1	0					SDLC MODE (01111110 SYNC FLAG)
		1	1					EXTERNAL SYNC MODE
0	0							X1 CLOCK MODE
0	1							X16 CLOCK MODE
1	0							X32 CLOCK MODE
1	1							X64 CLOCK MODE

WRITE REGISTER 5

D7	D6	D5	D4	D3	D2	D1	D0	
							●	Tx CRC ENABLE
						●		RTS
					●			$\overline{\text{SDLC}}$/CRC-16
				●				Tx ENABLE
			●					SEND BREAK
	0	0						Tx 5 BITS (OR LESS)/CHARACTER
	0	1						Tx 7 BITS/CHARACTER
	1	0						Tx 6 BITS/CHARACTER
	1	1						Tx 8 BITS/CHARACTER
●								DTR

WRITE REGISTER 6

D7	D6	D5	D4	D3	D2	D1	D0
SYNC BIT 7	SYNC BIT 6	SYNC BIT 5	SYNC BIT 4	SYNC BIT 3	SYNC BIT 2	SYNC BIT 1	SYNC BIT 0

*ALSO SDLC ADDRESS FIELD

WRITE REGISTER 7

D7	D6	D5	D4	D3	D2	D1	D0
SYNC BIT 15	SYNC BIT 14	SYNC BIT 13	SYNC BIT 12	SYNC BIT 11	SYNC BIT 10	SYNC BIT 9	SYNC BIT 8

*FOR SDLC IT MUST BE PROGRAMME TO "01111110" FOR FLAG RECOGNIT

EXMON2

A NEW MONITOR FOR THE SORCERER

At last, here's a revised version of Exidy's operating system that has full terminal functions. EXMON2 will increase the flexibility of your Sorcerer computer and open the door to many excellent software applications that require special terminal features.

These New EXMON2 routines can be used directly from your Sorcerer keyboard and are easy to use in BASIC programs:

- **Direct cursor positioning**
- **Clear to end of line**
- **Clear to end of screen**
- **Delete line**
- **Text highlight on/off**
- **Reverse ASCII characters**

EXMON2 has many additional features that give your Exidy Sorcerer even greater flexibility:

- **Define and *reserve* top of screen**
- **Search memory for hex or ASCII string**
- **Parallel *and* serial printer drivers *with* or *without* line feeds**
- **A *properly* working serial port (for modems, etc.)**
- **A *f-a-s-t* keyboard status routine that's CP/M compatible**
- **And there's *more* . . .**

Each EXMON2 set comes complete with two burnt-in, fully-tested Eproms and an accompanying user's manual with easy-to-understand installation instructions.

1 set (2 Eproms & manual) . . .		**$65.00** U.S. + postage*
Group discounts	(5 to 9 sets) . . .	**$55.55** U.S. + postage*
	(10 or more) . . .	**$52.00** U.S. + postage*

***Add $5.00** U.S. with the first set, **$3.50** U.S. with **each additional** set for postage and handling.

Please make cheques payable to:
H.A. LAUTENBACH, and mail to: **P.O. BOX 1173, STN. B, DOWNSVIEW, ONTARIO, CANADA M3H 5V6**

TO ORDER EXMON2:

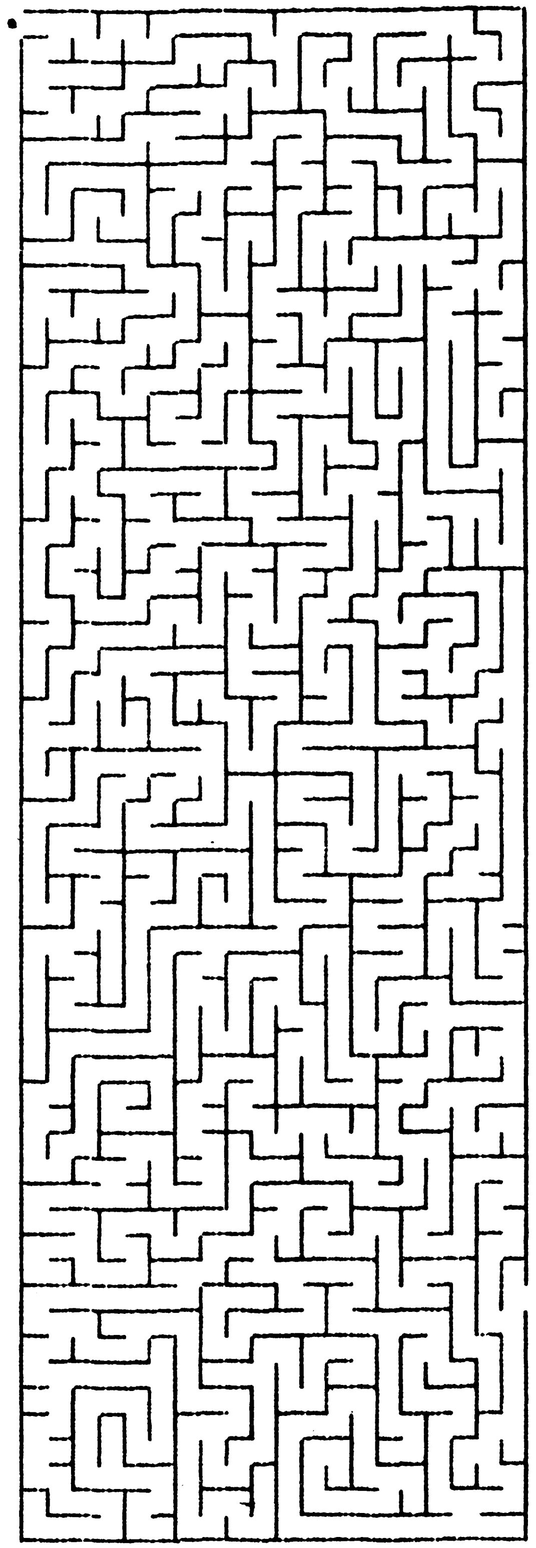
_____ Please send more information on EXMON2.
I have enclosed my certified cheque/money order for $_______; please send me ____ EXMON2 set(s).

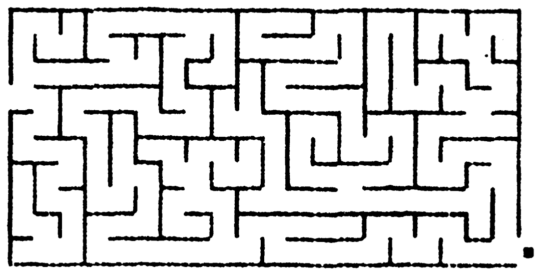
Company/Club____________________

Name____________________

Address____________________

____





Membership Application Form Covering Jan. to Dec 1982

Membership to the group is not restricted to the TORONTO area. All persons willing to participate are invited to join.

As a member of the Sorcerer Users' Group (Toronto), I enclose the annual membership fee and agree to the following Terms.

1. That I will not, without the authorization of the board of directors, represent myself or take any action as agent, or representative or become spokes-person of the group.

2. That I will not use any software obtained from the SUGT library for any commercial purpose or financial gain. The library shall be available to me should I wish to obtain programs donated by other members. These programs shall not be distributed without the owners consent and/or the consent of the board of excecu-tive officers.

3. That I have the right to vote for the officers and directors of the organization at the annual general meeting.

4. That any breach of the above conditions and any other restrictions that the Officers of the Club may invoke in the future on my part may result in suspension or termination of my membership without refund.

Annual Membership Rates: (Jan - Dec)

Canadian - $15.00 Cdn - PLUS $6.00 Postage
U.S.& Foreign $15.00 (U.S Funds) PLUS $10.00 Postage

Payable to - SORCERER USERS' GROUP (TORONTO) - by Cheque or Money Orders.

The SUGT program library is available to all members in the following manner.

You may send $6.00 + $1.50 postage for each volume as they become available and we shall supply the cassette/s. Program cassettes shall be sent via Air Mail.
All issues of PORT FE shall be mailed first class, in the case of non local issues, they are mailed via Air Mail. Past issues of PORT FE are only available for the current calendar year. Contact the editor, he will advise the amount of payment for previous issues.

NAME(print):..
ADDRESS:..
CITY:..
POSTAL CODE:......................
TELEPHONE: Res.................. Bus.......................

Payments enclosed (membership):.............. Library tape/s........Vol 1 or 2

Signature:......................................

Please list the type of equipment you are using etc...
Sorcerer size: 8... 16... 32... 48... other..... S100... Graph board.....
Disk system - Micropolis..... Discus.... Exidy.... other... Size.........
Other Equipment ....
....

If you belong to any other Sorcerer Users' Group please list it below.
....