```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                       *
*                          A N N O U N C I N G                          *
*                                                                       *
*                 C O M M U N I C A T I O N S   S O F T W A R E         *
*                                                                       *
*                          C O M P E T I T I O N                        *
*                                                                       *
*                           First Prize:   $15Ø                         *
*                           Second Prize:  $5Ø                          *
*                           Third Prize:   $2Ø                          *
*                                                                       *
*  THE WINNING PROGRAM WILL:                                            *
*                                                                       *
* :Allow non-disk users to communicate with at least the SCUA CBBS.     *
* :Support file up-loading from, and down-loading to, available RAM     *
*   using the Ward Christensen Protocol. See Protocol Docs below.       *
* :May be adaptable to RTTY communication.                              *
* :Be as short as possible, for maximum Buffer space.                   *
* :Include user documentation and fully commented Source Code.          *
*                                                                       *
* Potential entrants are reminded that STERM and the code is public     *
* domain and is available for downloading on SCUA C.B.B.S. However      *
* this competition is not intended to merely give prize money for       *
* making STERM cassette based. The judges hope for significant orig-    *
* inality. It is expected that two broad philosophies will emerge:      *
* Sorcerer specific using many monitor routines and thus shorter        *
* than another relatively Sorcerer-independent program.  Both           *
* approaches are worthy of consideration by entrants.                   *
*                                                                       *
*  CONDITIONS:                                                          *
*                                                                       *
* 1. The winning entry shall become Public Domain.                      *
* 2. No member of the judging panel shall be permitted to enter.        *
* 3. The current SCUA committee (excluding any entrants) will be the    *
*      final judge of the winning entries and of prize distribution.    *
* 4. The committee reserves the right not to award any prizes, if       *
*      the conditions of the competition are not met, or the standard   *
*      of entries is insufficient to warrant awards for merit.          *
* 5. First prize will only be awarded to a financial member of SCUA     *
* 6. The closing date for entries is 3Øth June 1984.                    *
* 7. Entries should be posted to:                                       *
*    SCUA COMMS COMP, G.P.O. Box 24Ø2, MELBOURNE, 3ØØ1. Australia.      *
*                                                                       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

MODEM/XMODEM Protocol Explained by Kelly Smith,

[Ed: Originally from CP/M-Net "SYSOP", January 8,198Ø and downloaded from SCUA CBBS as PROTOCOL.DOC for publication in SCUA Newsletter.]

I thought that it may be of some interest to those of you that use the MODEM/XMODEM file transfer capability of Remote CP/M Systems, to get a little insight as to the communications protocol (i.e. "handshaking method") used by these systems.

Herein lie the details of a very good (not perfect) data communicat-

ions protocol that has become the "de facto" standard for various remote CP/M systems (RCPM's) that are accessible across the USA (and now Australia). I also wish to give credit to Ward Christensen for writing MODEM.ASM (CPMUG Volume 25.11) and Keith Petersen, Bruce Ratoff, Dave Hardy, Rod Hart, Tom "C", and others, for enhancements to Ward's original program that we now call XMODEM (external modem).

Data is sent in 128 byte blocks with sequentially numbered blocks, and appended by a single checksum at the end of each block. As the receiving computer acquires the incoming data, it performs it's own checksum and upon each completion of a block, it compares it's checksum result with that of the sending computers. If the receiving computer matches the checksum of the sending computer, it transmits an ACK (ASCII code protocol character for ACKNOWLEDGE (Ø4 Hex, Control-F)) back to the sending computer. The ACK therefore means "alls well on this end, send some more...". Notice in the following example, that the sending computer will transmit an "initial NAK" (ASCII protocol character for NEGATIVE ACKNOWLEDGE (15 Hex, Control-U))...or, "that wasn't quite right, please send again". Due to the asynchronous nature of the initial "hook-up" between the two computers, the receiving computer will "time-out" looking for data, and send the NAK as the "cue" for the sending computer to begin transmission. The sending computer knows that the receiving computer will "time-out", and uses this fact to "get in sync"...The sending computer responds to the "initial NAK" with a SOH (ASCII code protocol character for START OF HEADING (Ø1 Hex, Control-A)), sends the first block number, sends the 2' complement of the block number (VERY important, I will discuss this later...), sends 128 bytes of 8 bit data (thats why we can transfer ".COM" files), and finally a checksum, where the checksum is calculated by summing the SOH, the block number, the block number 2's complement, and the 128 bytes of data.

Receiving Computer:

```
----/NAK/-------------------------/ACK/----------------------
     15H                           Ø6H
```

Sending Computer:

```
--------/SOH/BLK#/BLK#/DATA/CSUM/---/SOH/BLK#/BLK#/DATA/etc.
         Ø1H ØØ1H ØFEH 8bit 8bit     Ø1H ØØ2H ØFDH 8bit ....
```

This process continues, with the next 128 bytes, IF the block was ACK'ed by the receiving computer, and then the next sequential block number and it's 2's complement, etc.

But what happens if the block is NAK'ed?...easy, the sending computer just re-sends the previous block. Now the hard part...what if the sending computer transmits a block, the receiving computer gets it and sends an ACK, but the sender does not see it?...The sending computer thinks that it has failed and after 1Ø seconds re-transmits the block...ARGH!...the receiving computer has "stashed" the data in memory or on disk (data is written to disk after receiving 16 blocks), the receiving computer is now 1 block AHEAD of the transmiting computer! Here comes the operation of the block numbers...The receiver detects that this is the last block (all over again), and transmits back an ACK, throws away the block, and (effectively) "catches up"...clever! Whats more, the integrity of the block number is

verified by the receiving computer, because it "sums" the SOH (Ø1 Hex) with the block number plus the 2's complement of the block number), and the result MUST BE zero for a proper transfer (e.g. Ø1+Ø1+FE hex = ØØ, on the first block). The sequence of events then, looks like this:

Receiving Computer:

```
----/ACK/----------------------/NAK/----------------------
     Ø6H                          16H
```

Sending Computer:

```
CSUM/---SOH/BLK#/BLK#/DATA/CSUM/---/SOH/BLK#/BLK#/DATA/etc.
8bit    Ø1H ØØ3H ØFCH 8bit 8bit     Ø1H ØØ3H ØFCH 8bit ....
```

Normal completion of data transfers will then conclude with an EOT (ASCII protocol END OF TRANSMISSION, Ø4 Hex, Control-d) fromthe sending computer, and a final ACK from the receiving computer. Unfortunately, if thew receiving computer misses the EOT, it will continue to wait for the next block (sending NAK's every 1Ø seconds, up to 1Ø times) and eventually "time out". Yhis is rarely the case however, and although not "bullet proof", it is a very workable protocol.

Receiving Computer:

```
----/ACK/---/ACK/"Transfer Complete'/A>(or B>)
     Ø6H     Ø6H ..............................
```

Sending Computer:

```
CSUM/---/EOT/---/A>(or B>)
8bit     Ø4H ............
```

In some case, where the telephone transmission is repeatedly "trashed" (weak signals, multiple noise "hits", etc.) the receiving computer (and operator) will be provided the option to quit. Here, the operator enters "R" or "Q" in response to "Retry or Quit?" (after 1Ø retries), and if quit is evoked by the operator, a CAN (ASCII code protocol CANCEL. 18 Hex, Control-X) is sent by the receiving computer to cancel the entire transfer session (Note: it is possible to "garble" an ACK to a CAN, and abort prematurely):

Receiving Computer:

```
----/NAK/...NAK's ten times.../"Retry or Quit?"(Q)/CAN/A>...
     15H                                            18H
```

Sending Computer:

```
CSUM/---/...Garbled Data...../----------------------/A>...
8bit
```

A final considerations when using the MODEM program, is a timing related problems when transfer status messages and/or textual data is directed to the screen of a slow (48ØØ Baud or less) terminal or to a hard copy printer. This problem is readily apparent (multiple NAK's) when using MODEM for the first time, and can usually be "cured" by NOT

PARALLEL DATA TRANSFER - SORCERER/PIO/MICROBEE.

Frank Herne.

[Ed: This is the only article I have seen interfacing a Sorcerer to a PIO bearing computer. Hence its value as at the very least as a starting point for other tinkerers. It is transcribed from hard copy, so beware any mistakes. Please write in with any ERRATA. Soon to come, an article about reading Bee tapes.]

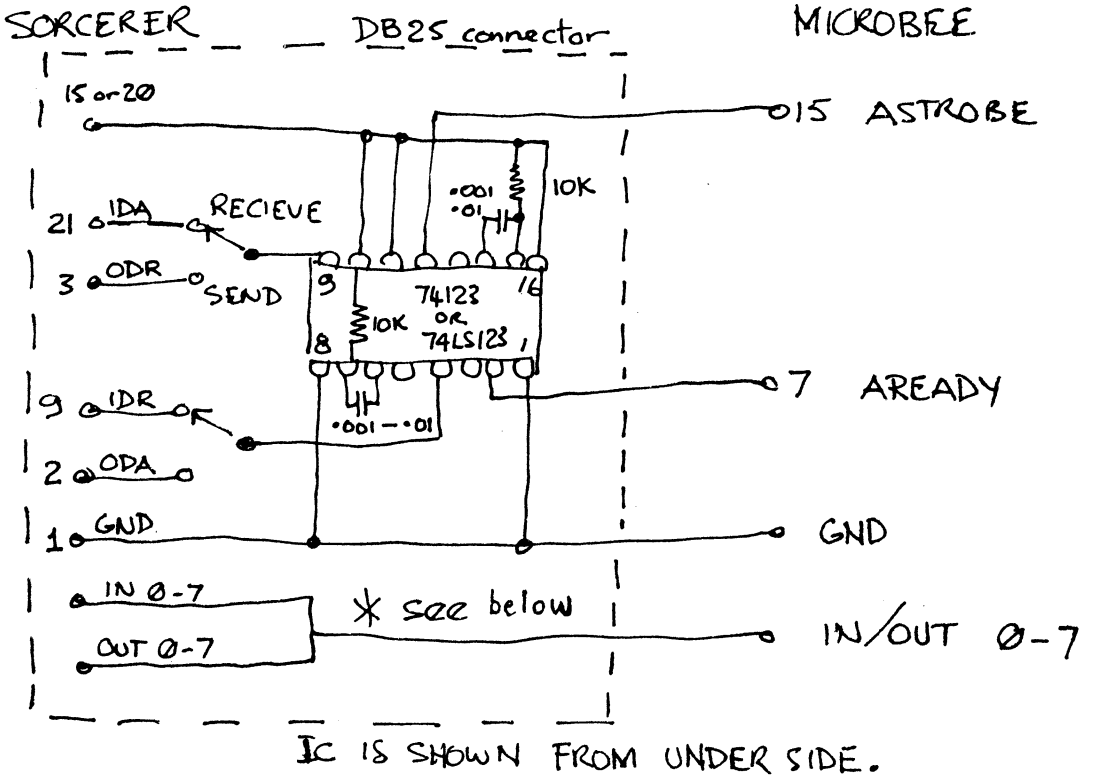
This first bit is by way of a preamble and a sort of summary for those members into parallel data transfer. Yearbook 3 indicates that some users are not getting their parallel ports to work in 8-bit mode. There are two methods that I know work:

1. That by Ken Grimes (N/L No.41 Oct 82) for those who fear the removal of screws from the Sorcerer and,

2. My own internal mod to link pin 11 of the 74LS374 to pin 14 of the parallel port connector. (N/L No.54 Dec 83).

[Ed: 3. That by Ivan Reid also in N/L No.54]
4. Sorcerer to Sorcerer data transfer using the drivers also in this issue.]

The following is a method for data transfer between a Sorcerer and a Microbee. The problem to be solved is that of bidirectional data transfer along a common buss. It does not need either methods 1 or 2 above.



* Link input lines to output lines inside the Sorcerer DB25 connector.

The mod: Disconnect pin 1 of 74LS374 (8H/1, Mk1; 9H/1, Mk2) from ground and join it to pin 8 of 74LS74 (8F/8, Mk1; 10E/8, Mk2). Now the output buffer (Octal D-type Flip Flops) will only be enabled when data is sent (OUTPUT DATA AVAILABLE goes low) and is reset when Acknowledge

(OUTPUT DATA ACCEPTED) goes high. The reason is so that the input and output bits can be sent along a common buss. And this is needed because a Z8Ø PIO (inside the MicroBee) uses bidirectional data lines. There are no drawbacks as far as I can see.

The above circuit diagram includes a switch and 74LS123 which can be mounted inside the DB25 connector. The switch selects the handshake lines for the Sorcerer depending on whether it is sending or receiving. The MicroBee does software switching.

Sending Sorcerer Basic Programs to MicroBee.

Sorcerer: Add this line to your program - Ø NULL 1Ø :LIST :NULLØ :END
Set the switch to 'send'.
BYE to Monitor.
>SE S=5<CR> <-- Delay is to allow the Bee to process the
>SE O=P<CR> the lines. Isn't the Sorcerer wonderful!
Bee: IN£1 ON<CR>
Sorcerer: PP<CR> <-- Will not show on screen now.
RUN<CR>
Watch the Bee Screen for errors which will occur in lines longer than 64 characters.

Sending MicroBee Basic Programs to the Sorcerer.

Bee: Add this line to your program.
5Ø OUT£1 ON :PRINT"PP";CHR$(13) :LIST :OUT£Ø :END
Set the switch to 'receive'.
Note the Bee sends a linefeed at the end of each line which must be filtered out as per the MicroBee Engineers Notebook or in the Sorcerer as below.

Sorcerer: BYE<CR>
>EN FEØØ<CR>
FEØØ: CD 1E FE ØA 28 F9 C9/<CR>
>SE I=FEØØ<CR>
Bee: RUN<CR>

Sending the MicroBee Pak to the Sorcerer.

Sorcerer: Enter this code in a suitable location, e.g., 1ØØH. The following routine loads the Bee Pak into a Sorcerer RAM Pac at CØØØ.

```
21 ØØ CØ              LD      HL,CØØØ ;Point to buffer
CD 1E EØ      LOOP1   CALL    EØ1E    ;Parallel in
FE C3                 CP      C3      ;Wait for first byte
2Ø F9                 JR      NZ,LOOP1
77            LOOP2   LD      (HL),A  ;Store it
23                    INC     HL      ;Point next
7C                    LD      A,H
FE EØ                 CP      EØ      ;Is it end?
C8                    RET     Z       ;Yes!
CD 1E EØ              CALL    EØ1E    ;No! Get a byte
18 F5                 JR      LOOP2   ;Repeat
```

Bee: (Written in Basic 'cause I dunno how a PIO works. It still takes less than 3Ø seconds.)

```
1ØØ OUT£1 :REM SET PIO
11Ø FOR J=Ø TO 1Ø :REM SEND SOME NULLS
```

```
        12Ø OUT Ø,Ø :NEXT J
        13Ø FOR A = 49152 TO 57343
        14Ø OUT Ø,PEEK(A) :REM SEND BYTE
        15Ø NEXT A :OUT£Ø :END
Sorcerer: >GO 1ØØ<CR>
Bee:      >RUN<CR>
```

When the prompt returns the job should be done.

SORCERER TO SORCERER PARALLEL DATA TRANSFER.

David Woodberry.
11 Morris Road, UPWEY, Vic., 3158 (Ø3) 754-651Ø.

The following was developed for use in connection with a program (DISK2.COM) to enable transfer of data between two Sorcerers with dissimilar disk systems connected via the parallel port. The program transfers data at a speed comparable with that achieved with PIP on a single machine.

Note that the operation of these routines does not require any hardware modification to the sorcerer.

Documentation on cable for transfer programs

The parallel port is used on both the send and receive computers and these are connected by a 2Ø wire cable as follows:

One end	Other end
1	8
25-2	4
3	9
4	-25-2
5	13
6	24
7	12
8	1
9	3
1Ø	16
11	18
12	7
13	5
16	1Ø
17	22
18	11
19	23
22	17
23	19
24	6

NOTE: Pins 14,15,2Ø,21 are not used at either end.
Pins 25 and 2 are commoned at both ends and connected with a single wire to pin 4 of the other.
The connections are 'mirror-image' and may be read either way.

This routine is the parallel driver for the sending side:-

```
SENDIT: PUSH    AF
```

```
SNDT1:  IN      A,(ØFEH)        ;check to see if receiver ready
        BIT     6,A
        JR      Z,SNDT1-$
        POP     AF
        OUT     (ØFFH),A        ;send it.
        RET
```

This routine is the parallel driver for the receiving side:-

```
GETIT:  IN      A,(ØFEH)        ;check to see if data available
        BIT     7,A
        JR      Z,GETIT-$
        IN      A,(ØFFH)        ;get it
        PUSH    AF
        XOR     A               ;this bit is the data accepted
        OUT     (ØFFH),A        ;signal from the receiver to
        LD      A,Ø8ØH          ;the sender
        OUT     (ØFFH),A
        POP     AF
        RET
```

The sending side routine is capable of sending 8 bits in parallel (contents of A register) to the receiving side (contents of A register). Because of the nature of the connection it is also possible to send 7 bits of data in the reverse direction. This can typically be used for control information and would need additions to the simple drivers shown here. Note further that since the connection between the two machines is symmetrical, either can act as sender and receiver.

2-3 MHZ MODIFICATION - ANOTHER ONE.

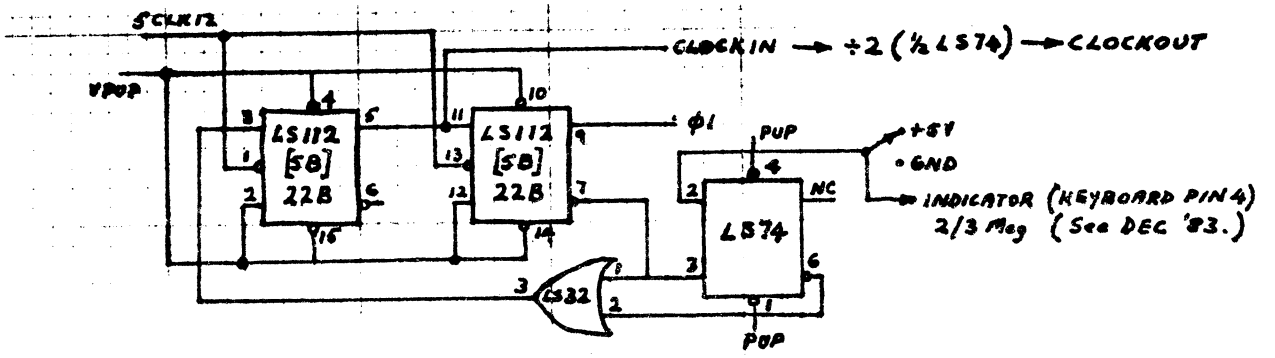
Rick Millicer.
199 Lower Heidelberg Road, IVANHOE, Vic., 3Ø79. (Ø3) 497-3435.

I am conscious that I am featuring a lot in this issue, but there has been a demand for this article - pat myself on the back! The instructions may seem daunting, but are purposely detailed for step by step construction. Have fun. This 2-3Meg mod. has several benefits. First, the CPU never needs to be RESET when changing speeds, because the change is clocked with a D-type flip flop. Second, the UART (cassette and RS232) runs at normal speed irrespective of the CPU speed. Third, only two tracks need to be cut and the less one cuts the better, I guess. And fourth, it applies to all Mk 2 boards (B, C2 and C4) and to the Mk 1 with only substitution of chip numbers. In what follows chip numbers in square brackets [] refer to Mark 1.

The 2-->3 MHz modification involves breaking the feedback loop between pins 7 and 3 of the 5CLK12 divide-by-three divider (LS112 - 22B or [5B]) and interposing a LS32 OR gate. Clocking the OR gate with a D type flip-flop changes the division to divide by 3 or 2 coincidental with the clock pulses, so that the CPU is not upset.

However, not all of the Sorcerer likes 3 MHz, in particular the UART, which needs a fixed 2 MHz CLOCKOUT signal for dividing down to provide standard baud rates for cassette and RS232. So in addition to the above changes to the 5CLK12 divider, this modification duplicates the 'old' divider network and uses this solely for use by the UART. The old divider is composed of an LS112 (5CLK12 --> CLOCKIN) and half an

LS74 (CLOCKIN --> CLOCKOUT). Thus the UART runs at 2 Meg and the rest of the Sorcerer at either 2 or 3 Meg. This suits all known versions of the Sorcerer, including the C4 modification.



2-3 Meg CIRCUIT (÷3 or ÷2 for everything except UART)

Parts list: 1 X 74LS112 (Dual J-K flip flop))Total cost
1 X 74LS74 (Dual D-type flip flop))approx. $2.50
1 X 74LS32 (Quad OR gate))
1 X SPDT switch)

1. On the underside of the board, cut the trace joining 22B/7 and 22/3, [5B/7 and 5B/3]. This was the divider feedback loop.

2. Next find CLOCKOUT to the UART and cut it. The fixed 2 MHz CLOCKOUT will be later connected beyond this cut.

For Mk2 only. On the topside, cut the trace running BETWEEN pins 4 and 5 of 16E. Note the word 'between' means that the trace disappears under the chip, runs between pins 4 and 5 and is not connected to either pin 4 or 5. This trace is CLOCKOUT going under the chip to pin 11 of 16E. Check it with a multimeter first! (NOTE also that the Mark 2 circuit diagrams incorrectly omit this inverter (16E/11,10) from the CLOCKOUT input to 13H and 14H.)

[For Mk 1 only]. On the topside, cut the trace running BETWEEN pins 9 and 10 of 11H. Note the word 'between' means that the trace disappears under the chip, runs between pins 9 and 10 but is not connected to either pin 9 or 10. This trace is CLOCKOUT going to pin 6 of J11.

3. Bend out pins 3,5,6,7,9,11 of the new LS112. The remaining pins 1,2,4,8,10,12,13,14,15,16 are left down. Cut pins 6 & 9 flush with the chip; they are not used. Jumper pin 3 to 7 and pin 5 to 11. I find it easier to do the jumpering before piggybacking. Solder the remaining pins to chip 22B [5B], so that you finish up with two LS112's one on top of the other playing leapfrog with legs 1,2,4,8,10,12,13,14,15,16 joined in fond embrace.

4. Bend out pins 2,3,5,6,8,9,11,12,13 of the new LS74. The remaining pins 1,4,7,10,14 are left down. Jumper pin 13 to 14.

5. Bend out all the pins EXCEPT 7 and 14 of the new LS32. Piggyback the LS32 on top of the LS74 with pins 7 and 14 soldered together.

6. Join the following with small lengths of fine wire: LS32/2 to LS74/6, LS32/1 to LS74/3, LS74/8 to LS74/12.

7. Piggyback this married pair of chips on top of 22C [1B] and solder the unbent pins of LS74 to 22C [1B], i.e., 1,4,7,10,14.

8. Jumper 22B/7 [5B/7] to LS32/1, 22B/3 [5B/3] to LS32/3, new LS112/11 to LS74/11.

9. Connect the fixed 2 MHz CLOCKOUT to UART circuitry:
   Jumper LS74/9 to 16E/11 - Mark 2.
   Jumper LS74/9 to J11/6 - Mark 1.

10. Mount the switch in a convenient place and solder the wire from LS74/2 to the centre contact. The other two switch contacts are to any convenient +5 volt line and ground.

I have not had to junk any Z80 nor RAM chips (yes, even 350 nS) in the dozen machines that this mod has been done to. RAM timing in the Mark 1 and Mark 2B boards has occasionally needed adjustment though. See N/L Oct-Nov 1983 for RAM discussion.

Finally, a caveat. At 3MHz the 12 volt regulator will 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 power supply, the 12 volt regulator gets very hot due to the proximity of the resistor (radiator!!), and can and does 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.

CHECK FOR Z8Ø cf. 8Ø8Ø.

Phil Charley.
P.O. Box 164, WICKHAM, WA., 672Ø

Have those CP/M programmers among you, who use the relative jumps etc available for Z8Ø use, but not 8Ø8Ø, ever considered the potential plight of an unsuspecting CP/M user with an 8Ø8Ø system when one of your super creations reaches him through the friend-to-friend public program transfer network? If you have a conscience, consider using this routine to provide a warning if your program may have to run on an 8Ø8Ø system.

```
Z8Ø?:   LD    A,Ø7FH      ;Ø1111111 into accumulator
        INC   A           ;make it overflow ie. 1ØØØØØØØ
        JP    PO,NOTZ8Ø   ;only 8Ø8Ø resets for odd parity here
Z8Ø:    etc               ;continue, as Z8Ø detects overflow
                          ;and sets "parity/overflow" flag
;
NOTZ8Ø: LD    DE,Z8ØMSG   ;output "Needs Z8Ø" message,
        etc               ;and return to CP/M
```

This routine relies on the Z8Ø's multiple uses of the parity flag. For arithmetic operations it becomes an overflow flag, whereas in most other situations it reflects the parity. On the other hand, the 8Ø8Ø always uses it for parity.

So far, I have had time to work through the first third of the book. I have found only one program which will not work on the SORCERER. This program is intended to "ring the console bell". The SORCERER does not have a built-in sound generator which responds to an ASCII 7, so naturally this program will not perform properly.

If you are a beginner who wants to make CP/M "succumb to your wicked will", buy THE SOUL OF CP/M. My copy cost me about $33, which might seem high. I think I got excellent value for my money. If you want to get a good overview of what the book covers, read the author's INTRODUCTION in the bookshop before you purchase.

If anyone reading this review is considering running a course on the inner workings of CP/M, I suggest that you have a look at THE SOUL OF CP/M. It could be the ideal textbook for your students. But on the other hand, that may not be a good suggestion. You might find that most of your students abandon your class because they find SOUL is a better teacher than you are!

LATEST CASSETTE NOTE.

Rick Millicer.
199 Lower Heidelberg Road, IVANHOE, Vic. 3079 (03) 497-3435

Poor cassette loading, after ALL known previously published and unpublished mods (EXIDY, SCUA, ESC, Dick Smith unpublished but observed, etc) were tried, was fixed by changing R12 (Mk 1), R40 (Mk2), from 220k (or 200k) to 47k. The problem was finally tracked to a very uneven mark/space ratio of the recovered square wave on the output of the third op amp (LM324 pin 14). This was after changing all the active analogue elements in the interface and some filter fiddling.

This reduction reduced the window of the hysteresis loop for the switching of the op amp comparator - LM232 pins 12,13,14. Reducing its value gave a more accurate 1:1 or 1:2 mark/space ratio of the recovered 'square' waves using a simulated Manchester encoder as input. Took 4 tedious nights to fix one cassette interface....

Don't try it unless you have access to a 'scope - in some cases it worsens accuracy of the mark/space ratio.

Also, two Tandy Data Cassette Recorders I have had dealings with appear to be factory set with too much HF pre-emphasis to work well with Sorcerer interface. It is presumably needed for Tandy tape recording format, but that's speculation.

PROPOSED ADDITIONS TO THE Z80 INSTRUCTION SET.

Anon.

Courtesy John Green.

BBW	Branch Both Ways.	EIC	Execute Invalid Code.
BEW	Branch Either Way.	EPI	Execute Programmer Immediately
BH	Branch and Hang.	EROS	Erase Read Only Storage.
BMR	Branch Multiple Registers.	IBP	Insert Bug and Proceed.
BOB	Branch On Bug.	IP	Ignore and Proceed.
BOI	Byte Operator Immediately.	MLR	Move and Lose Record.
BPO	Branch on Power Off.	RPM	Read Programs Mind.
BST	Backspace and Stretch Tape.	RPP	Rip Printer Paper.
CLBR	Clobber Register.	SRSD	Seek Record and Scar Disk.
CM	Circulate Memory.	SRZ	Subtract and Reset to Zero.
CRN	Convert to Roman Numerals.	SROM	Store in Read Only Memory.
DC	Divide and Conquer.	TDB	Transfer and Drop Bit.
DMP	Destroy Memory Protect.	UER	Update and Erase Record.
DO	Divide and Overflow.		

[Proposed additions? I thought they were already there. Ed]

The following 'shortform' hard copy Library Catalogue was kindly sent to SCUA by Chris Halliday, President of Sorcerer Users Group (S.U.G.). It was transcribed to conform to our N/L format. It does not figure in the N/L page numbering and is supplied as an appendix which may be filed separately from the N/L.

A fuller and more descriptive Catalogue, soon to be released, will be on tape only. The boys in Sydney have been busy. Thanks guys.

DIGITRIO, BASIC AND SYSTEM SOFTWARE PROGRAMS.

Phil Moyle.
Current address???

As a user of the Digitrio System, I have been progressively transferring my cassette files onto CP/M disks. Among the files I have wanted to save have been Richard Swannel's (SYSTEM SOFTWARE), 'Touch Type Tutor' programmes, TTY1- and TTY2-.

Richard was kind enough to write an uploader to be implemented after loading the tape file at 200H, but after all this, still no success. [Ed: see also Bob Stafford's contributions in N/L Sept '81, Nov '81. and Steve Maddocks comment in Jan '82.]

After uploading , the TTY programmes also reconfigure the first 100H bytes of memory. Poking around in the 0 to FF memory area, I found a CALL C858H (C3 58 C8) at 1B-1D. As this is a call to the RomPac Basic RUN routine, I knew I had trouble because the Digitrio Controller does not allow the use of a RomPak.

With the Digitrio CP/M disk comes a copy of Standard Basic which relocates itself to 8000H, so I started to work my way through the TTY file looking for C3 58 C8. I found it at 41B-41D and changed the C8 to 88. It works!

Thus the procedure for this exercise is:

1. Boot CP/M.
2. Go to Monitor.
3. LO 1 0200 <-- load the cassette file @ 200H.
4. EN 100: C3 00 50/
5. EN 41D: 88/
6. EN 5000: 21 00 02
   11 00 FE
   01 00 48
   ED B0
   C3 10 00/
7. GO 0
8. SAVE 90 TTY1-.COM

In order to use TTY1-, it is necessary to...

1. Boot CP/M
2. Load BASIC
3. Load TTY1-

[Ed: Thank you, Phil, for the idea and writing about it. Readers may care to note the address of the relocator at 5000H. I cannot see why the relocator is so far up in memory for such a short program. Bears thinking about for preservation of disk space for us poor chaps with only 20Sk

* *

MICROBEE Tape Loader

Brian McHugh
35 Blackwood Terrace, HOLDER, ACT 2611. AH (062) 88 5550

MICROBEE tapes at 300 BAUD are essentially identical to SORCERER tapes. The physical encoding is the same at the byte level and the only significant difference in file format is that there is no 'leader' sequence between the header and the data blocks. The header is the same length, lengths and addresses in the header are the same, blocks are the same length and the 'CRC' checking is identical. The MICROBEE header supports one more file name byte and has speed and execute control bytes corresponding to bytes which are not used by the SORCERER.

The BEELD program listed below will read a 300 BAUD MICROBEE tape and store the header in the monitor work area (MWA) and the data in memory starting at address 'ADR'. The program can be loaded anywhere, but if it is used at F030, it will handle the maximum MICROBEE file for any SORCERER memory size without overwriting itself and will be compatible with the Development PAC.

The program does not attempt to load the file at the address in the header as this will probably not work in the SORCERER. ADR is set at zero, but, if the file is to be loaded somewhere else, the data can be moved after loading or the address at 'SET' can be changed before loading. Note that MOve may not work if there is an overlap.

After reading the header the program displays 'LOADING -' and finally writes the

standard header display on the screen. Only the first 5 characters of the name will be displayed. All CRC bytes are checked and you may get 'CRC ERROR'. It returns to the Monitor when loading is complete. If nothing seems to be happening, the <ESC> key etc. will get you out of the program.

The BEELD program uses a lot of EXIDY MONITOR 1.0 routines but it shouldn't be too hard to convert it to another monitor. It has actually been derived from a much longer program which is more independent of the monitor.

Having read the data in, what then? Some machine code programs may be run after a few minor changes to monitor calls and port references, but most would need a lot of work. BASIC programs would need token adjustment and there would still be trouble with PEEKs and POKEs and other incompatibilities. ASCII text files should be OK so the program may be useful for word processor files and machine code programs with simple console I/O and no special graphics. Assembler source files may also be useable.

So, the procedure is:
1. Get the program as listed into memory starting at F030.
2. Save it on tape --> SE X=F030
   --> SA BEELD F030 F080

To use:
1. LOG<CR>, or LO<CR> and GO F080<CR>
2. No indication is given of the end of loading BEELD, if LOG is used.
3. Insert a uB tape, start the recorder and the program will load.
4. The monitor prompt appears at the end of a successful uB tape load.

```
;------ MICROBEE TAPE LOADER ------
; EXIDY MONITOR 1.0 ADDRESSES ETC.
;
WARM    EQU     0E003H
GETIY   EQU     0E1A2H
MOTRON  EQU     0E28AH
MTROFF  EQU     0E2AFH
GETHED  EQU     0E71BH
BLKADJ  EQU     0E6A9H
TAPEIN  EQU     0E2DAH
FINISH  EQU     0E1D4H
CKCRC   EQU     0E74EH
LDGMSG  EQU     0E4BFH
FILHD   EQU     0E453H
MSGOUT  EQU     0E1BAH
HEDPRT  EQU     0E6DEH
TAPES   EQU     3DH
THEAD   EQU     57H
HSIZE   EQU     7
; ADDRESSES
ADR     EQU     0       ;LOAD ADDRESS
        ORG     0F030H  ;PROGRAM
; START
BEELD   CALL    GETIY   ;MWA
        LD      SP,IY   ;STACK
        LD      (IY+TAPES),0 ;300 BAUD
        LD      B,1     ;UNIT 1
        CALL    MOTRON  ;START TAPE
        CALL    GETHED  ;LEADER + HEADER
; BEE HAS NO DATA LEADER
        LD      HL,LDGMSG ;REPORT
        CALL    MSGOUT
;
SET     LD      HL,ADR  ;LOAD ADDRESS
        LD      E,(IY+THEAD+HSIZE) ;£BYTES
        LD      D,(IY+THEAD+HSIZE+1)
;
LDBL    CALL    BLKADJ  ;SET FOR BLOCK
        JR      Z,LDBLX-$$ ;END?
;
LDBLA   CALL    TAPEIN  ;BYTE
        JP      Z,FINISH ;ESC?
        LD      (HL),A  ;STORE IN BUFFER
        INC     HL      ;NEXT BUFFER
        DJNZ    LDBLA-$ ;END BLOCK?
        CALL    CKCRC   ;CHECK CRC
        JR      LDBL-$  ;NEXT BLOCK
;
LDBLX   CALL    MTROFF  ;TAPE OFF
; LOADED - PRINT HEADER
```

```
        LD      HL,FILHD
        CALL    MSGOUT
        CALL    HEDPRT
        JP      WARM    ;BACK TO MONITOR

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

F030:  CD A2 E1 FD F9 FD 36 3D 00 06 01 CD 8A E2 CD 1B
F040:  E7 21 BF E4 CD BA E1 21 00 00 FD 5E 5E FD 56 5F
F050:  CD A9 E6 28 0F CD DA E2 CA D4 E1 77 23 10 F6 CD
F060:  4E E7 18 EC CD AF E2 21 53 E4 CD BA E1 CD DE E6
F070:  C3 03 E0 00 00 00 00 00 00 00 00 00 00 00 00 00
```

* *

DIGITRIO DISK CONTROLLER - REVIEW

Frank Schuffelen.
66 Porter Street, TEMPLESTOWE. 3106. (03) 846-2424

Recently I decided to purchase the DIGITRIO disk controller, having previously owned the imported 5.25" VISTA disk drives and controller. Though I had been perfectly happy with the VISTA, I felt somewhat restricted in not having access to 8" drives, for easy access to commercial software.

DIGITRIO will supply the controller only, or the controller and one or two 5.25" or 8" disk drives built into a cabinet. One thus has the option of buying the unit ready built, or to assemble it oneself. I chose to buy my own 8" drives and cabinet. This was because I specifically wanted Slimline Drives and DIGITRIO's built up system only has Standard Drives. Moreover, the 50 way connector from the Sorcerer to the controller is purposely very limited in length, so I elected to mount the controller inside the Sorcerer case. Ribbon cable length from the controller to the drives can be quite long (a metre, in my case) permitting a greater scope for hardware layout.

For those wishing to build up their own 8" Mitsubishi system, as I did, I will attempt to describe what to do and what to look out for. You will need the following:

One or two 8" disk drives (mine are Double Sided Double Density), DIGITRIO controller, a power supply and case for drives, screws for mounting the drives in the case (from drive supplier), terminal blocks, fan & safety guard, fuse and holder, mains lead to suit, toggle switch, ribbon cable and a sheet of thick aluminium to mount the power supply. Optionally, a BULGIN mains filter with 240V socket attached.

Cut out a hole in the back of the drive case for the ribbon cable, bolt a thick sheet of aluminium in the back of the case, and the switch in the front of the case. Wire up the connections to the power supply, fan, switch, etc. Make changes to the links on your drives as per the DIGITRIO manual. For MITSUBISHI slim line drives I had to make the following 6 changes:

1. Solder a small link across two adjoining dots marked Y at D9.
2. Remove the green connector from JFG and put it on JSG at A1.
3. For the B drive, move the connector from 1 to 2 at D1.
4. Remove the terminating resistor network from one of the 2 drives, leaving the other one in place, where your end 50-pin plug goes (usually top drive) at E1. If you are using one drive, leave the terminating resistor network in place.
5. Remove the connector link from X at C4, place it across 2S at C1.
6. Remove the link from Z at D10 and place it across C at D1.

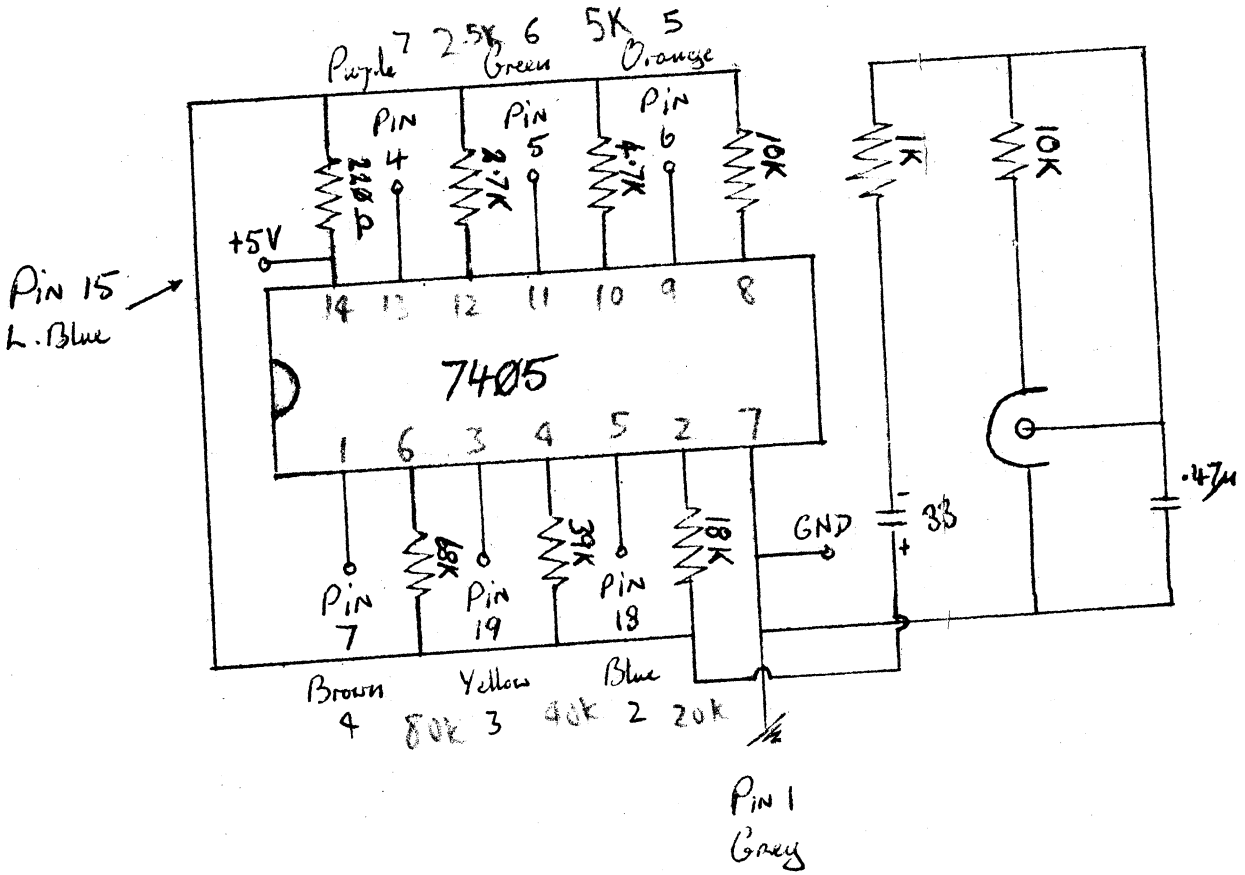
To install the controller inside the SORCERER you will also need to do the following. Note that power is drawn from the Sorcerer supply rails. Cut and bend an aluminium plate to act as a chassis for the controller board. Fit connectors to the ribbon cables. Mount the fan - you may need to remount the transformer, fuse and switch.

I am grateful to the following members for their help: John Cheeseman, who designed the installation of the controller and fan into the SORCERER; Bruce Alexander, who converted the SORCERER to switch between 48K and 54K; Dale Grose, for supplying the cabinet for 2 Mitsubishi slimline drives and their power supply, and much advice.

The price of the DIGITRIO controller also includes a well written technical manual plus a book explaining the use of CP/M. Software supplied includes the following:

Software supplied by Digital Research:

Sorcerer Sound System



Sorcerer Connections To Parallel Port

1.	Gnd	
2.	OD	accepted
3.	OD	available
4.	Out	7
5.	↙	6
6.	↙	5
7.	↙	4
8.	Gnd	
9.	Input Data	available
10.	↙	0
11.	↙	2
12.	↙	4
13.	↙	6
14.	N/C	
15.	5 v	
16.	Out	0
17.	↙	1
18.	↙	2
19.	↙	3
20.	5v	
21	Input Data	accepted
22.	↙	1
23.	↙	3
24.	↙	5
25.	↙	7

PORT FF

SORCERER PORT ASSIGNMENTS.

Jim Graham.
8 Verdun Street, SURREY HILLS. Vic. 3127 (Ø3) 836-38ØØ

PORT ØFCH

	IN	OUT
DB7		DB7
DB6		DB6
DB5		DB5
DB4	SERIAL	DB4 SERIAL
DB3	IN	DB3 OUT
DB2	DATA	DB2 DATA
DB1		DB1
DBØ		DBØ

PORT ØFDH

	IN	OUT
DB7	-	DB7 -
DB6	-	DB6 -
DB5	-	DB5 -
DB4	UART PARITY ERROR	DB4 NO PARITY
DB3	UART FRAMING ERROR	DB3 PARITY. 1 EVEN Ø ODD
DB2	UART OVERRUN	DB2 STOP. 1=2 STOP Ø=1
DB1	Rx DATA AVAILABLE	DB1 BITS PER CHR NB2.
DBØ	Tx BUFFER EMPTY	DBØ BITS PER CHR NB1.

PORT ØFEH

	IN	OUT
DB7	PAR. DATA AVAIL.	DB7 Ø=CASSETTE 1=RS232
DB6	Unused	DB6 BAUD RATE Ø=3ØØ 1=12ØØ
		PARALLEL DATA ACK
DB5	E256 (GETIY)	DB5 MOTOR CONTROL 2
DB4	-)	DB4 MOTOR CONTROL 1
DB3	) KEYBOARD	DB3 -) KEYBOARD
DB2	) IN	DB2) OUTPUT
DB1	) DATA	DB1) BITS
DBØ	-)	DBØ -) Ø-3

PORT ØFFH

	IN	OUT
DB7	-)	DB7 -)
DB6	)	DB6)
DB5	)	DB5)
DB4	)-PARALLEL IN	DB4)-PARALLEL OUT.
DB3	)	DB3)
DB2	)	DB2)
DB1	)	DB1)
DBØ	-)	DBØ -)

****

SHEDDING A BIT OF LIGHT ON DISK FORMATS

Andrew Marland.
35, Avenue Chevreul, Bois-Colombes, France. Tel (1) 784.89.56

[Ed: The following is part 1 a series of articles on disk formats which are too long to publish in one go. The tables will be in the November N/L.]

Ever since I started word processing (1980), I've been trying to get some sense on the subject of disk compatibility from suppliers and other people who-ought-to-know. I felt then, and I still feel, that if only the manufacturers hadn't gone out of their way to make things difficult, floppy disks would be an ideal medium for communicating large amounts of text over space and time (eg. to send my Australian correspondent a machine-readable copy of a patent I'm drafting now, so that he can amend it easily in response to Australian patent office objections in, say, three years' time).

For years I was simply fobbed off with remarks like "can't be done" or "too messy". But a little information did begin to trickle in. Then one or two firms started selling programs to read several formats on this or that specified machine.

Mechanical compatibility:
Drives come in various sizes eg. 8" and 5" which can't read disks for another size. 5" drives come in three different stepper motor pitches: 48, 96, and 100 tracks per inch (tpi). Some drives are switchable between 48 and 96. The number of tracks actually used is under software control and is not a direct function of the drive mechanics, but in general: 35 or 40 tracks means 48 tpi, 80 tracks means 96 tpi and 77 tracks means 100 tpi. Finally, for double sided operation, the drive must have two heads.

Electrical compatibility:
Data is written onto disks in two stages, in a first stage (track formatting) a lot of blank data is written onto the disk, together with "sign-posts" marking the beginnings and ends of sectors. This is true both of hard sectored disks (the ones with lots of little holes round the hub) and of soft sectored disks. In a second stage, the operating systems writes and reads useful data in the places marked in the track formatting stage.

Different disk controller electronics produce (at least) the following mutually incompatible kinds of track format:

1) Soft sectoring on the lines laid down by IBM in about 1970 and generally embodied in the Western Digital family of chips;
2) Soft sectoring in any of the various ways devised for CBM, Sirius, or Apple;
3) Hard sector 10 sectors per track (North Star and Vista are similar enough for software to get round their mutual incompatibilities); and
4) Micropolis 16 sectors per track and either 48 or 100 tpi. The only soft sectored 100 tpi formats I am aware of are used by the Exidy Sorcerer, which started in Micropolis hard sector format and then moved on to soft sector using the same drives, but a different controller card.

A controller card also needs to be able to control selection of one or other head of a double-headed drive and to be able to cope with single density or double density or both. For simplicity, let's assume that there is only one kind of single density recording scheme (FM) and only one kind of double density recording scheme (MFM).

Operating system compatibility:
Each operating system has its own idea of where to store directory information on a disk, and what kind of information it will use to locate a file, and as a side issue, it may encode directory data (eg. DEC RT/11 packs a file name at six bits per letter).

Skew:
The best way to use a disk is to read or write an entire track in one go. This is what the IBM PC does, for instance. However, an entire track may have 6K bytes or more of data, and a smaller buffer may be preferred. The buffer must be at least one host (ie. on-the-disk) sector long. Common sector lengths are 128, 256, 512 and 1024 bytes. In between reading or writing each sector in a sequence of sectors, the computer needs to do various bits of computing, eg. transferring data from an input buffer to where it's needed in RAM or vice versa. This takes finite time, during which the disk moves on. Thus the next sector actually available when the computer is next ready for disk I/O is not the immediately adjacent sector, but rather the next-sector-but-N. This has lead to two different schemes for interleaving sectors round a track, both schemes being called "skewing".

1) The sectors round a track are numbered in sequence, but the cunning operating system asks the disk controller hardware to work on sectors in a funny order, eg. 1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21, 2, 8, 14,

20, 26, 6, 12, 18, 24, 4, 10, 16, and 22, instead of 1, 2, 3, ...

2) The sectors are initially numbered in a related higgledy-piggledy order round a track by the original track formatting operation, eg. in the above example, the sector numbered 2 would be in 7th place, the sector numbered 3 would be in 13th place, and so on. With this second system, skewing is done "once-and-forever" by the formatting program, and thereafter the operating system just asks for the sector number it first thought of.

Note that the principle of giving sectors labels that do not make sense at first glance can be taken to greater lengths: the associated recorded track number need not match the host track, and sector and/or track numbering can start at various arbitrary values. Tracks generally start at 0 for the outside track, but sectors commonly start at 1, with 0 and 81H being alternatives mentioned in the tables below.

CP/M variants:
The strength of CP/M is that many of its format features are table-driven so that each manufacturer can do what it thinks best in disk usage. The weakness is that this gives rise to a lot of not-quite compatible formats. Tables of relevant information follow, but several words of explanation are needed first.

CP/M provides for a "disk parameter block" (DPB) which defines most of the features a designer needs to be able to modify in order to fit CP/M to some specific hardware, eg double sided 48 tpi drives, a double density controller card, and a Winchester hard disk unit. Although floppy disks are the only kind of disk of interest in trying to exchange data, the DPB table must also be capable of coping with hard disks, which means that in giving "likely" ranges of values below, I have limited myself to the more likely parameters for floppy disks only.

SPT is a word defining the number of 128-byte CP/M sectors per track. These CP/M sectors are usually bunched into "host" sectors of 256, 512 or 1024 bytes per sector. The number of bytes per host sector is an essential item of information for your BIOS, but does not appear in the DPB, since CP/M "thinks" in 128 byte sectors at all times.

Likely values of SPT are:
12H for 5" SD 128 bytes/sector (BPS)
14H for 5" SD 256 BPS
20H for 5" DD 256 BPS
22H for 5" DD 256 BPS
24H for 5" DD 256 or 512 BPS
28H for 5" DD 512 or 1024 BPS

1AH for 8" SD 128 BPS
34H for 8" DD 256 BPS
(sorry, no examples of 512 BPS)
40H for 8" DD 1024 BPS

(One of the common double sided schemes "pretends" that the extra sectors on the "other" side run on from the sectors on "this" side, in which case the SPT number is doubled).

It should be possible to write a routine for inspecting an "alien" disk, determining the size and number of the host sectors, and then setting SPT accordingly. Also, if the sector numbering around the track turns out to be skewed, there is a very good chance that no further skew data is required.

BSH and BLM are two single byte values defining the SAME quantity, namely the "block" length. CP/M allocates disk space to files in whole numbers of "blocks" so half a block length is "wasted" by each file on the disk (on average, in the long run, and assuming that files come in random lengths that are long in comparison with the block length). Likely values are as follows:

Block length	BSH	BLM
1K = 8 CP/M sectors:	03	07
2K = 16 sectors :	04	0F
4K = 32 sectors :	05	1F

in other words block length in CP/M sectors is equal to 2 to the power BSH or, alternatively, it is equal to BLM + 1.

EXM is the extent folding mask. After blocks, CP/M "thinks" in "extents" which are chunks 16K bytes long, regardless of block size. EXM indicates the number of 16K byte extents that can be pointed to from a single directory entry. Likely numbers are: 0 means 1 extent, 1 means 2 extents, and 3 means 4 extents; ie. there are EXM + 1 extents possible per directory entry. Warning! there is no requirement to pack all the extents possible into each directory entry; for example, by using an EXM of 0, where 1 would have been possible, 8 bytes in each entry could be "reserved", (or wasted, depending on how you look at it).

DSM is a word defining the number of blocks on a given disk, with the blocks being numbered 0 thru DSM, ie. there are DSM + 1 blocks on the disk. This number includes the blocks used by the directory, but does not include the tracks reserved for the operating system. This number of tracks is given by the word OFF. Thus:

1) (BLM + 1)/8 = block size in K bytes
2) (DSM + 1)(block size) = Storage in K bytes
3) OFF + 8 x (Storage in K bytes)/SPT = tracks

DSM has an effect on EXM: if DSM is a one-byte value, then twice as many blocks can be pointed to in a given 16 byte length of directory than would be possible if DSM were a two-byte value. Another twist is that each directory entry must be capable of pointing to at least one entire extent, which means that 1K byte blocks can only be used with a disk having a DSM of FF or less.

The remaining three entries relate to directory size. The directory always starts at block 0, and a whole number of blocks must be assigned to the directory. Each block assigned to the directory is represented by a corresponding 1 bit in the allocation vectors 0 and 1. Thus likely values of AL0 are 80 (for one directory block) C0 for 2 blocks, E0 for 3 blocks and F0 for 4 blocks. The rest of AL0 and all of AL1 are all zeros on floppy systems.

DRM is the maximum directory entry number (numbering is 0 thru DRM so there are DRM + 1 directory entries), and CKS is the number of CP/M directory sectors to be checked (to make a surreptitiously changed disk R/O) and is thus normally equal to (DRM + 1)/4. Since there are 4 directory entries to one CP/M sector or 32 entries to 1K bytes, the following values are likely for DRM->CKS:

AL0	80	C0	E0	F0
Block size	1 block	2 blocks	3 blocks	4 blocks
1K	1F->08	3F->10	5F->18	7F->20
2K	3F->10	7F->20	BF->30	FF->40
4K	7F->20	FF->40	17F->60	1FF->80

Warnings:
Given a defined block size and AL0, DRM may be less than the value given in the table. Some manufacturers appear to have used this as a way of "reserving" some of the directory space on a disk for a little bit of system that wouldn't squeeze into the available system track space. For example, Kaypro appears to get away with a single system track (OFF = 1) by putting some of the system into space "allocated" to the directory. I call this a "funny". Similary CKS may be smaller than the value expected from DRM, eg. half the expected value, in order to speed up directory checking at the expense of confusion if two disks have identical first directory halves but different second halves, and are then surreptitiously swapped without doing a control-C. One of the many reasons for hard disks being faster than floppies is that it is safe do no directory checking since the medium is fixed.

At first glance that gives an awful lot of possible formats. But the number of likely formats is only "moderately" large, say a few thousand. Look at it from a designer's point of view: for a single sided format, the possible variations in drive diameter (8" or 5") and in sector size (128, 256, 512, or 1024) give rise in a slightly roundabout way to about 8 different possible values for SPT (say 3 bits of data); by convention the number of tracks on a disk is 35, 40, 77 or 80 (two more bits, total = 5); block size is "normally" 2K, but "abnormally" it may be 1K on a small system or 4K on a big system (one more bit only, total = 6); the number of system tracks is nearly always 2 or 3, but 1 and 4 happen often enough to be watched for (2 more bits, total = 8); and the directory may 1 to 4 blocks long (2 more bits, total = 1024). That leaves out double sided formats, skew, and "funnies".

Double sided formats:
CP/M "expects" only one side, so all double sided schemes work by "pretending" that the data on the "other" side of the disk is located in extra sectors or in extra tracks on "this" side of the disk. With extra sectors, the SPT number is double the equivalent single density number, and the BIOS has to sort out whether the extra sectors are from SPT/2 up (the only kind I've seen with CP/M) or every other host sector (seen on other operating systems). With extra tracks several schemes are about: odd tracks on one side even tracks on the other is the most popular; but the first half of the tracks may be on "this" side and the second half may be on the "other" side, in which case the tracks on the "other" side may be numbered from the outside inwardly (just like "this" side) or from the inside outwardly.

These various conventions are hinted at in the DS column of the tables by sec for extra sectors, tks for extra tracks, 35+ for tracks 0 to 34 on "this" side with tracks 35+ on the "other" side counting in the usual direction; 40+ is the same for a 40 track format, and 79- means counting backwards on the "other" side of a 40 track system (ie. CP/M's "pretend" track 41 is actually 79-41 =

38, but on the "other" side). All the above systems assume that the track format of both sides is the same and is conventional, ie. track 0 at the outside to track N near the hub and sector numbering 1 to n on each track. Other track formats count as "funnies". By convention "this" side is side 0 and the "other" side is side 1 - jargon can get very confusing!

Skew:
The skew "factor" is quoted as the number of CP/M 128 byte sectors between the first CP/M sector number in the first host sector and the first CP/M sector number in the second host sector. The skew factor must therefore be an integer multiple of the host sector size in 128 byte units. The skew tables use the convention that CP/M sectors are numbered 1 to SPT around a track.

If you want to do lots of different skews, it is cheaper on memory to define any particular skew with a table of 6 or so bytes and then calculate the skew, rather than solemly listing all the numbers as in the tables, and then looking the skew up. Skew can be calculated using the following arbitrary data: skew factor, host sector size, SPT, flag for double sided scheme, end correction (ie. where to pick up sector numbers on the second circuit round a track), and CP/M number of first sector per track (an Exidy funny).

Funnies: I've given special columns to three other "funnies":
1) the lowest sector number used by the track format: HP and Lobo use 0 instead of 1, while Datavue uses 81H!
2) whether the data needs inverting to make sense (Superbrain seem to use a different data convention from the rest of us);
3) for a double density system: the number of the track on which double density starts. I've put "all" instead of "0" to avoid being misleading. It took me some time to get my BIOS to accept double density track 0, and I'm now trying to twist its arm to cope with double density starting on track 2.

Remaining "funnies" have all been mentioned above with reference to the DPB:
(EXM) maximum extent folding is not essential;
(DRM) more blocks may be allocated to the directory than are actually used by the directory; and
(CKS) a reduced number of directory sectors may be checked.

THE TABLES

The following tables are derived from numerous sources including magazine articles, manuals for various items of equipment, and disassembling several BIOSes and programs for doing lots or formats. The whole point of my doing the tables in the first place was simply to try to get the data into consistent form. I hope the resulting explanation is understandable (well, at least on second reading).

I have used an exclamation mark (!) to flag items I guess are wrong, but there may be other errors too. Question marks (?) mean that I think I've interpreted the relevant source correctly - but I might be quite mistaken. Underlined (_) means I know it's odd, but I'm convinced it's right all the same. Within any one subdivision, formats are listed in order of increasing DSM.

All the values of your own DPB can be deduced by using the above equations on the information given to you by STAT when you type: STAT DSK:<CR>. Unfortunately this only works on "home" disks, STAT is not capable of deducing the DPB of any old "alien" disk you might slot into one of your drives.

Data about:
skew;
host sector size and numbering;
data inversion;
where double density starts on a double density disk; and
what scheme is used for access to the "other" side
is rather harder to winkle out.

[Ed: to be continued next month.]

* *

AN BIDDLEONIAN WALKED INTO A HOSPITAL WITH HIS ARM PIT BLOWN UP. THE DOCTOR SAID; "YOU'RE THE FIFTH PERSON WITH A BLOWN UP ARMPIT WE'VE HAD THIS WEEK. HOW DID IT HAPPEN."
"WELL DOCTOR," REPLIED THE BIDDLEONIAN, "I PULLED THE PIN OUT OF MY HANDGRENADE AND STARTED COUNTING...(COUNTS ON FINGERS)...ONE...TWO...THREE...FOUR...FIVE...(TUCKS HANDGRENADE UNDER ARM TO USE OTHER HAND)...SIX...SEVEN...EIGHT...BOOM."

FILE TRANSFER BETWEEN SORCERER AND MICROBEE

Brian Riley

This article describes how to transfer Microbee Wordbee files to the Sorcerer which is running under STERM93. I have tried the following connection by direct connect with success at both 300 and 1200 baud. The following shows the connection:

```
        MicroBee                    Sorcerer

      pin 2 XMT--------------pin 3 RCV->>>>(-12Volts via 1K resistor)
      pin 3 RCV--------------pin 2 XMT
      pin 7 GRD--------------pin 7 GRD
```

The -12 Volts is required because the Microbee can have problems driving some RS232 devices and was derived from a -16 Volts supply used in the S-100 Expansion unit.

```
< -16 V------------------c(BC558)e------>  -12 volts
      -!                 ↑        !
      --- 1000uf         ↑1.5Kohm !          where: c=collector
   capacitor             ↑resistor!                 e=emitter
      --- 25VW            !        b                 b=base
      +!                  !--------!                 A=anode
       !                           ! A               K=cathode
     grd.                    (13V zener diode)
                                   ! K
                                   !
                                 grd.
```

The -16 volts comes from the -ve. side of the 4000uf. capacitor closest to the S-100 sockets. [Ed: see also note re Serial Interface N/L Vol84/118. Maybe, just maybe you may not need the -12v pull down resistor?]

Project 676 from ETI Feb. 1984 describes a "fair dinkum RS232er" for the Microbee which could be used if you haven't got the S100 Expansion unit.(The ICL7660 chip can be hard to get although All Electronic Components of Lonsdale St; Melb. have it in stock now and then - ring before making a trip..)

OPERATION NOTES:

(1) Load the file into the Microbee under WORDBEE then exit to the NETWORK Rom via the 'N' command. Set up Baud rate and leave other values as default.(ie 8 data bits)
(2) Load STERM93 and go to file receive mode (make sure UART parameters are set for 8 bits and same Baud rate as Microbee).
(3) Answer prompts from STERM ie filename & display data.
)4) On the Microbee give the 'SAVE' command and soon after the transfer should begin.
(5) The file, if it was named with a .WPF suffix can then be accessed via JIM NELSON'S SWORD program. [N/L Vol84/73.]
(6) Files can be sent from the Sorcerer back to the Microbee by using the 'S' command in STERM93 and the 'LOAD' command on the Microbee. (Be sure to exit from the Network Rom in the Microbee back to the Wordbee Rom by using the 'RETURN' key, as pressing reset will corrupt the Wordbee file!).

So far, successful file transfers have been name and address listings of about 500 people in my tennis club from the Microbee to the Sorcerer, and The Australian Beginning New Feature notes from the Sorcerer to the Microbee. (The Microbee was the 32k series 2 Personal Communicator)

* *

VECTOR 'OUTPUT' TO WORD-PROCESSOR FILES

John Carragher
18 Essington Street, Flagstaff Hill, S.A. 5159.

The following code allows output from SORCERER to be sent to a WP file (or any area of RAM). The bytes at F045/6 are used as a pointer to the next location to be used, and F047/8 are used to set the last useable address for output. The code from F030 sets up these pointers for your current WP file, to append the output to the WP file, and the code from F04E is the output driver.

As an example, to add a Hex dump to a WP file, you would proceed as follows:-

(1) Use 'X' to exit from th WP to Monitor.
(2) Load the code from F030 to F07F.
(3) GO F030
(4) SE 0=F04E
(5) DU ADDR1 ADDR2
(6) Restore the Video driver using CTL-C.

SHEDDING A BIT OF LIGHT ON DISK FORMATS - The Tables.

Andrew Marland.

[Ed: This is part 2 of Andrew's huge article, the first part being published last month. Please refer to Part 1 for explanations of the various symbols.]

SINGLE SIDED 5.25" FORMAT TABLES FOR CP/M

5" 48 tpi SS SD - Soft sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk	tk	Host sec: size	1st:	data Inv?	double den	sid
Osborne 1	14 00	04	0F	01	2D 00	3F 00	80 00	10 00	03 00	OSB1	4	2K	92K	40	256	1	no	SD	SS
TRS-80 I Omikr.	12 00	03	07	00	47 00	3F 00	C0 00	10 00	03 00	TRS1	4	1K	72K	35	128	1	no	SD	SS
Cromenco	12 00	03	07	00	52 00	3F 00	C0 00	10 00	03 00	CDOS1	5	1K	83K	40	128	1	no	SD	SS
Xerox 820	12 00	03	07	00	52 00	1F 00	80 00	08 00	03 00	CDOS1	5	1K	83K	40	128	1	no	SD	SS
Sorcerer Dtrio.	12 00	03	07	00	54 00	3F 00	C0 00	10 00	02 00	0	0	1K	85K	40	128	1	no	SD	SS

5" 48 tpi SS DD - Soft sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk	tk	Host sec: size	1st:	data Inv?	double den	sid
35 tracks																			
Aardvark	28 00	04	0F	01	51 00	3F 00	80 00	10 00	02 00	Aard	08	2K	164K	35	512	1	no	DD	SS
Superbrain (JR)	28 00	04	0F	01	51 00	3F 00	80 00	10 00	02 00	Aard	08	2K	164K	35	512	1	yes	DD	SS
CCS SS DD	28 00	03	07	00	9F 00	3F 00	C0 00	10 00	03 00	CCS	18	1K	160K	35	1024	1	no	DD	SS
40 tracks																			
Sorcerer Exidy	28 00	04	0F	00	4B 00	7F 00	C0 00	20 00	02 00	Exidy	0A	2K	152K	40	256	1	no	all	SS
Heath Magnolia	24 00	04	0F	01	52 00	5F 00	C0 00	10 00	03 00	0	0	2K	166K	40	512	1	no	DD	SS
TRS-80 III (NM)	24 00	04	0F	01	54 00	7F 00	C0 00	20 00	02 00	0	0	2K	170K	40	256	1	no	all?	SS
Cromenco+IntlT.	28 00	04	0F	01	5E 00	7F?00	F0?00	20 00	02 00	IntlT	0C	2K	190K	40	512	1	no	DD	SS
Morrow	28 00	04	0F	01	5E 00	7F 00	C0 00	20 00	02 00	CCS	18	2K	190K	40	1024	1	no	all?	SS
Sorcerer CData.	28 00	04	0F	00	5E 00	7F 00	C0 00	20 00	02 00	CData	08	2K	190K	40	512	1	no	all	SS
Superbrain 40tk	28 00	04	0F	01	5E 00	3F 00	80 00	10 00	02 00	Aard	08	2K	190K	40	512	1	yes	DD	SS
NEC 8001 PC	28 00	03	07	00	97 00	3F 00	C0 00	10 00	02 00	0	0	1K	152K	40	256	1	no	DD	SS
Zenith Z-100	28 00	03	07	00	97 00	7F 00	F0 00	20 00	02 00	0	0	1K	152K	40			no	DD	SS
Zenith Z-100(!)	28 00	03	07	00	97 00	7F 00	C0!00	20 00	02 00	0	0	1K	152K	40	512	1	no	1?	SS
Zenith Z-90	28 00	03	07	00	97 00	7F 00	F0 00	20 00	02 00	0	0	1K	152K	40	256	1	no	DD	SS
IBM PC CP/M-86	28 00	03	07	00	9B 00	3F 00	C0 00	10 00	01 00	0	0	1K	156K	40	512	1	no	DD	SS
TI Professional	28 00	03	07	00	9B 00	3F 00	C0 00	10 00	01 00	0	0	1K	156K	40	512	1	no	DD	SS
Xerox 820 DD	22 00	03	07	00	9C 00	3F 00	C0 00	10 00	03 00	0	0	1K	157K	40	256	1	no	DD	SS
Lobo max 80	24 00	03	07	00	A5 00	3F 00	C0 00	10 00	03 00	0	0	1K	166K	40	256	0	no	DD	SS
Actrix	24 00	03	07	00	AA 00	3F 00	C0 00	10 00	02 00	Actrx	0C	1K	171K	40	512	1	no	DD	SS
DEC Robin VT180	24 00	03	07	00	AA 00	3F 00	C0 00	10 00	02 00	Aard	08	1K	171K	40	512	1	no	DD	SS
Olivetti ETV300	24 00	03	07	00	AA 00	3F 00	C0 00	10 00	02 00	Olive	04	1K	171K	40	256	1	no	1	SS
Sorcerer Dtrio.	24 00	03	07	00	AA 00	3F 00	C0 00	10 00	02 00	0	0	1K	171K	40	256	1	no	1	SS
Osborne 1 (DD)	28 00	03	07	00	B8 00	3F 00	C0 00	10 00	03 00	0	0	1K	185K	40	1024	1	no	DD	SS
ATR 8000	28 00	03	07	00	BD 00	3F 00	C0 00	10 00	02 00	0	0	1K	190K	40			no	DD	SS
Cromenco CDOS	28 00	03	07	00	BD 00	3F 00	C0 00	10 00	02 00	CDOS	16	1K	190K	40	512	1	no	DD	SS
Kaypro II	28 00	03	07	00	C2 00	3F 00	F0 00	10 00	01 00	0	0	1K	195K	40			no	DD	SS
TRS-80 4 (Aero)	28 00	03	07	00	C2 00	3F 00	F0 00	10 00	01 00	0	0	1K	195K	40			no	DD	SS

5" 96 and 100 tpi SS DD - Soft and hard sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk tk	Host sec: size	1st	data Inv?	double den	sid
77 or 80 tracks & 96 or 100 tpi																		
Sorc. Exidy HS:	28 00	04	0F	00	95 00	7F 00	C0 00	20 00	02 00	Exidy 0A		2K	300K 77	256	1	no	all	SS
Sorc. Microp HS:	28 00	04	0F	00	95 00	7F 00	C0 00	10 00	02 00	Microp0A		2K	300K 77	256	1	no	all	SS
Sorc. Exidy SS:	28 00	04	0F	00	95 00	7F 00	C0 00	20 00	02 00	Exidy 0A		2K	300K 77	256	1	no	all	SS
Sanyo MBC	28 00	04	0F	01	9B 00	3F 00	80 00	10 00	02 00	Sanyo 06		2K	312K 80	256		no	all?	SS
Sorcerer Dtrio.:	24 00	04	0F	01	AE 00	3F 00	80 00	10 00	02 00	0	0	2K	350K 80	256	1	no	1	SS
Sorcerer CData.:	28 00	04	0F	00	C2 00	7F 00	C0 00	20 00	02 00	CData 00		2K	390K 80	512	1	no	all	SS

DOUBLE SIDED 5.25" FORMAT TABLES FOR CP/M

5" 48 tpi DS SD - Soft sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk tk	Host sec: size	1st	data Inv?	double den	sid
Sorcerer Dtrio.:	24 00	03	07	00	AA 00	3F 00	C0 00	10 00	02 00	0	0	1K	171K 40	128	1	no	SD	sec
Cromemco	12 00	03	07	00	AC 00	3F 00	C0 00	10 00	03 00	CDOS1	5	1K	173K 40	128	1	no	SD	tks

5" 48 tpi DS DD - Soft sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk tk	Host sec: size	1st	data Inv?	double den	sid
35 tracks																		
Superbrain QD	28 00	04	0F	01	A9 00	3F 00	80 00	10 00	02 00	Aard 08		2K	340K 35	512	1	yes	DD	35+
HP-125	20 00	03	07	00	FB 00	7F 00	F0 00	20 00	03 00	0	0	1K	250K 35	256	0	no	DD	tks
Toshiba T100	40 00	03	07	00	FF 00	3F 00	C0 00	10 00	03 00	Tosh 00		1K	256K 35	256	1	no	DD	sec
40 tracks																		
Datavue	50 00	05	1F	03	5E 00	7F 00	80 00	20 00	02 00	0	0	4K	380K 40	512	01	no	DD	sec
IMS 5000	48 00	04	0F	01	97 00	3F 00	80 00	10 00	02 00	IMS	10	2K	384K 40	256	1	no	DD	sec
NEC PC-8001A	48 00	04	0F	01	97 00	7F 00	C0 00	20 00	02 00	0	0	2K	384K 40	256	1	no	DD	sec
Zenith Z-100	20 00	04	0F	00	9A 00	FF 00	F0 00	40 00	02 00	0	0	2K	310K 40	512	1	no	DD	tks
Sanyo	20 00	04	0F	01	9B 00	3F 00	80 00	10 00	02 00	Sanyo	6	2K	312K 40	256	1	no	DD	tks
IBM PC CP/M-86	20 00	04	0F	01	9D 00	3F 00	80 00	10 00	01 00	0	0	2K	316K 40	512	1	no	DD	79-
TeleVideo(?)	48 00	04	0F	01	A5!00	3F 00	80 00	10 00	02 00	0	0	2K	332K 39!!	256	1	no	all?	sec
TeleVideo	24 00	04	0F	00	AA 00	3F 00	80 00	10 00	04 00	0	0	2K	342K 40	256	1	no	DD	tks
Sorcerer Dtrio.:	48 00	04	0F	01	AA 00	3F 00	80 00	10 00	02 00	0	0	2K	342K 40	256	1	no	1	sec
Actrix DD	48 00	04	0F	01	AE 00	3F 00	80 00	10 00	01 00	Actrx 0C		2K	350K 40	512	1	no	DD	sec
Otrona Attache	28 00	04	0F	01	B5 00	7F 00	C0 00	20 00	03 00	0	0	2K	364K 40	512	1	no	DD	40+
Epson QX-10	28 00	04	0F	01	BD 00	7F 00	C0 00	20 00	04 00	0	0	2K	380K 40	512	1	no	DD	tks
Cromemco CDOS	28 00	04	0F	00	C2 00	7F 00	C0 00	20 00	02 00	CDOS 10		2K	390K 40	512	1	no	DD	tks
Cromemco+IntlT	50 00	04	0F	01	C2 00	7F 00	C0 00	20 00	01 00	IntlT 0C		2K	390K 40	512	1	no	DD	sec
MAGIC computer	28 00	04	0F	00	C2 00	3F 00	80 00	10 00	02 00	0	0	2K	390K 40	512	1	no	DD	tks
Morrow	28 00	04	0F	01	C2 00	BF 00	E0 00	30 00	02 00	CCS	10	2K	390K 40	1024	1	no	all?	tks

5" 96 and 100 tpi DS DD - Soft sectored:

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk tk	Host sec: size	1st	data Inv?	double den	sid
77 or 80 tracks & 96 or 100 tpi																		
Sorc. Exidy	40 00	04	0F	00	2B	01	7F 00	C0 00	20 00	02 00	Exidy	0A	2K	600K	77	256	1	no all sec
Sorcerer Dtrio.	48 00	04	0F	00	5E	01	7F 00	C0 00	20 00	02 00	0	0	2K	702K	00	256	1	no all sec
Sorcerer CData.	50 00	04	0F	00	85	01	7F 00	C0 00	20 00	02 00	CData	00	2K	780K	00	512	1	no all sec

8" FORMAT TABLES FOR CP/M

Parameter Name	Hex dump of CP/M disk parameter block (DPB) SPT	BSH	BLM	EXM	DSM	DRM	AL0&1	CKS	OFF	Skew table	F	Capacity blk	dsk tk	Host sec: size	1st	data Inv?	double den	sid
SSSD																		
STANDARD (!)	1A 00	03	07	00	F2	00	3F 00	C0 00	10 00	02 00	8"	6	1K	243K	77	128	1	no SD SS
SSDD																		
ADD-X CP/M	34 00	04	0F	01	F2	00	7F 00	C0 00	20 00	02 00	0	0	2K	486K	77		1	no 2 SS
Sorcerer Dtrio.	34 00	04	0F	01	F2	00	3F 00	80 00	10 00	02 00	0	0	2K	486K	77	256	1	no 1 SS
DSSD																		
ADD-X CP/M	34 00	04	0F	01	F2	00	7F 00	C0 00	20 00	02 00	0	0	2K	486K	77		1	no SD sec
Sorcerer Dtrio.	34 00	04	0F	00	F2	00	3F 00	80 00	10 00	02 00	0	0	2K	486K	77	128	1	no SD sec
DSDD																		
ADD-X CP/M	68 00	05	1F	03	F2	00	7F 00	80 00	20 00	02 00	0	0	4K	972K	77		1	no 2 sec
TRS-80 II exten	48 00	04	0F	00	2B	01	7F 00	C0 00	20 00	02 00	TRS?	18	2K	1200K	77	1024	1	no all? tks
Sorcerer Dtrio.	68 00	04	0F	00	E6	01	7F 00	C0 00	20 00	02 00	0	0	2K	974K	77	256	1	no 1 sec

SKEW TABLES

128 bytes per sector

```
TRS1  = TRS-80            :01 05 09 0D 11
  SPT = 12, F = 4          03 07 0B 0F
                           02 06 0A 0E 12
                           04 08 0C 10     .... 01

CDOS1 = Cromenco & Xerox  :01 06 0B 10
  SPT = 12, F = 5          03 08 0D 12
                           05 0A 0F
                           02 07 0C 11
                           04 09 0E        .... 01

8" STANDARD skew table    :01 07 0D 13 19
  SPT = 1A, F = 6          05 0B 11 17
                           03 09 0F 15
                           02 08 0E 14 1A
                           06 0C 12 18
                           04 0A 10 16     .... 01
```

256 bytes per sector

```
OSB1  = Osborne SD        :01 02 05 06 09 0A 0D 0E 11 12
  SPT = 14, F = 4          03 04 07 08 0B 0C 0F 10 13 14  .... 01

Olive = Olivetti ETV 300  :01 02 05 06 09 0A 0D 0E 11 12 15 16 19 1A 1D 1E 21 22
  SPT = 24, F = 4          03 04 07 08 0B 0C 0F 10 13 14 17 18 1B 1C 1F 20 23 24  ... 01
```

```
Sanyo = Sanyo MBC          :01 02  07 08  0B 0E  13 14  19 1A  1F 20
  SPT = 28, F = 6           05 06  0B 0C  11 12  17 18  1B 1E
                            03 04  09 0A  0F 10  15 16  1B 1C  ...  01

Tosh  = Toshiba T100       :01 02  09 0A  11 12  19 1A
  SPT = 40, F = 8           03 04  0B 0C  13 14  1A 1B
                            05 06  0D 0E  15 16  1B 1E
                            07 08  0F 10  17 18  1F 20
                            21 22  29 2A  31 32  39 3A
                            23 24  2B 2C  33 34  3A 38
                            25 26  2D 2E  35 36  38 3E
                            27 28  2F 30  37 38  3F 40          ...  01

Exidy = Exidy Sorcerer     :0B 0C  15 16  1F 20
  SPT = 28, F = A           09 0A  13 14  1B 1E
                            07 08  11 12  1B 1C
                            05 06  0F 10  19 1A
                            03 04  0D 0E  17 18
                            01 02   ....       08

Microp = Micropolis        :01 02  0B 0C  15 16  1F 20
  SPT = 28, F = A           09 0A  13 14  1B 1E
                            07 08  11 12  1B 1C
                            05 06  0F 10  19 1A
                            03 04  0D 0E  17 18  ...  01

IMS   = IMS 5000           :01 02  11 12  21 22  31 32
  SPT = 40, F = 10          03 04  13 14  23 24  33 34
                            05 06  15 16  25 26  35 36
                            07 08  17 18  27 28  37 38
                            09 0A  19 1A  29 2A  39 3A
                            0B 0C  1B 1C  2B 2C  3B 3C
                            0D 0E  1D 1E  2D 2E  3D 3E
                            0F 10  1F 20  2F 30  3F 40   ....  01
```

512 bytes per sector

```
DEC   = DEC Robin/VT180    :01 02 03 04  09 0A 0B 0C  11 12 13 14  19 1A 1B 1C  21 22 23 24
  SPT = 24, F = 8           05 06 07 08  0D 0E 0F 10  15 16 17 18  1D 1E 1F 20            ... 01

Aard  = Aardvark & SuperB  :01 02 03 04  09 0A 0B 0C  11 12 13 14  19 1A 1B 1C  21 22 23 24
  SPT = 28, F = 8           05 06 07 08  0D 0E 0F 10  15 16 17 18  1D 1E 1F 20  25 26 27 28 ... 01

CData = Sorcerer+Compudata :05 06 07 08  0D 0E 0F 10  15 16 17 18  1D 1E 1F 20  25 26 27 28
  SPT = 28, F = 8           01 02 03 04  09 0A 0B 0C  11 12 13 14  19 1A 1B 1C  21 22 23 24 ... 05

Actrx = Actrix             :01 02 03 04  0D 0E 0F 10  15 16 17 18  1D 1E 1F 20
  SPT = 24, F = C           05 06 07 08  11 12 13 14  19 1A 1B 1C  21 22 23 24
                            09 0A 0B 0C  15 16 17 18  1D 1E 1F 20     ...    01  if single sided
  SPT = 48, F = C           25 26 27 28  31 32 33 34  39 3A 3B 3C  41 42 43 44
                            29 2A 2B 2C  35 36 37 38  3D 3E 3F 40  45 46 47 48
                            2D 2E 2F 30  39 3A 3B 3C  41 42 43 44  49 4A 4B 4C
                            31 32 33 34  3D 3E 3F 40  45 46 47 48     ...    01  if double sided

IntlT = Cromemco+IntlTerm  :01 02 03 04  0D 0E 0F 10  15 16 17 18  1D 1E 1F 20
  SPT = 28, F = C           09 0A 0B 0C  15 16 17 18  1D 1E 1F 20  25 26 27 28
                            05 06 07 08  11 12 13 14  19 1A 1B 1C  21 22 23 24
                            29 2A 2B 2C  35 36 37 38  41 42 43 44  49 4A 4B 4C
  SPT = 50, F = C           31 32 33 34  3D 3E 3F 40  45 46 47 48  4D 4E 4F 50
                            2D 2E 2F 30  39 3A 3B 3C  41 42 43 44  49 4A 4B 4C
                            35 36 37 38  41 42 43 44  49 4A 4B 4C     ...    01  if double sided
```

```
CDOS  = Cromenco CDOS DD  :Ø1 Ø2 Ø3 Ø4  11 12 13 14  21 22 23 24
  SPT = 28, F = 1Ø         Ø9 ØA ØB ØC  19 1A 1B 1C
                           Ø5 Ø6 Ø7 Ø8  15 16 17 18  25 26 27 28
                           ØD ØE ØF 1Ø  1D 1E 1F 2Ø               ...   Ø1

1Ø24 bytes per sector

CCS   = CCS & Morrow      :Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8  19 1A 1B 1C 1D 1E 1F 2Ø
  SPT = 28, F = 18         Ø9 ØA ØB ØC ØD ØE ØF 1Ø  21 22 23 24 25 26 27 28
                           11 12 13 14 15 16 17 18          ...         Ø1

TRS   = TRS-8Ø mod II     :Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8  19 1A 1B 1C 1D 1E 1F 2Ø  31 32 33 34 35 36 37 38
  SPT = 4Ø, F = 18         Ø9 ØA ØB ØC ØD ØE ØF 1Ø  21 22 23 24 25 26 27 28  39 3A 3B 3C 3D 3E 3F 4Ø
                           11 12 13 14 15 16 17 18  29 2A 2B 2C 2D 2E 2F 3Ø ... Ø1
```

* *

STOP PRESS ITEMS. [Ed: last minute addition]
===============

S.C.U.A. MONITOR for 8Ø and 64 columns. Frank Schuffelen.
=======================================

Following Rick Millicer's NEW DIRECTION article in Vol. 84-135, we are please to advise the availability of the new monitor for 8Ø and 64 column SORCERERS. As well as providing Jim Graham's 64 column monitor (version 1.1 plus extra commands, as described in the September 1983 N/L in one half of a pair of 2732 EPROMs, the new SORCERER monitor in the other half features the following:

A. INPUT/OUTPUT DRIVERS:
----

1. Superior keyboard routine with auto-repeat (simulates interrupt driven keyboard), bell, key-click, disk-boot and user configurable keypad plus 3 standard configurations: ADM-3A, WORDSTAR, NORMAL NUMERIC.
2. Video terminal emulation, ADM-3A plus enhancements, such as: Inverse video, Flashing or block cursor, Cursor on or off, Clear to end of line, Clear to end of screen, Graphic line drawings, Text and graphics mode.
3. Implementation of the CP/M IO byte for redirection of input/output. Enables software selection of say a printer driver.
4. Standard CP/M vectors, being CONSTAT, CONIN, CONOUT, (CONNIVE phantom pheature??), LIST, PUNCH, READER and LISTST.
5. Parallel printer routine, compatible with BOTH 7 and 8 bit printers.
6. Serial driver using XON/XOFF.

(Sound functions use parallel output port and hardware as used for your games.)

B. MONITOR FUNCTIONS:
----

1. Memory search in Hex or ASCII with wild cards.
2. Dump in Hex and ASCII - 8Ø column format.
3. Enter characters in Hex, decimal or ASCII.
4. Hex, Decimal, ASCII conversion.
5. Hex and Decimal sum and difference.
6. Screen print.
7. Clear memory.
8. Move command which works correctly with overlapping memory segments.
9. 'Turn-key' auto boot.
1Ø. Plus the now 'standard' enhancements and a few more, over Exidy Monitor 1.Ø

To order your copy of the pair of 2732 EPROMs containing BOTH MONITORS, send a cheque or money order for $4Ø.ØØ to S.C.U.A., including information about disk system used, your boot address and first byte of the bootstrap for the auto-boot feature (eg. B9ØØ, BCØØ, BFØØ, D8ØØ, or whatever + F3, 3E or ??). Please allow approximately 3 weeks for delivery. Please DO NOT PIRATE these monitors, as several of our members have spent many hours of hard work creating them.

* *

Thank heavens that's over. What a mammoth this was!

APPENDIX E
ASCII CHARACTER CODE

ASCII stands for American Standard Code for Information Exchange.

Decimal and Hexadecimal Designations of the Standard Characters

DEC.	HEX.	CHAR.	DEC.	HEX.	CHAR.	DEC.	HEX.	CHAR.
000	00	NUL	031	1F	US	062	3E	>
001	01	SOH	032	20	SPACE	063	3F	?
002	02	STX	033	21	!	064	40	@
003	03	ETX	034	22	"	065	41	A
004	04	EOT	035	23	#	066	42	B
005	05	ENQ	036	24	$	067	43	C
006	06	ACK	037	25	%	068	44	D
007	07	BEL	038	26	&	069	45	E
008	08	BS	039	27	'	070	46	F
009	09	HT	040	28	(	071	47	G
010	0A	LF	041	29	)	072	48	H
011	0B	VT	042	2A	*	073	49	I
012	0C	FF	043	2B	+	074	4A	J
013	0D	CR	044	2C	,	075	4B	K
014	0E	SO	045	2D	-	076	4C	L
015	0F	SI	046	2E	.	077	4D	M
016	10	DLE	047	2F	/	078	4E	N
017	11	DC1	048	30	0	079	4F	O
018	12	DC2	049	31	1	080	50	P
019	13	DC3	050	32	2	081	51	Q
020	14	DC4	051	33	3	082	52	R
021	15	NAK	052	34	4	083	53	S
022	16	SYN	053	35	5	084	54	T
023	17	ETB	054	36	6	085	55	U
024	18	CAN	055	37	7	086	56	V
025	19	EM	056	38	8	087	57	W
026	1A	SUB	057	39	9	088	58	X
027	1B	ESC	058	3A	:	089	59	Y
028	1C	FS	059	3B	;	090	5A	Z
029	1D	GS	060	3C	<	091	5B	[
030	1E	RS	061	3D	=	092	5C	\

DEC.	HEX.	CHAR.	DEC.	HEX.	CHAR.	DEC.	HEX.	CHAR.
093	5D	]	105	69	i	117	75	u
094	5E	^	106	6A	j	118	76	v
095	5F	<	107	6B	k	119	77	w
096	60	'	108	6C	l	120	78	x
097	61	a	109	6D	m	121	79	y
098	62	b	110	6E	n	122	7A	z
099	63	c	111	6F	o	123	7B	{
100	64	d	112	70	p	124	7C	¦
101	65	e	113	71	q	125	7D	}
102	66	f	114	72	r	126	7E	~
103	67	g	115	73	s	127	7F	DELETE
104	68	h	116	74	t			(Rubout)

Standard Abbreviations for ASCII characters 0 through 31 (00 through 1F Hex.)

ACK = Acknowledge
BELL = Bell
BS = Backspace
CAN = Cancel
CR = Carriage Return
DC1 = Direct Control 1
DC2 = Direct Control 2
DC3 = Direct Control 3
DC4 = Direct Control 4
DLE = Data Link Escape
EM = End of Medium
ENQ = Enquiry
EOT = End Of Transmission
ESC = Escape
ETB = End Transmission Block
ETX = End Text
FF = Form Feed
FS = Form Separator
GS = Group Separator
HT = Horizontal Tab
LF = Line Feed
NAK = Negative Acknowledge
NUL = Null
RS = Record Separator
SI = Shift In
SO = Shift Out
SOH = Start of Heading
STX = Start Text
SUB = Substitute
SYN = Synchronous Idle
US = Unit Separator
VT = Vertical Tab

concurrently and reuse disks for any soft sectored system.

3. Vista = Hard sectored 5", 48 TPI. Compatibility: drives only. Upgrade means new controller, new disks. Old drives still useable but capacity limited. Power supply and box still good though.

4. Disk Jockey = S100-based soft sector controller for 8" drives. Compatibility dependent on software. No 5" capability.

5. Digitrio = S100-independent soft sector controller for any 8" and 5" drives. Compatibility dependent on software: Dream Bios. Dream Bios will read 48 TPI (40T) disks in a 96 TPI (80T) drive.

We are working on Micropolis upgrades. Possibly, new controller may, relatively cheaply, resolve some of the pressing upgrade problems.

* *

48 to 56K DRAM - Mark 2

Rick Millicer.
199 Lower Heidelberg Road, Ivanhoe, Vic 3079. (03) 497-3435.

Parts: 1 x LS02. 1 x SPST switch. 1 x 3k3 resistor. 1 x 47R resistor. 8 x 4116 DRAM or equivalents.

Explanation of terms:

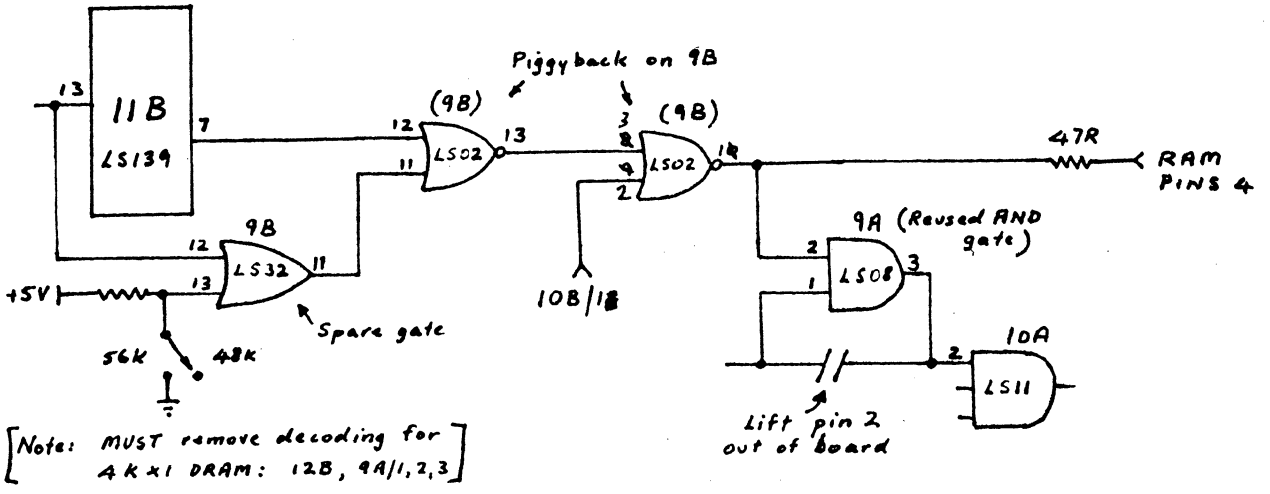
9B/1	=	pin one of chip 9B
9B/1 hole	=	pad where pin one of chip 9B was before lifting it out of the board.
(9B)/1	=	pin one of piggybacked chip on top of 9B.

1. Test new RAM chips by installing them in Row 2 and run a memory test program over 4000 to 7FFF. Always remove PS board before removing DRAM.

Steps 2 & 3 remove decoding for obsolete 4k x 1 DRAM chips. See Oct/Nov 83, Yearbook V p24.

2. Lift out 9A/1,2,3. Jumper 9A/2 hole to 9A/3 hole.
3. Remove 12B (LS00). Jumper 12B/2 hole to 12B/6 hole. Reconnect power and test whether so far so good.
4. Lift out 10A/2. Jumper 10A/2 hole to 9A/1.
5. Jumper 9A/3 to 10A/2.
6. Bend out pins 1,2,3,4,5,6,_,8,9,10,_,12,13,_ of the new LS02. Piggyback it on 9B and solder the down legs (7,11,14). This top chip is now known as (9B).
7. Jumper 11B/13 to 9B/12.
8. Wire pull-up resistor (2k2 - 4k7) between 9B/13 and 5V track.
9. Jumper (9B)/12 to 11B/7.
10. Jumper (9B)/13 to (9B)/8.
11. Jumper (9B)/9 to 10B/1.
12. Jumper (9B)/10 to 9A/2.
13. Piggyback old RAM chips on top of new chips, leaving pin 4 of each top chip bent out. Clean off ALL REMAINING SOLDER FLUX from legs of lower chips. This is important! (Also never solder chips together while socketted!!!)
14. Replace all double RAMs into row 2. Connect all pin 4's of top row together and then to (9B)/10 via 47R resistor.
15. Connect one pole of switch to 9B/13 and the other pole to ground.

The 48/56k switch may be mounted on the top case or interlocked with ROMPak insertion. *MEMOFF (derived from *ROMPRE) may be used. There are two spare wires in the ribbon cable running to the keyboard - one is that mentioned in Dec 83 page 13, Yearbook V p20, and the other is a useless -5V on pin 2. You may use both wires for different signal switching using ground on the keyboard for logic 0.



I strongly recommend this mod. to anyone requiring the best possible green or amber output regardless of program type. Nice work, Mick.

==========

Excalibur and BASIC By Bob Green.

The following Commands are in (ROM) but unfortunately were never implemented. LSET,PSET,SPEEK and SPOKE. So, what are you missing out on ?.... Probably one of the most valuable attributes for the Excalibur, would be the ability to SET individual screen pixels. Complete control of colour and high resolution graphics are just not possible without this function. Well all is not lost, Kinetic Systems have had ideas for some time about a re-vamp of BASIC. Avenues explored included exchange of the 16K Basic Eproms to a healthy 32K thereby permitting heaps of additional routines such as a Monitor program, Routines for LSET, PSET, DRAW, CIRCLE, LINE etc, Routines for Cassette users which would enable greater flexibility, the mind boggles !!!!!! However Kinetics do not have the time to write these routines, but do have contacts with certain people that can. The question is, how many users want an update and are prepared to pay for it? I for one are 100% in favour. In order to get an idea of the demand, I propose that all those interested drop me a line pronto or ring, and let me know the kinds of things you would like to see included in a new Excalibur Basic. Extensive testing would be a prime requirement,in order to ensure bug-free ROMs, as would be the need to bank-select the additional ROM's to ensure that there is no (RAM MEMORY) lost in the transition. The other possibility is that certain users may have the necessary know-how to write some of the routines needed. If so, PLEASE come forward and reveal your shining talents. My postal address is: 3 Wooton Ct, Melton South, Vic 3338. Phone 743-3149.

Using USR

It is possible to call any of the routines in ROMS, from BASIC by the use of the USR Statement, eg; to call the CLS Routine we must force the CPU to jump to location 58 Hex. OK! type in the following program;

```
10  Poke -958,88:Poke -957,0
20  X= USR (0)
```

and run. You will be rewarded with a Clear Scrn and READY>. How does this work?..Simple, Decimal 88 in line 10 is the equivalent of 58H (your ROM routine location) and is the (LSB) value, Dec 0 is Equiv to 00Hex and is the (MSB) value. Therefore we have poked into locations 957-958 the Hex address of 0058. Line 20 then causes the CPU to jump to line 10 addresses and do a further jump to the CLS routine in Rom. All the other jump table functions should be available in the same way. To return to MEMORY SIZE Message, use LSB=1 : MSB=1

A Brick-bat

I am getting a lot of requests for Cassette Software, but no-one has sent any programmes to assist in building up the Libary. Without your help, this service will cease to function. Let's have some programmes to help maintain some perspective. Your cassette will be returned, together with any latest additions - if applicable). New!- 'Hangman' and 'Programming the 6845'

Extended Disc BASIC.

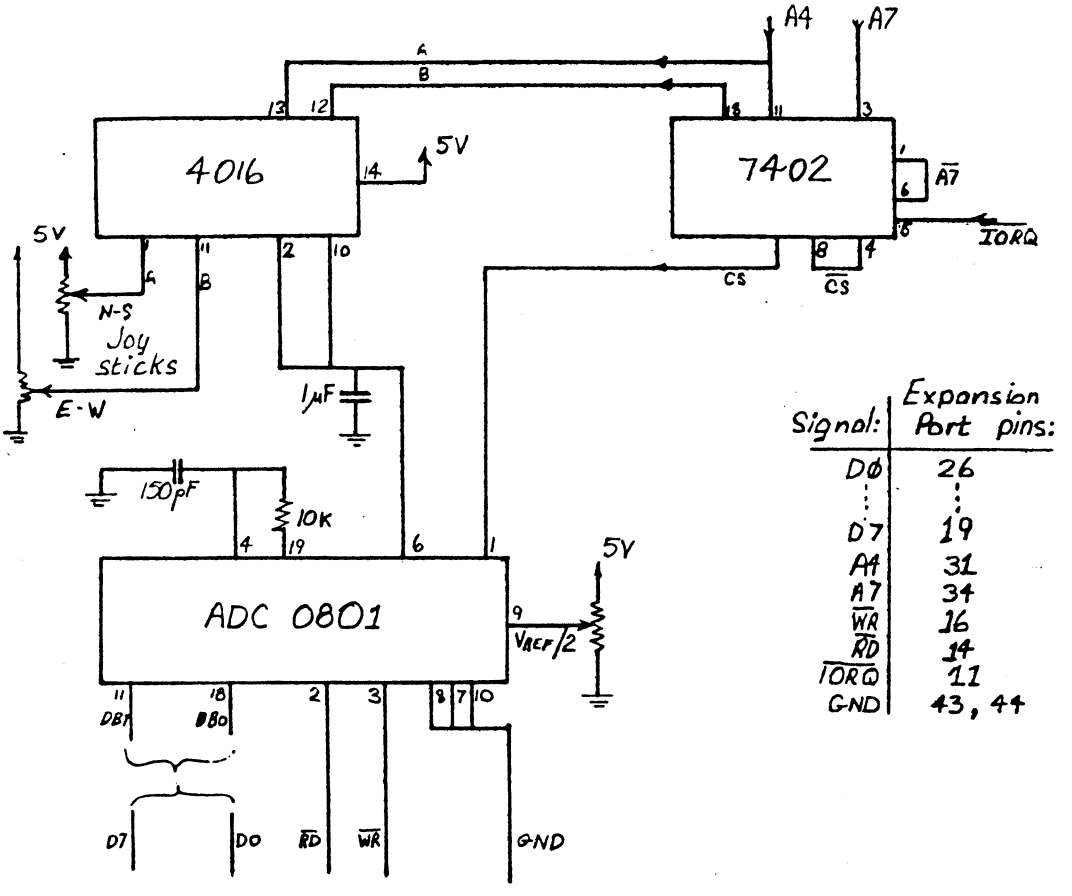
Did anybody notice the article in E.T.I re. an Extended Basic for the TRS-80, SYSTEM-80?..... Many of the commands made available through this disc or cassette programme would greatly enhance EXCALIBUR'S Rom Basic. After contacting the author of this creation (A Queensland kid), the offer has been made to look into adapting EXTENDED BASIC for the Excalibur. What is needed, is an assembly dump of ROMS on disc in TRS-80 disc format before any further progress can be made. If any user has a TRS-80 or has access to same, please contact me on Melb. (03) 743-3149.

==========

A Proportional Joy-stick for the Excalibur by J. Deerson

Two broad types of joy-stick controller exist: switched, and proportional. The simple switched type has a set of four switches which are operated by N, S, E, or W movement of the stick. A much better arrangement from the operator's viewpoint is the proportional type, which allows a direct "mapping" of the

monitor screen dot positions on to all the joystick positions - i.e. stick movement causes proportional cursor movement. Thus an analog to digital converter (ADC) interface is required. A joystick is good value in many situations. Imagine adapting Wordstar to joystick control! Or drawing pictures on the screen! In fact, it's useful wherever operator controlled cursor movement is required. I used a special ADC chip and created two extra ports to do the job. However there are many other solutions (e.g. use an op-amp and do the a-d conversion in software). My circuit is shown in Fig. 1. At present, all ports from 80H to FFH are unused on the Excalibur. The circuit shows N-S joystick pot sampled by port 80H, and E-W at 90H. A BASIC listing which will move the cursor around the screen is shown below.



Signal:	Expansion Port pins:
D0	26
⋮	⋮
D7	19
A4	31
A7	34
WR	16
RD	14
IORQ	11
GND	43, 44

```
5    OUT 48,10:  OUT 49,0      REM MAKE CURSOR STEADY
10   DEFINT A-C:   OUT 128,0   REM START CONVERSION (N-S)
20   B=INP(128)/10:  B=80*B    REM GET ADC VALUE
30   OUT 144,0                 REM START CONVERSION (E-W)
40   C=(INP(144)/3.2)+B        REM GET ADC VALUE
50   IF C>1919 THEN C=1919
60   A=C/256                   REM "6845" CURSOR REG. VALUE
70   OUT 48,14                 REM      (HIGH BYTE)
80   OUT 49,A
90   A=((C/256)-A)*256         REM GET "6845" CURSOR
100  OUT 48,15                 REM      LOW BYTE VALUE
110  OUT49,A
120  GOTO 10                   REM LOOP BACK INDEFINITELY.
```

Line 5 puts the cursor into "non-blinking" mode on entry to the program. Line 10 starts conversion of the N-S pot value (a Write to port 80H). The digital result , which can have any value in the range 0 to 256, is then divided by 10 to give a value in range 0 to 25 (24 lines on a screen) then multiplied by 80 (characters per line), and assigned to variable B. Dividing first by 10 and converting immediately to an integer value reduces cursor jitter, as slight variations in voltage and pot position are rounded off. Line 30 gets the E-W digital value; dividing by 3.2 reduces the range to 0 to 80. C is assigned the sum of this value and B. Thus C value now represents the pot position as a screen character position in range 0 to 1919 approximately.

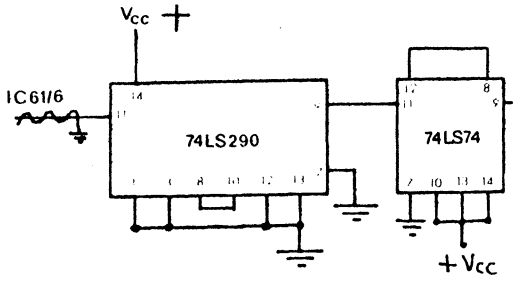
It's all plain sailing now. Line 50 restricts C to the proper range. Line 60 gets the value to be placed in the High byte of the 6845 Cursor Position regis-

ter pair and lines 70, 80 place it there. Line 90 gets the (integer) remainder and puts this in the Low register. Then do the whole thing again. I bought a neat little joy-stick at Dick Smith just before Christmas for $4.50. The I.C.s may amount to $10 or so depending on what you have lying around the place. I mounted the whole circuit on an I.C. prototyping board. Be careful when connecting the wires to the expansion bus: bus contention could arise if you connect up the wrong address lines for example.

This project is a sort of follow-up and complement to the light-pen I constructed some time ago. What about a mouse, anybody? Another "urgent" project is expansion RAM - say another 64k - to take further advantage of the Excalibur's excellent graphics capability. Present RAM prices should get you this for about $50 I would say. Well worth it! Of course, all these projects would be so much easier if there was a plug-in PCB capability on the Excalibur, e.g. STD or S100. Ideas?

==========

Errata



In the video mod. last issue there were two mistakes in the circuit as submitted. The corrected circuit is shown on the left. Note that the 0.1 microfarad capacitors must be the monolithic sort (my mistake - Ed).

==========

Digressing with Don **by Don McKenzie.**

Instead of writing an Excalibur related article this month, I decided to have a general answer session. The questions have been asked many times already by many users, and I answer the same questions over and over to each individual, either by phone or letter. So here they are for the benefit of all.

B.G.R. folded some time ago, but for reasons that I can't explain, there has been no mention of this in the newsletter. The truth is, they didn't do their sums correctly on the kit form Excalibur. This coupled to the fact that the Brunswick factory had reported a large robbery that was not covered by insurance, lead to B.G.R. closing shop.

Another fact. You won't get any support now or in the future from B.G.R. or it's directors. It appears that the only support that will be available from now on will come from Kinetic Systems, the users group and its members. The new keyboard is a PC-5500 model and is available from many electronic outlets including Kinetic Systems. It has a Z80A plus a 2732 amongst other things. When a key is pressed, standard ASCII is output in 7 bit format along with a low going strobe. The strobe stays low for only a couple of instructions, then goes high again. The ASCII data stays latched on the output until the next key is pressed. Any keyboard that uses this method of data transfer should work, but a lot of the features will be missing on other ASCII keyboards, as I have re-written the software in the keyboard to cater for the Excalibur specifically. It requires the changeover of ROM 1, and the keyboard ROM at a $30 changeover fee. This changeover is available from me.

A small percentage of new keyboard users have experienced hardware problems with the keyboard. Known fixes:- Run a large insulated conductor between the +5 volt power supply and the small keyboard cable printed circuit board +5 volt connection. This is done to get the voltage as close as possible to +5 volts. Intermittent operation of the programmable keys may occur if the voltage is too low. Connect a 4.7K resistor between pin 4 of the new 74LS74 and 5volt, and/or short out diode D8. This overcomes the problem of some keys being intermittent or giving wrong ASCII values.

Next touchy subject. Chris Moss of Kinetic Systems. (We are now calling him

1: OVERALL STRUCTURE OF BASIC

At the top of the Rom Pac (DFFA & DFFD) are jumps to warm and cold start routines respectively.

COLD START (C000): moves a block of data into the Basic Work Area (BWA, see SCUA N/L 15, August, p4), searches for the top of RAM (ignoring the value already created for this by the Monitor!), initiates the stack and various other pointers, and then jumps to the interactive area.

WARM START (C06B): enters towards the end of the cold start sequence.

THE INTERACTIVE AREA starts at C25D, with the display of the READY message. It waits on the current input vector (usually the Keyboard) and fills the line buffer until it gets a CR. It then calls a routine which adjusts upper to lower case and substitutes tokens for reserved words. It next looks at the first character to see if it is in direct or indirect mode. In the latter case it loads the line into the appropriate slot of the program and returns for the next input. In direct mode it jumps to the STATEMENT PROCESSOR.

THE STATEMENT PROCESSOR at C6AD scans the line pointed to by HL and executes the commands as these are encountered, using the command jump table at C1E0 to locate the relevant routine for each token. In direct mode HL is pointing to the input line buffer. If an expression is met it passes control to the EXPRESSION EVALUATOR. At the end of a line it returns control to the interactive area if in direct mode, or fetches the next line and starts again if running a program.

THE EXPRESSION EVALUATOR at CB8F scans the expression, storing intermediate values and operators in a stack. Unary functions are evaluated at CC6F, using the function jump table at C0C6 to locate the relevant routines.

THE DATA BLOCK between C075 and C2BC is the key to the system. It contains the following messages and tables:

C075: The sign on message.

C0C6: The FUNCTION Table. This contains a list of addresses of the routines used to process BASIC functions (Tokens between AF (SGN) and C6 (MID$)). The address of the required routine is got by using:
(Token - AF) X 2 as the offset into the table.

C0F6: The RESERVED WORDS Table. The first character of each word has its MSB set high. When a line is input the INTERACTIVE area uses this to convert words into tokens.

C1E0: The COMMAND Table. This is similar to the function table but deals with commands (those reserved words which can occur at the start of a line, ie tokens 80 (END) to 9D (NEW)).

C21D: The OPERATOR Table (?): This is a table of three byte entries referred to from CBE7. I think it deals with the binary operators (+,-,*,/,↑,AND,OR). The first byte is checked (for precedence ?) and the following two bytes are addresses.

C232: The ERROR CODES. A list of the two character error codes.

C258: The BWA DATA BLOCK which is moved down to 0100 on a cold start.

C2A3: MESSAGES. The ERROR, IN ,READY,& BREAK messages.

MEMORY MAP OF MAJOR ROUTINES (Useful subroutines NEXT ISSUE)

C000: Warm and cold start initiations
C075: The Data Block
C2RD: Something to do with processing FOR loops
C2E0: Block move subroutine (see part 3)
C2F1: test for OM Error
C30F: Entry points to the error message sequence start here. The actual error message routine starts at C322

INTERACTIVE AREA

C35D: Displays READY message
C366: Line input and processor starts here
C3RR: Inserts a line from the input buffer into the program area
C41A: NEW token processor
C45A: Prints a '?' and then jumps to get input line (used by INPUT)
C467: Converts the input buffer into processable form (lower case to upper, words to tokens, etc)
C52A: Loads input buffer from input vector (see part 3)
C574: Subroutine to do a 16 bit compare (see part 3)
C585: Sends a character to the terminal vector (see part 3)
C5R4: Gets a char from the input vector (see part 3)
C5C6: LIST token processor
C62F: FOR token processor

STATEMENT PROCESSING

C6AD: Processing of a new statement starts here
C6CD: Scan text subroutine (see part 3)
C6DD: RESTORE token processor
C6F8: Check for Control-C, Run/stop, etc
C707: STOP token processor
C709: END token processor
C74R: CONT token processor
C74R: NULL token processor
C75A: CLOAD* and CSAVE* of arrays done here
C7RC: subroutine which tests if char is alpha-capitals (see part 3)
C7C4: ?
C7FA: converts a numeric string into a binary integer (see part 3)
C80F: CLEAR token processor
C855: RUN "
C861: GOSUB "
C872: GOTO "
C890: RETURN "
C8R5: DATA token (skips to EOS - see part 3)
C8R7: REM token (skips to EOL - see part 3)
C8CC: LET "
C926: ON "
C944: IF "
C964: PRINT "
CA01: TAB "
CA1R: The "?REDO FROM START" message which is never used!
CA2F: The code that would print that message, but is never called!
CA43: INPUT token processor
CA72: READ
CAFF: The "?EXTRA IGNORED" message
CB34: NEXT token processor

EXPRESSION EVALUATION

C88F: Evaluates the expression pointed to by HL
CC5F: Loads ACC (the F-PT Accumulator) with contents of a variable or array item
CC6F: Evaluates the Unary Functions (tokens >= AF hex)
CCA7: ?
CCF1: ? To do with string functions? Part at CD0D seems to do block string compare
CD4F: DIM token processor
CD53: Finds variable name in HL (sets up new one if it doesnt exist) and returns either the value or a pointer to it, depending on where it is called from
CFE9: FRE token processor
CF17: POS
CF1F: DEF
CF9F: STR$
D015: Prints a string pointed to by HL (see part 3)
D02F: Various string handling routines
D18B: LEN token processor
D19A: ASC
D1AB: CHR$
D1BB: LEFT$
D1EB: RIGHT$
D1F5: MID$
D225: VAL
D24A: INP
D256: OUT
D25C: WAIT
D29E: Serial input and output routines used by CLOAD & CSAVE (see part 3)
D2C9: CSAVE token processor
D310: Loads MWA tape header area
D341: CLOAD token processor
D39D: MATHS ROUTINES START HERE: I have not got very far into this area yet
---
D4AB: LOG token processor
D53F: ? F-PT Divide?
D666: SGN token processor
D61C: ABS
D635: Subroutines for moving F-PT values about (see part 3)
D6CA: INT token processor
D6FE: PEEK
D705: POKE
D7BB: outputs a number to the terminal (see part 3)
D8A3: A table of powers of 10 as 24 bit binary values
D8BA: SQR token processor
D908: EXP
D948: some sort of table for use of the above
D999: RND token processor
DA0E: COS
DA14: SIN
DA58: Another table
DA75: TAN token processor
DABA: ATN
DAB1: Table used by ATN
DAD2: END OF BASIC, FROM HERE ON IS UNUSED GARBAGE (with the exception of the warm and cold start jumps right at the top).
Next month I will describe some more-or-less useful routines that are hidden away in the Basic.

Ken Grimes (QSUG)

- CONT on 3/32

3 48

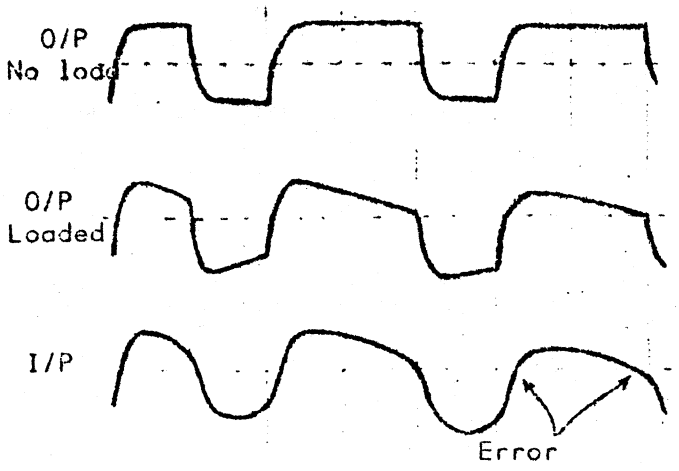


Fig 1.

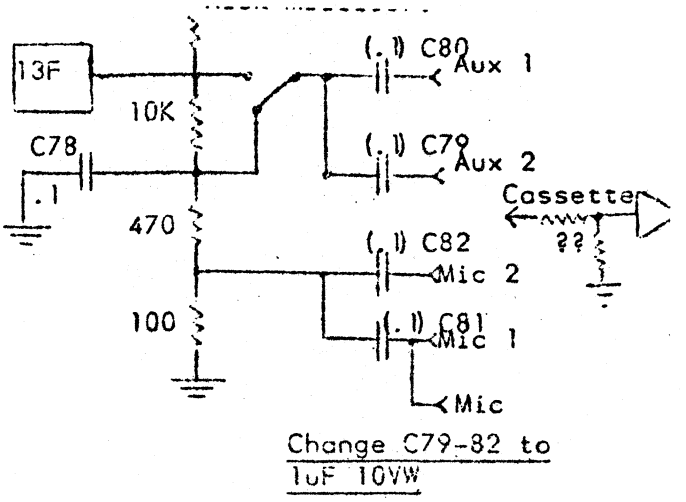


Fig 2.

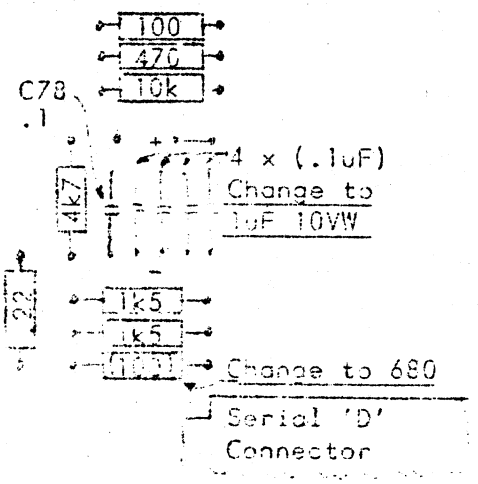


Fig 3

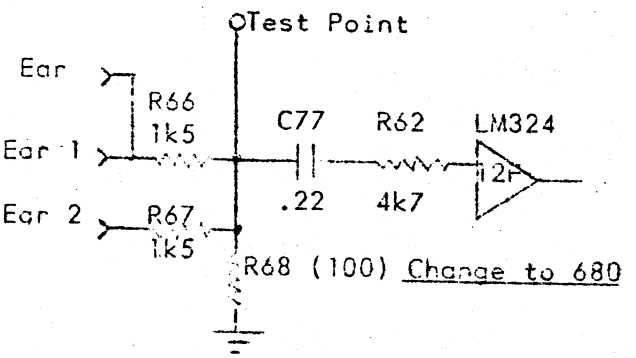


Fig 4.

If some Sorcerer wizards are still having trouble I would like to hear from them. I believe that further improvement could be obtained by fixing the input circuitry where the filters remove information at frequencies of interest. They also introduce phase shifts, which do cause trouble.

Rick Millicer, 199 Lower Heidelberg Rd., IVANHOE, Vic. 3079.

* * * * * * * * * *

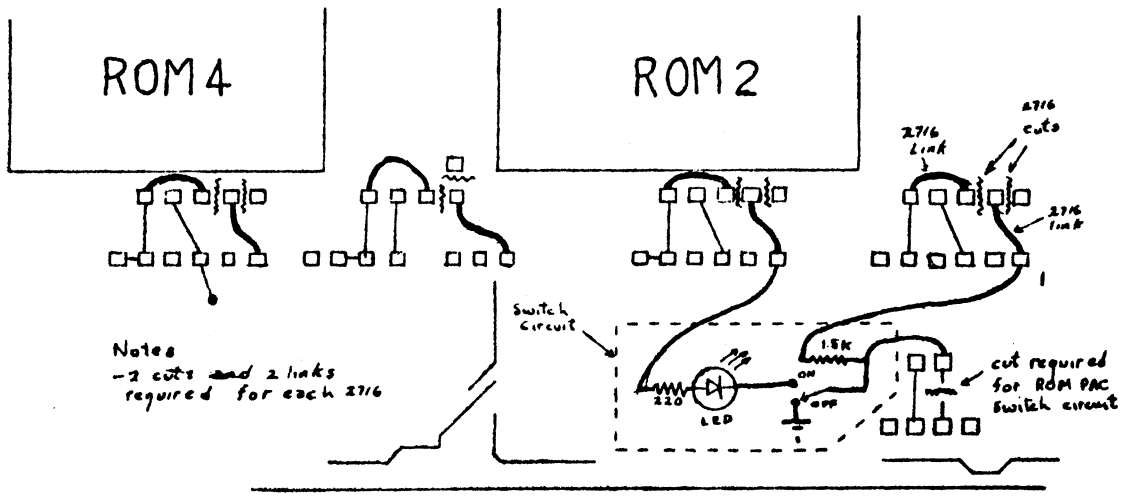
Wayne Gail (Melb) would like to contact anyone who has a Direct Memory Access (DMA) board for use in a S100 expansion unit. If you can help please ring on (03)2884310.

```
 '020C  B0        0150          OR      B           ;MAKE NEW VALUE
 '020D  12        0151          LD      (DE),A      ;PUT BACK IN MEMORY
*'020E  CD5301'   0152 INIT:     CALL    DISPLY      ;UPDATE SCREEN VALUES
 '0211  18BD      0153          JR      BYTE-$      ;GO BACK
 '0213  13        0155 NEXT1:   INC     DE          ;INCREMENT ADDRESS
*'0214  CD5301'   0156          CALL    DISPLY      ;UPDATE DISPLAY
 '0217  18B7      0157          JR      BYTE-$      ;GO BACK
 '0219  1B        0159 NEXT2:   DEC     DE          ;DECREMENT ADDRESS
*'021A  CD5301'   0160          CALL    DISPLY      ;UPDATE DISPLAY
 '021D  18B1      0161          JR      BYTE-$      ;GO BACK
 '021F  13        0163 EXIT:    INC     DE          ;INCREMENT ADDRESS
*'0220  C30601'   0164          JP      NEXT        ;BACK TO ADDRESS MODE
 '0223  1A        0166 SEARCH:  LD      A,(DE)      ;GET BYTE TO LOOK FOR
 '0224  47        0167          LD      B,A         ;SAVE IN B
 '0225  13        0168 UNTIL:   INC     DE          ;INCREMENT ADDRESS
 '0226  1A        0169          LD      A,(DE)      ;GET BYTE
 '0227  B8        0170          CP      B           ;DOES IT MATCH?
 '0228  20FB      0171          JR      NZ,UNTIL-$  ;LOOP BACK IF NO MATCH
*'022A  C30601'   0172          JP      NEXT        ;GO BACK IF IT DOES
*'022D  210601    0174 GO:      LD      HL,0100H    ;SET RETURN ADDRESS
 '0230  3190BF    0175          LD      SP,0BF90H   ;RESET STACK
 '0233  E5        0176          PUSH    HL          ;SAVE ADDRESS
 '0234  EB        0177          EX      DE,HL       ;GET ADDRESS
 '0235  E9        0178          JP      (HL)        ;JUMP TO IT
 '0236  C303E0    0180 OUT:     JP      0E603H      ;BACK TO EXIDY MON
 '0239            0182 STORE:   EQU     $           ;MODE STORE
```

Brian Dennis

SIMPLE ROM PAC MODIFICATIONS

I have found it convenient to be able to turn the ROM PAC off and on without having to turn the Sorcerer off. This can be accomplished by mounting a switch on the ROM PAC as illustrated below. A LED is used as an indicator to show if the PAC is on. Also illustrated are the jumpers options for using standard 5 volt EPROMs in the PAC.



- Ben Williams

GLW EPROM CARTRIDGE

The GLW EPROM cartridge is similar to Exidy's PROM Pac. It will accept most single +5V supply 2516/2716/2532 or 2732 EPROMs, providing either an 8K (2516/2716 EPROMs) or a 16K (2532/2732 EPROMs) cartridge.

A sketch of the component side of the printed circuit board giving the orientation of the EPROMs and the location of the various jumpers for configuring the cartridge to accept the different EPROMs is shown in Fig. 1. The broken lines shown in the sketch give the location of jumpers for changing the cartridge to 16K, while the solid lines are for an 8K cartridge.

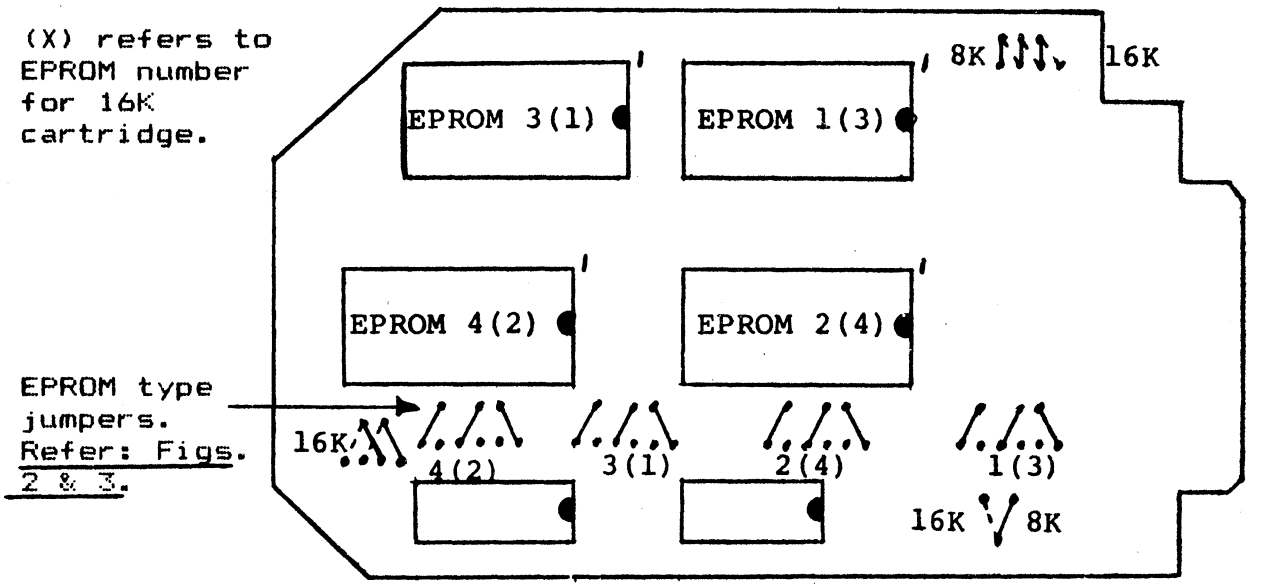
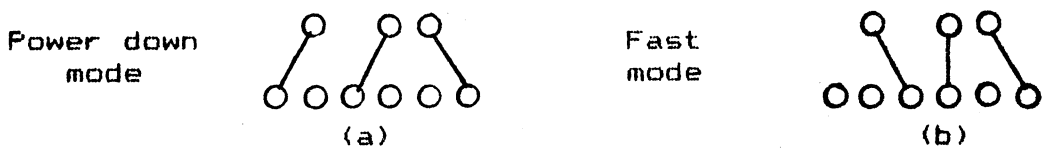


Fig.1. Sketch of component side of printed circuit board showing orientation of EPROMs and location of jumpers.

As received, the unmodified GLW EPROM cartridge is configured for 8K, accepting 2516/2716 EPROMs operating in the 'power down' mode. Mark 1 Sorcerer owners may find that some 450ns access time EPROMs will not work reliably in this mode, so the cartridge may be reconfigured so the EPROMs operate at maximum speed (non 'power down' mode). The configuration for the jumpers on the component side of the board for the 'fast' mode is shown in Fig. 2b.



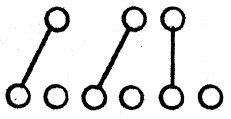
NB: Jumpers at each of the locations 1,2,3,4 in Fig. 1 will have to be changed.

Fig. 2. Possible jumper configurations for 2516/2716 EPROMs shown from the component side of the board.

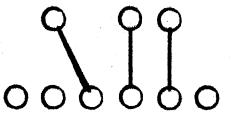
Note that the original jumpers must be cut before the new jumpers are installed. It is advisable to solder the jumpers with the EPROMs removed from the sockets and placed in protective-conductive foam or aluminium foil etc. When soldering the jumpers, a small pencil type soldering iron of low wattage should be used otherwise the printed circuit board may be damaged.

NB: BE CAREFUL WHEN INSTALLING EPROMS OTHERWISE THE IC SOCKETS ON THE BOARD AND/OR THE EPROM PINS MAY BE DAMAGED.

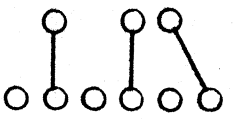
The jumper configurations for 2532/2732 EPROMs are given in Fig. 3.



Intel, Mitsubishi 2732 low power mode



Intel, Mitsubishi 2732 fast mode



Texas 2532

NB: Jumpers at each of the locations 1,2,3,4 in Fig.1 will have to be changed.

Fig.3. Possible jumper configurations for 2532/2732 EPROMs shown from the component side of the board.

If you have any problems or queries, you can contact me, Graham Wragg, on telephone No. (03)890-6918 between 6:00 and 10:00 PM EST most nights of the week.

SOME USEFUL (more-or-less) SUBROUTINES IN THE BASIC ROM PAC

I have not looked at all of the ROM PAC by any means. So far I have been concentrating mostly on the interactive areas and overall structure. Some of the printing routines listed here are done more effectively by the monitor, but if you mix Monitor and BASIC print routines, BASIC looses track of the cursor position and is likely to start a new line unexpectedly. So it is best to stick entirely to either MONITOR or BASIC calls.

CONVENTIONS: 'Bottom' and 'low' refer to the 0000 end of memory; 'top' and 'high' refer to the FFFF end of memory.

BLOCK-MOVE

Moves a block of data identified by pointers. A useful alternative to LDDR as it does not require the calculation of the length of the block.

ENTRY: C2E0 calls test for OM ERROR then...
C2E3 exchanges BC and HL then...
C2E6 moves source block pointed to by HL (high end) and DE (low end) to the destination pointed to by BC (high end), working from top to bottom (decrementing the pointers).

RETURNS with AF, BC, DE, and HL all modified.

FIND-LINE

Scans through the BASIC program looking for a line number (in binary format) held in DE.

ENTRY: C3FA searches from start of BASIC program (01D5)
C3FD searches from position pointed to by HL

RETURNS with flags and pointers as follows:

Z & C flags if line number found; BC points to the found line and HL points to the following line.

Z & NC flags if it reached the end of the program without finding one exceeding the value of the line number; BC & HL both point to the end of the program.

NZ & NC flags if reached a higher line number without finding the number in DE; BC points to the first higher line, and HL to the following one.
AF is modified and DE unaltered in all cases.

INPUT-LINE-BUFFER

Inputs a line from the keyboard and stores it in the BASIC line buffer at 014C. It handles RUB, Ctrl-C and @ functions. It includes a test at C564 to see if it has exceeded the buffer limit but does not branch out if it has! This is the bug which kills BASIC if you enter over-long lines. On receipt of a CR it puts a null at the end of the entered string, resets HL to the start of the buffer (ready for scanning by the caller), sends a CRLF to the terminal and returns.

ENTRY: C53A
RETURNS with AF, BC, & HL modified, but DE unaltered.

MAKE-BINARY-INTEGER

Converts an ASCII string of decimal digits pointed to by HL into an unsigned 16 bit binary integer which is returned in DE. It ignores leading spaces and returns with HL pointing to the first non-digit in the string and A holding this character. If the binary value exceeds 64K it jumps to the SN ERROR message. Would often be called as a follow-up to INPUT-LINE-BUFFER

ENTRY: C7EA decrements HL then...
C7EB scans from current value of HL.

RETURNS with AF, DE, & HL altered, but BC preserved.

SKIP-TO-EOS/EOL

Skips to the end of the statement or to the end of the line pointed to by HL. That is, it scans the text pointed to by HL until it finds either a ":" or a null, but ignores any ":"s within quote-bounded strings. Returns with HL pointing to the terminating character, and with that character (: or null) in A. It will substitute any other search character for the ":" if you load this into C and enter at C8B9.

ENTRY: C8B5 skips to a null or to the first ":" outside quotes
C8B7 searches only for a null, ie it skips to the EOL marker
C8B9 skips to the null or to the first occurrence of the char in C which is not within quotes.

RETURNS with AF, BC, & HL altered but DE unchanged.

NEWLINE?

Checks whether the terminal is at the start of a new line and if so returns with Z flag and 0 in A. If not it jumps to MAKE-NEW-LINE (see below).

ENTRY: C9B2
RETURNS with AF altered and other registers preserved (unless it jumps - see below).

MAKE-NEW-LINE

Makes a new terminal line by sending a CRLF and nulls, and resets the print-column counter to zero.

ENTRY: C9BF
RETURNS with AF & HL modified, BC & DE unaltered.

CP-HL-DE

A very useful subroutine which compares the 16 bit contents of the HL and DE registers and sets the flags as follows:

Z if HL=DE
NZ & C if HL<DE
NZ & NC if HL>=DE

ENTRY: C574

RETURNS with AF modified, all others preserved.

PRINT-CHAR

Prints the character in A. It tests the Suppress Output flag (0144) first and returns without printing if this is set. The character is sent via the monitors CHROUT routine at E00C, If it is not a control char it increments the print-column counter and calls MAKE-NEW-LINE if needed.

ENTRY: C585 with char in A

RETURNS with all registers preserved.

GETKEY

Calls the monitor CHRIN routine at E009 untill a key is pressed and released. Checks if Ctrl-O and if so disables keyboard and returns with a null in A, otherwise A has the character from the keyboard.

ENTRY: C5R4

RETURNS with AF altered, BC,DE,HL preserved.

SCAN-(HL)

Scans the text pointed to by HL, skipping all spaces and returning with HL pointing to the first non-space, with that character in A and with the flags set as follows:

Z if char is a null.
NZ & C if the char is a digit (0..9).
NZ & NC for any other characters.

ENTRY: C6CD increments HL then...
C6CE scans from current value of HL.

RETURNS with AF & HL altered, BC & DE unchanged.

ALPHACAPS?

Returns with NC if char is A..Z, otherwise with C.

ENTRY: C7BC with HL pointing to char, or...
C7BD with char in A

RETURNS with AF modified (only F if entered at C7BD), all others OK.

PRINTSTRING

Prints the string pointed to by HL. A longer and more complicated routine than the Monitors MSGOUT, but keeps track of the BASIC print-column counter.

ENTRY: D014 increments HL (past a quote?), then...
D015 prints strin pointed to by HL and terminated by a quote or a null
D01E prints a string of length held in E, and pointed to by HL.

RETURNS with all registers altered (except possibly D?).

SERIAL-OUT Routines

These three routines are used by CSAVE*. The code is a bit odd, with unnecessary register moves and PUSH/POPs. The innermost routine (D2BD) waits for the handshake flag (bit 0 of port FD) to be set by the UART then sends the char in A to port FC.

ENTRY: D2A1 does the D2A4 routine twice (ie it sends the char out twice)
D2A4 fiddles about uselessly then calls D2BD
D2BD does the actual work

RETURNS with all registers intact.

SERIAL-IN

Waits on bit 1 of port FD to be set by the UART, then loads A from port FC.

ENTRY: D29E simply Jumps to...
D2A0 calls QCKCHK and Jumps to UF ERROR if Ctrl-C or ESC keys pressed, otherwise...
D2B3 the main input routine.

RETURNS with AF modified, all others preserved.

SKIP-CR

Scans the text pointed to by HL until it finds a CR which it converts to a null. Returns with HL pointing to the char befor the CR/NULL.

ENTRY: D304 POPs HL (NB, if you CALLed it this will be your return address!), then...
D305 INC HL then...
D306 scan from current (HL).

RETURN with AF & HL changed, BC & DE unaltered.

FLOATING POINT MOVEMENTS

I have not looked at the maths routines in any detail yet, however the followins are a number of useful routines for moving four byte floating point numbers. They could also be used for any four byte string of data. ACC is the floating point accumulator (01BF..01C2) which holds, from low to high: Least Significant Byte (LSB), Intermediate Byte (IB), Most Significant Byte (MSB) with the sign as Bit 8, and Exponent (Exp). The same order is used in the variable space storage. During transfers and operations these are stored in BC & DE as follows: B=Exp, C=MSB, D=IB, E=LSB.

The routines could be regarded as MACROs with the following Zilog-like Mnemonics:

D635 = LD ACC (HL); (via BCDE)
D638 = LD ACC BCDE
D643 = LD BCDE ACC
D646 = LD BCDE (HL)
D64F = LD (HL) ACC; (direct, NOT via BCDE)
D652 = LD (HL) (DE); (" " ")
D6C3 = DEC BCDE