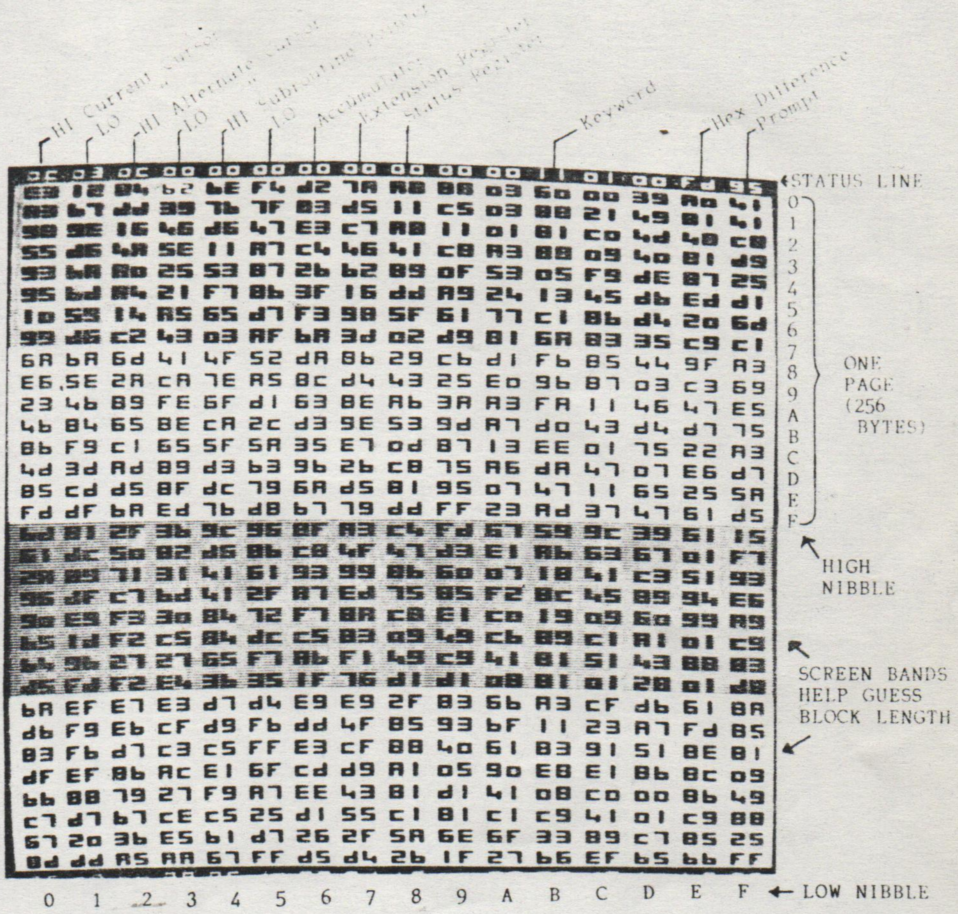


```
oc o3 oc oo oo oo oo oo oo oo oo 11 o1 oo Fd 95
E3 12 84 b2 bE F4 d2 7A A8 86 o3 6o oo 39 Ao 41
A3 b7 dd 39 7b 7F 83 d5 11 c5 o3 88 21 49 81 41
98 9E 16 46 d6 47 E3 c7 A8 11 o1 81 co 4d 40 c8
55 d6 4A 5E 11 A7 c4 46 41 c8 A3 88 o9 4o 81 d9
93 bA Ao 25 53 87 2b b2 89 oF 53 o5 F9 dE 87 25
95 bd A4 21 F7 8b 3F 16 dd A9 24 13 45 db Ed d1
1o 55 14 A5 65 d7 F3 98 5F 61 77 c1 8b d4 2o 6d
99 d6 c2 43 o3 AF bA 3d o2 d9 81 6A 83 35 c9 c1
6A bA 6d 41 4F 52 dA 8b 29 cb d1 Fb 85 44 9F A3
E6,5E 2A cA 7E A5 8c d4 43 25 Eo 9b 87 o3 c3 69
23 4b 89 FE 6F d1 63 8E Ab 3A A3 FA 11 46 47 E5
4b 84 65 8E cA 2c d3 9E 53 9d A7 do 43 d4 d7 75
8b F9 c1 65 5F 5A 35 E7 od 87 13 EE o1 75 22 A3
4d 3d Ad 89 d3 b3 9b 2b c8 75 A6 dA 47 o7 E6 d7
85 cd d5 8F dc 79 6A d5 81 95 o7 47 11 65 25 5A
Fd dF bA Ed 7b d8 b7 79 dd FF 23 Ad 37 47 61 d5
bd 81 2F 3b 9c 96 8F A3 c4 Fd 67 59 9c 39 61 15
61 dc 5o 82 d6 8b c0 4F 47 d3 E1 Ab 63 67 o1 F7
2A 89 71 31 41 61 93 99 8b 6o o7 18 41 c3 51 93
96 dF c7 bd 41 2F 87 Ed 75 85 F2 8c 45 89 94 E6
9o E9 F3 3o 84 72 F7 8A c8 E1 co 19 o9 6o 99 A9
b5 1d F2 c5 84 dc c5 83 o9 49 cb 89 c1 A1 o1 c9
b4 9b 27 27 65 F7 Ab F1 49 c9 41 81 51 43 88 83
d5 Fd F2 E4 3b 35 1F 76 d1 d1 o8 81 o1 28 o1 d8
bA EF E7 E3 d7 d4 E9 E9 2F 83 6b A3 cF db 61 8A
db F9 Eb cF d9 Fb dd 4F 85 93 bF 11 23 A7 Fd 85
83 Fb d7 c3 c5 FF E3 cF 88 4o 61 83 91 51 8E 81
dF EF 8b Ac E1 6F cd d9 A1 o5 9o E8 E1 8b 8c o9
bb 88 79 27 F9 A7 EE 43 81 d1 41 o8 co oo 8b 49
c7 d7 b7 cE c5 25 d1 55 c1 81 c1 c9 41 o1 c9 88
67 2o 3b E5 b1 d7 26 2F 5A 6E 6F 33 89 c7 85 25
8d dd A5 AA 67 FF d5 d4 2b 1F 27 b6 EF b5 bb FF
```

UNDERSTANDING THE TV PICTURE

The main area of the screen contains 512 bytes ($\frac{1}{2}$K) of RAM information - two PAGES of 256 bytes - arranged in hexadecimal digit pairs which give the data contents of memory address locations. The address of each location is found from its position in the screen map - the low nibble is found from the lateral displacement and the high nibble from the vertical displacement. There are SIXTEEN PAGES (4K) - EIGHT DIFFERENT SCREENS FULL - of data to be viewed.

SHADED BANDS

The shaded bands divide the screen area into blocks of 128 bytes (80 in Hex). Most MPUs have single byte branches, jumps or memory-reference instructions to locations which are plus or minus 128 bytes from the program counter. These bands help when estimating whether a location is within range.

SETTING-UP PROCEDURE

Ensure that there is no device in the ZIF SOCKET. Plug the power-unit into SOFTY, DIN PLUG groove uppermost. Connect SOFTY to a TV receiver using the lead supplied. Plug the power-unit into the mains supply. Press RESET. Tune a picture in channel 36. If the picture is not sharp you are tuned to a harmonic or the lead is broken.

SOFTY's picture is to the left of the screen. There is a lot of information presented and the height and width controls of the TV receiver may need adjusting to view it all. Set up the picture so that at least one line of inverted video (White characters on black) is visible as you see here. A TV set which will display SOFTY's picture can be used for normal viewing without readjustment.

DEFINITIONS

THE STATUS LINE is the line of inverted-video immediately above the main screen contents which displays information essential to the SOFTY USER. Inverted video means white characters on black background.

SCRATCHPAD means the 128 bytes of RAM which do not form part of the main RAM BUFFER. SCRATCHPAD is used to store information during the performance of SOFTY's functions. SCRATCHPAD is, in fact, seen as inverted video at top and bottom of the screen and the STATUS LINE is the last 16 locations (70 to 7F) of SCRATCHPAD.

RAM BUFFER means the 2Kx8 of RAM available to hold the user's programs (which may be extended to 4K by placing a BYTEWIDE RAM in the ZIF SOCKET).

THE ZIF SOCKET means the Zero-Insertion-Force socket, which has a release lever, used to hold the EPROM when programming.

THE DIL SWITCH means the DUAL-IN-LINE PERSONALITY SWITCH which is used to change the connections to the ZIF to suit the device present.

J1 AND J2 are two 3.5mm JACK SOCKETS which are used to send and receive signals for the CASSETTE TAPE INTERFACE. The INPUT JACK is nearest the TV output and should be connected to the EAR OUTPUT of the cassette recorder. The OUTPUT JACK should be connected to AUX INPUT of the recorder.

THE DIN SOCKET is used to take the unregulated supply voltages (Nominally +8 volts for Vcc and +30 volts for Vpp) from the MAINS POWER PACK into SOFTY.

THE FLYLEAD is the COAXIAL LEAD with a TV PLUG on one end and a PHONO PLUG on the other used to take the modulated UHF TV signal to the TV receiver. (The phono plug with the long centre goes into SOFTY).

JA is the 26 PIN PLUG on the right which is used for the ROMULATOR LINK.

JB is the similar 20 pin plug which may be fitted on the left above the DIL SWITCH. JB is used to connect SOFTY to EXTERNAL DEVICES. Two eight bit PARALLEL PORTS, SERIAL INPUT, SERIAL OUTPUT, +5 VOLTS and GROUND are connected to JB.

HEXADECIMAL NUMBERS

You cannot do serious work with micros if you do not know the HEX equivalent of a BINARY number. You can calculate 8's+4's+2's+1's but it is better to learn them as identities. Instructions and data of eight bit micros are now invariably expressed as HEX digit-pairs.

HEX	to	BINARY
0	=	0000
1	=	0001
2	=	0010
3	=	0011
4	=	0100
5	=	0101
6	=	0110
7	=	0111
8	=	1000
9	=	1001
A	=	1010
B	=	1011
C	=	1100
D	=	1101
E	=	1110
F	=	1111

KEYBOARD FUNCTIONS

RESET will interrupt the processor from whatever program is running and reset all internal registers to zero. Then the program restarts at the bottom of the monitor, clears spurious cursor highlights and puts the cursor to start of RAM BUFFER which is the top-left location of page 8. RESET dqes not alter any locations in the RAM BUFFER. RESET should be used after switching-on.

SHIFT gives access to a second function for most of the keys. When the SHIFT key has been pressed and SOFTY is waiting for a second-function keystroke the PROMPT location contains 55 (Which looks a little like SS for SHIFT). Pressing the SHIFT key twice will tidy-up spurious CURSORS from previous operations without changing anything else. **In this text the dollar sign $ is used to mean a shifted keystroke.**

HEXADECIMAL ENTRY is made by simply pressing the appropriate keys in sequence.

CURSOR keys move the CURSOR LEFT and RIGHT. When either of these keys is held down the CURSOR first STEPS automatically for one full line: then HOPS a line at a time until the key is released. If $ is pressed first the CURSOR moves directly up or down without stepping first. A little practice brings easy control of the CURSOR. **The actual ADDRESS of the cursor is shown in the first two locations of the STATUS LINE.** This always starts with a 1 which may be ignored. Pages 0 to 7 are EPROM (or whatever is in the ZIF). Pages 8 to F are RAM BUFFER. If you move the CURSOR around you will see that the CURSOR ADDRESS changes: you can tell the absolute location of the CURSOR by looking at this CURSOR ADDRESS.

The PAGE key permits a direct jump between pages which can save a lot of CURSOR movement. Press PAGE, which puts CC in the PROMPT, then press whatever PAGE you want, a single digit, 0 to F.

$FIX provides an ALTERNATE CURSOR function. When FIX is pressed the CURRENT CURSOR is stored in locations 3 and 4 of the STATUS LINE. Thereafter the DIFFERENCE IN HEX between the CURSOR ADDRESSES is shown in the penultimate location of the STATUS LINE, next to the PROMPT LOCATION.

$SWAP exchanges the CURRENT CURSOR with the ALTERNATE CURSOR. This COMPLEMENTS (subtracts from zero) the HEX DIFFERENCE as shown on-screen. One purpose of the two CURSORS is to permit calculation of displacements for jumps or branches which are program-counter-relative. (If the program-counter is incremented by the instruction you will need ONE LESS than the actual HEX DIFFERENCE between source and destination).

INPUT and OUTPUT ROUTINES are available. These are covered separately.

$RUN transfers program-control to the ZIF SOCKET. The INS8060 microprocessor gets its next instruction at the first location of whatever device is in the ZIF.

$COPYFIRM copies SOFTY's own FIRMWARE MONITOR into the RAM BUFFER.

$COMPARE examines every location of the device in the ZIF and COMPARES the corresponding byte in the RAM BUFFER. Locations which differ are HIGHLIGHTED (on the current page) and the total of all discrepancies is entered in the PROMPT location in HEX.

$BURN writes the contents of the RAM BUFFER into EPROM. After the BURN function is finished the COMPARE function is called automatically: if the PROMPT location contains zero then the burn is successful. At 50 ms. per location the theoretical time for programming a 2K EPROM is 102 seconds. However locations which are clear (contain FF) do not need a program pulse and these are skipped. So the real time for a burn depends on the size of the program.

$COPY transfers the contents of the ZIF device into RAM BUFFER. This operation takes less than a second. All SOFTY's functions apart from programming and data-transfers are almost instantaneous.

$PRETEST tells you whether your program will go into the EPROM. EPROMS are programmed HIGH-TO-LOW. The only way to go from LOW-TO-HIGH is the ERASE the device completely with SHORT-WAVELENGTH-ULTRAVIOLET. The PRETEST function compares every byte to see whether it is capable of being programmed: those locations which "won't go" are highlighted and counted as in the COMPARE function. This function was not available on the original SOFTY: the inclusion is based on experience. It's value is to check out an EPROM for erasure without losing the program in RAM BUFFER. It is also useful if you are trying to modify an EPROM without erasing it - the mods could be in the right direction.

$EX-RAM copies the contents of the RAM BUFFER into a BYTEWIDE RAM in the ZIF socket. These RAMs have similar pin-out to the EPROM and it is only necessary to connect WRITE STROBE and CHIP SELECT to have a useful extension to 4K of RAM BUFFER.

$MATCH is a function which will HIGHLIGHT all occurrences of any byte. MATCH enters BB (for byte) in the PROMPT. The following two keystrokes are taken as the byte and all locations which contain the byte are highlighted.

$CLEAR FORWARD OR BACK will write all high bits (FF) into RAM BUFFER in front of or behind the CURSOR. The reason why the CLEAR function gives all-high is that an unprogrammed EPROM is all high.

INSERT looks forwards to the first occurrence of three unused bytes (FF FF FF) and shifts code rightwards to give you room to insert another instruction. Most programmers would not permit the correction of faults so readily!

DELETE is the opposite of INSERT - the first three bytes clear (FF FF FF) are found and the code is shifted leftwards to remove the instruction under the CURSOR. Both the INSERT and DELETE instructions will shift as much of of the 2K as is necessary but neither will work if there is no free space.

BLOCK FUNCTIONS

SOFTY has a set of functions for handling code in BLOCKS which may be called by a single keystroke.

DEFINE puts DD in the PROMPT. The cursor may then be moved BACKWARDS to enclose the BLOCK of data.

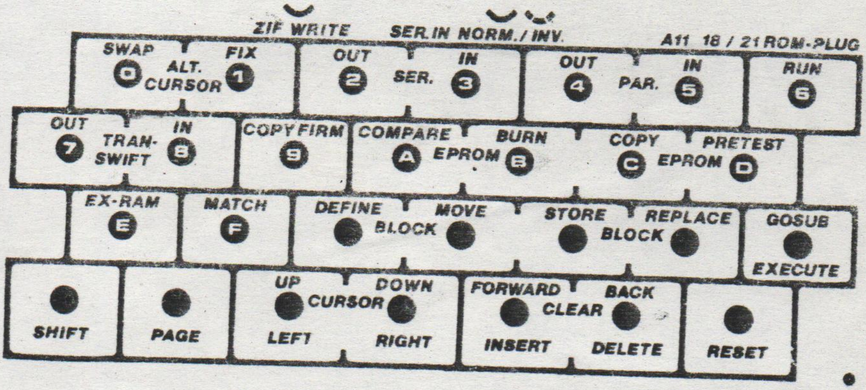
MOVE puts FE in the PROMPT. The CURSOR keys will then MOVE the defined block through RAM. Intervening data is shifted to the other side of the block. Nothing is erased: the code is only REARRANGED. The size-limit on a block for shifting is from a single byte to 127 bytes (one less than a screen "band"). It is possible to look directly at EPROM on-screen without first copying it into RAM BUFFER - remember that you cannot MOVE code unless it is in RAM.

STORE stores the BLOCK in SCRATCHPAD. There is only room to store 110 bytes (7 complete lines all but two bytes).

REPLACE brings back the BLOCK after STORE. The BLOCK will be written to RAM at the location of the CURSOR and the cursor will be transferred to the end of it. This provides a useful facility for picking-up code from one place (even EPROM) and putting it in another. If there is a particular section of code which is frequently used in your program (a MACRO) you can store it as a BLOCK and insert it with a single keystroke. If you need to fill the RAM BUFFER with some repetitive code this function provides a quick method of doing it.

It is possible to call the block as a SUBROUTINE whilst it is in SCRATCHPAD using the $GOSUB function. This provides a method of writing your own KEYFUNCTION(S) - see example later.

TO GET OUT OF BLOCK MODE USE THE $ KEY.



EMULATION:- A ROM SUBSTITUTE.

SOFTY has a ribbon-cable lead terminated in a 24 pin plug which may be inserted into a system as ROM. The white arrow indicates pin 1. The normal chip-select is a negative going signal to pin 20 of the ROM PLUG. It is possible that a masked ROM may have multiple chip-selects, both positive and negative going. Pins 18 and 21 are also brought through the cable and these connections are exposed above the right side of the keyboard. Address line 11 may be connected to either pin 18 or pin 21 - standards vary.

If A/11 is connected then the external system will see whichever device is in the ZIF as the lower 2K of the 4K ROM: otherwise it sees only the RAM BUFFER as 2K of Read-Only Memory. The lower 2K could be a 2K BYTEWIDE RAM, a 2716 or half a 2532 or 2732. If there are multiple chip-selects then it will be necessary to add a gate (a single 74LS00 is usually adequate) to decode these into a single negative going chip-select. The line from pin 20 can be cut beneath the board (This track is marked CS in two places near JA). The most effective way to add a decoder gate into a system is to bend all its pins horizontal except 7 and 14. Then you solder it over another chip to pick up Vcc and Ground. The connections are wire-wrapped or soldered to the flying pins.

The DATA, ADDRESS and CHIP-SELECT lines to the ROMULATOR LINK are buffered by LS TRI-STATE BUFFERS. The inputs present ONE LS LOAD to the external system. It is not possible for the external system to corrupt the program. SOFTY has priority of access: it is possible to change data in any address location via the keyboard without disconnecting the ROMULATOR LINK.

ACCESS TIME OF ROMULATOR

You cannot expect too much speed. Fast memory systems require careful board layout: stuffing signals down ribbon-cable is the antithesis. This speed problem is one of interconnection - it is not inherent to SOFTY. Any system which emulates the processor or memory will have this same problem no matter what it cost. There are, however, some things which can be done to help:

The chip-select lines can be removed from the cable and run separately. This prevents the worst of the cross-talk. (This access-time problem has nothing to do with propagation-delays in the cable. The cable only adds a couple of nanoseconds to memory access time. The buffers maybe add 80 nanoseconds more

You could use faster 2114's, nominally 450 nanoseconds as supplied.

The ribbon-cable could be shortened or replaced with a more elaborate type which has interwoven ground-screens. (We are looking for a source of supply). Make a good ground connection between the two systems - don't rely on the ground which passes through the cable.

Standard SOFTY access-time will be somewhat better than a microsecond. The "Tuning" techniques outlined here will improve this considerably.

INPUTS AND OUTPUTS.

SOFTY has three different methods of transmitting and receiving data.

1) Bit-serial input and output routines at 110, 300, 600, 1200 and 2400 baud.

2) Standard parallel input and output routines with handshake.

3) TRANSWIFT input and output routines for easy transfer of data between microsystems and cassette tape for storage.

In the case of the input routines it is assumed that the transmitted data is in ASCII and each word received is automatically translated into hexadecimal. Any character which is not hexadecimal (i.e. not ASCII 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E or F) is ignored. It does not matter if your computer transmits NULLS, CARRIAGE RETURNS, LINE FEEDS, SPACES etc. because SOFTY will ignore them. This makes it easier to write a program to transmit the code product of an assembler to SOFTY.

When SOFTY is transmitting, however, it is assumed that the receiving device is a printer.Each byte of code is transmitted as two ASCII characters with spaces between bytes. Line feeds and carriage returns are also transmitted, with extra line spacing between each 256 byte page. The output of the printer looks similar to the presentation of data in screen format.

PARALLEL INTERFACES.

THE INS8154 RAM I/O has the ability to strobe data in and out via PORT A independent of processor attention. The signals which perform the "handshake" are the top two bits of PORT B.

All bits of both ports are connected to JB. Via JB SOFTY will perform parallel data transfers with systems which are similarly equipped.

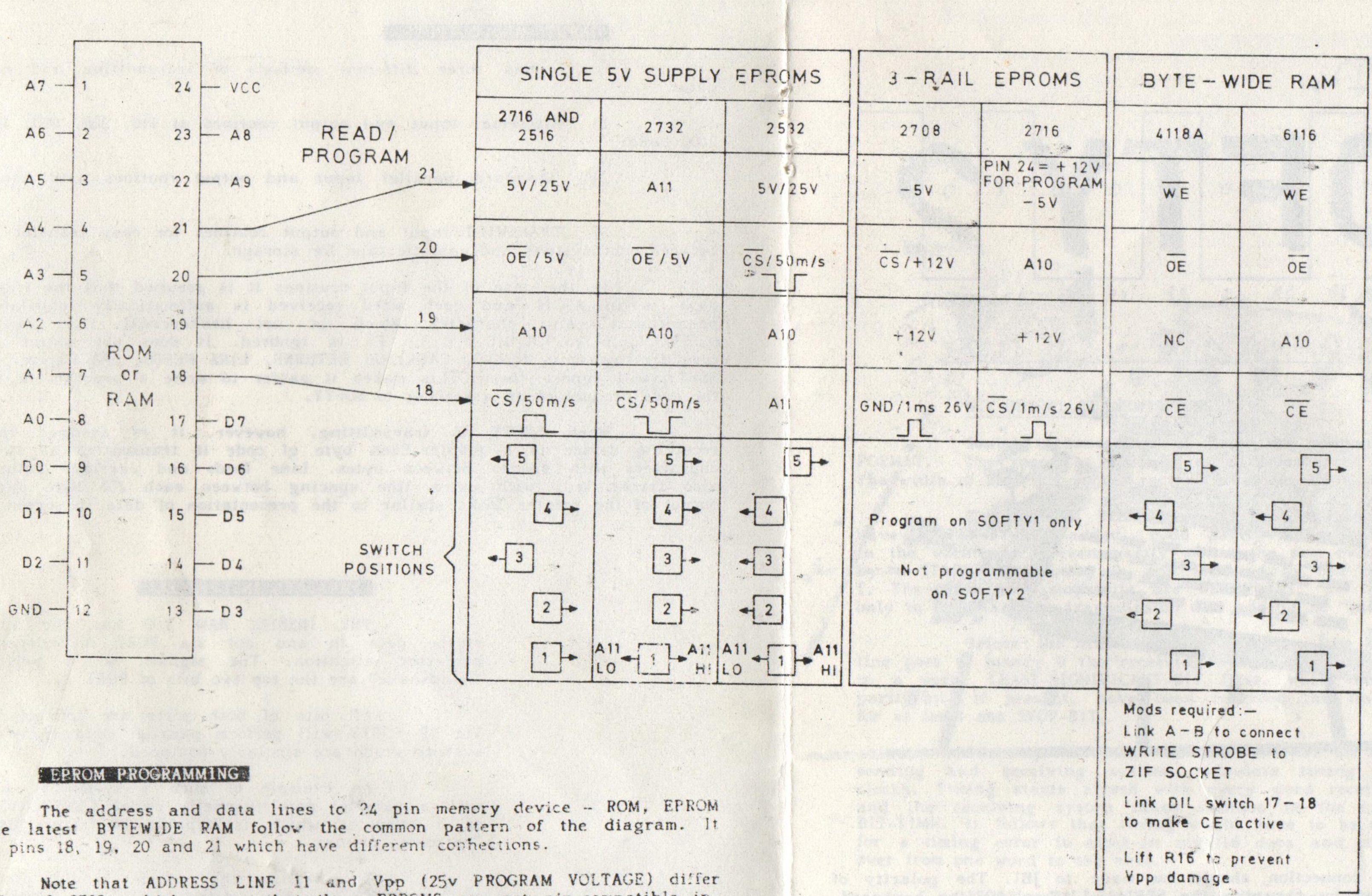
An example of such a system is a printer with a parallel port (normally called CENTRONICS TYPE) or a computer which is capable of transmitting to a Centronics printer.

The transmitting device should place data on the lowest seven data bits, the eighth being unimportant in an ASCII transmission, and then apply a negative going STROBE signal to port B7 When data is in the input buffer a high-going signal, BUSY, is output on port B6 After the data has been read from the buffer BUSY will go low again, which indicates tto the transmitting device that it is ready for more data.

Output signals use the same pins: data is placed on the lower 7 bits of PORT A, and STROBE is issued on Port B6 The answering ACK signal should go to port B7

SER.IN (CTS)	1	20	+5volts
B0	2	19	SER.OUT
B1	3	18	A7
B2	4	17	A6
B3	5	16	A5
B4	6	15	A4
B5	7	14	A3
B6	8	13	A2
B7	9	12	A1
GND	10	11	A0

JB CONNECTOR



	SINGLE 5V SUPPLY EPROMS			3-RAIL EPROMS		BYTE-WIDE RAM	
	2716 AND 2516	2732	2532	2708	2716	4118A	6116
21	5V/25V	A11	5V/25V	-5V	PIN 24 = +12V FOR PROGRAM -5V	$\overline{\text{WE}}$	$\overline{\text{WE}}$
20	$\overline{\text{OE}}$/5V	$\overline{\text{OE}}$/5V	$\overline{\text{CS}}$/50m/s	$\overline{\text{CS}}$/+12V	A10	$\overline{\text{OE}}$	$\overline{\text{OE}}$
19	A10	A10	A10	+12V	+12V	NC	A10
18	$\overline{\text{CS}}$/50m/s	$\overline{\text{CS}}$/50m/s	A11	GND/1ms 26V	$\overline{\text{CS}}$/1m/s 26V	$\overline{\text{CE}}$	$\overline{\text{CE}}$
SWITCH POSITIONS	← 5	5 →	5 →	Program on SOFTY1 only Not programmable on SOFTY2		5 →	5 →
	4 →	4 →	← 4			← 4	← 4
	← 3	3 →	← 3			3 →	3 →
	2 →	2 →	← 2			← 2	← 2
	1 →	A11 LO ← 1 → A11 HI	A11 LO ← 1 → A11 HI			1 →	1 →

Mods required:-
Link A-B to connect WRITE STROBE to ZIF SOCKET

Link DIL switch 17-18 to make CE active

Lift R16 to prevent Vpp damage

EPROM PROGRAMMING

The address and data lines to 24 pin memory device - ROM, EPROM and the latest BYTEWIDE RAM follow the common pattern of the diagram. It is only pins 18, 19, 20 and 21 which have different connections.

Note that ADDRESS LINE 11 and Vpp (25v PROGRAM VOLTAGE) differ on 2732 and 2532, which means that these EPROMS are not pin-compatible in-circuit or when programming.

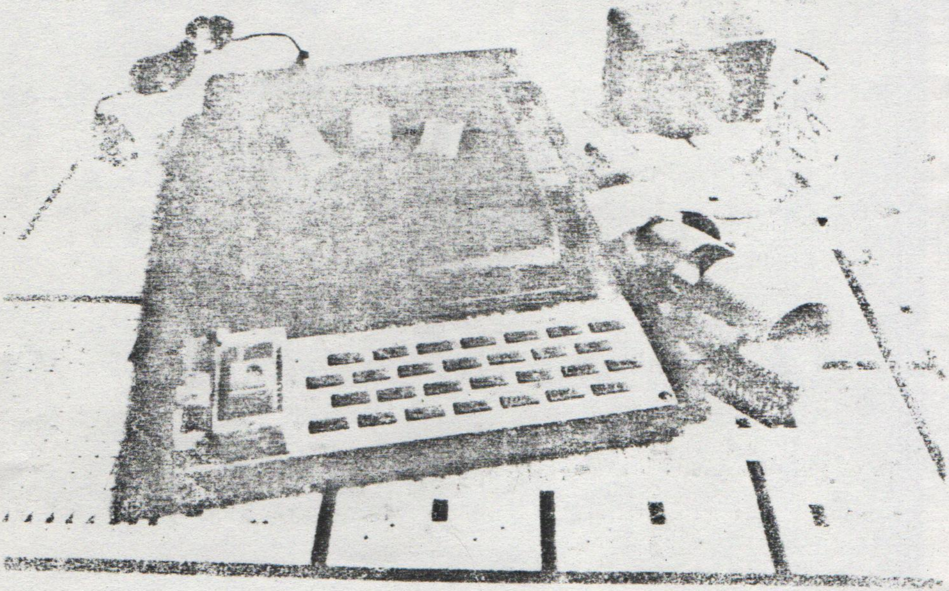
Set the switches to the correct position for the type of EPROM to be used. The EPROM will be destroyed if programming is attempted with the switches in the wrong position. Never operate SOFTY with the switches set to any other pattern than one of those detailed above - it is possible to damage the internal circuitry by failing to observe this precaution.

To program an EPROM with the contents of the RAM buffer press SHIFT (Observe the prompt 55) then BURN. SOFTY will then apply 25 volts to Vpp, and the correct data to each address in turn for 50 milliseconds whilst applying a program pulse. A 4K EPROM is copied in two halves, selected by the DIL switch 1 (LEFT SELECTS LOWER HALF).

DISMANTLING INSTRUCTIONS

SOFTY is held together by EIGHT PLASTIC RIVETS: FOUR in the CASE FLANGES and FOUR holding the PCB to the BASE. Push the centres right through and they will come apart easily.

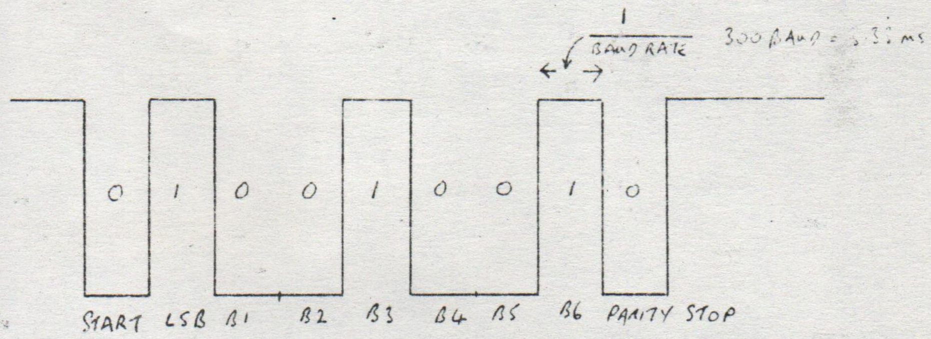
SOFTY 2



SERIAL INPUT connection should be made to JB1. The polarity of the input may be reversed by changing the SERIAL LINK above the keyboard. (Cut the track between C and D, then make D to E to invert). RV1 may be adjusted near centre of travel..

SERIAL OUTPUT may be taken from JB19, but if voltage level shifting or inversion are required external components and supplies must be added as shown in the diagram. CLEAR-TO-SEND may be input on JB1 and the polarity reversed by the SERIAL LINK as required. If not connected then RV1 should be adjusted to whichever extreme of travel gives an output.

When SERIAL IN or SERIAL OUT have been pressed the PROMPT will contain BA (for BAUD?): a single digit should be input: 1=110, 3=300, 6=600, C=1200 and F=2400.



ASCII "I", ODD PARITY. (49 HEX)

BIT-SERIAL TRANSMISSIONS.

Serial transmissions of data are commonly made in ASYNCHRONOUS FORMAT. The speed at which data is transferred is called the BAUD-RATE. The width of each bit is the reciprocal of the baud-rate in seconds.

The data is broken into "words" of seven or eight bits and may have a PARITY-BIT added. (Odd parity makes the sum of the binary 1's in the word odd - even parity makes the sum even). Each word is preceded by a START-BIT, binary 0, and followed by one or more STOP-BITS, binary 1. The start and stop bits are REDUNDANT, as is the parity bit: present only to facilitate the transmission and adding no "intelligence".

Before the transmission starts the line is at binary 1. When the line goes to binary 0 the receiving system recognises a START-BIT and clocks-in a word, LEAST-SIGNIFICANT-BIT first. When the word and the following parity-bit, if present, have been received the line goes back to binary 1 for at least one STOP-BIT.

One important fact to note is that NO CLOCK is transmitted: the sending and receiving systems calculate timing from their own internal clocks. Timing starts afresh with every word received, from the START-BIT, and the receiving system takes samples in the calculated centres of each BIT-TIME. It follows that timing would have to be more than 8% or so adrift for a timing error to clock-in invalid data and timing errors do not carry over from one word to the next.

ASCII code requires seven bits but most small computer systems transmit eight.

The term RS232 is used loosely to refer to transmissions as outlined. There is no real RS232 standard which is observed by all manufacturers. The system runs from + and - 12 volt rails but + and - 8 volts are accepted as binary 0 and binary 1. Many "RS232" systems will accept logic levels of 0.4volts and 2.5 volts. MARKING (binary 1) is normally low and SPACING (binary 0) is high. The polarity of the transmission is the opposite of what you might expect, but inverters in the sending and receiving systems usually take care of that. The receiver usually has a signal CLEAR-TO-SEND (Active when 0) to control the flow of information from the transmitter.

TRANSWIFT, THE CASSETTE INTERFACE

In order to leave the serial lines of the processor free for other forms of data transmission a method of storing data on tape has been devised which is entirely software. This novel method of data transmission may also be used for data interchange between any two microsystems - all that is required is a single bit of I/O somewhere in addressing space.

In effect TRANSWIFT sends a binary 0 as a single cycle of one frequency and a binary 1 as a cycle of a different frequency. The transmission may be passed through a capacitor because it has no dc component and still be recovered when the waveform has suffered degradation-- as it would after being passed through the audio-circuits of a tape recorder. Recovery of data is performed by squaring the waveform at its points of transit across some intermediate voltage level. This waveform is sampled by a known STEP interval from positive or negative transitions to determine whether a zero or a one was transmitted. In elaborate versions of TRANSWIFT this STEP INTERVAL may be calculated from the transmission itself.

TRANSWIFT solves many problems with regard to data transmissions in limited bandwidth media. It is relatively insensitive to changes in speed and signal level and errors do not accumulate. It is less prone to the effects of dropouts and it will run at a frequency which is close to the upper limit of the medium. Almost no hardware is required. The limiting factor is usually the ability of the processor to sample and store the incoming information. A version of TRANSWIFT can be devised which samples both positive and negative halves of the waveform to verify each bit and make an intelligent decision to correct faulty data.

HOW TO WRITE A TRANSWIFT INTERFACE FOR YOUR SYSTEM

Each data word of eight bits is sent LEAST-SIGNIFICANT-BIT first. A LOW bit is sent by a single cycle of 2000 HZ or by putting the output high for 250 microseconds, then low for 250 microseconds. A high bit is sent by putting the output high for 500 microseconds, then low for 500 microseconds. A high bit corresponds with a single cycle at 1KHZ. It takes twice as long to send a high bit as it does to send a low bit.

TRANSWIFT does not have any framing information between words - each bit is sampled as an entity and speed errors do not accumulate from bit to bit. TRANSWIFT is insensitive to quite large changes in speed. At the start of the transmission at least 20 bytes of AA should be sent (alternate 1's and 0's) to overcome any Automatic Volume Control bounces in the tape recorder input circuitry. Then a 69 byte (01101001) is sent to "sign-on" and this is immediately followed by the main transmission. Each word is transmitted straight after the last and any timing-loops used for fetching new data and calculation should be filled out to ensure that all transitions occur at the correct bit-times as described above.

During the transmission process each word is EXCLUSIVE-OR'd with a 'parallel-parity' byte to form a check on reception. In SOFTY this 'parallel-parity' byte is AA, but it could be anything you please. The resulting byte is transmitted at the end of the data. When all the data is EX-OR'ed with this final word the result will be AA (or whichever byte you chose). For this reason the TRANSWIFT input and output routines when called from the SOFTY keyboard always end with AA in the PROMPT location.

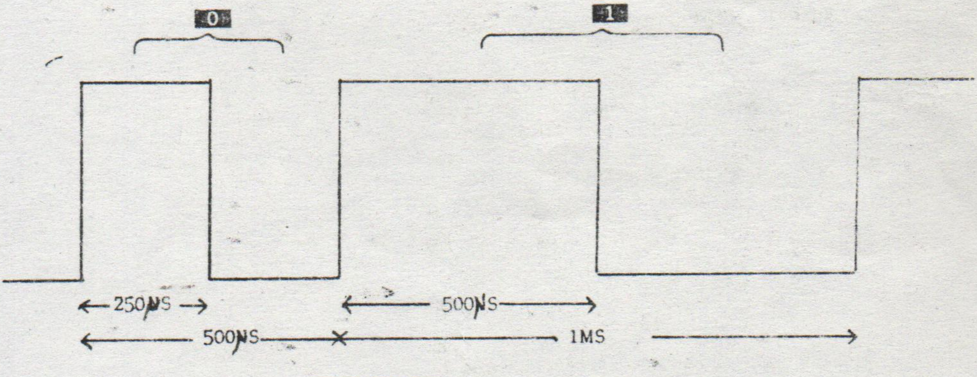
TRANSWIFT INPUT

TRANSWIFT bit-input routines require that a loop is formed which finds a positive transition. (Loop-till-low followed by loop-till-high.) Then a delay of 375 microseconds is executed and the bit is clocked in. It should be obvious that at this stage the input will be in the same state as the bit was when transmitted EVEN IF THE TRANSMISSION HAS BEEN INVERTED.

The data will not be meaningful if it is not correctly broken into bytes. The receiving routine should first clock in bits continuously, accepting only AA or 55 as valid leader bytes. A LEADER COUNTER is decremented after each bit input if the word is then AA or 55: if it has neither of these values the LEADER COUNTER is restored to its starting value again.

When the LEADER COUNTER is zero the system may then settle to look for the 69 byte which aligns and signs-on the transmission. When the 69 arrives the input routine should receive 8 bits at a time and then store away the byte in the next available memory location.

When the last byte is received it is EX-OR'ed with the previous data to check the validity of the transmission. In SOFTY the transmission is not aborted if it is invalid, because this may be due to a single dropout which could be found by inspection. ..).(The word drop-out meaning a missing bit, is a convention - drop-ins are just as frequent. TRANSWIFT is comparatively immune.)



TRANSWIFT

Bit is sampled 375NS after positive transition

EXAMPLES OF TRANSWIFT ROUTINES FOR Z80

```
                        ;
                        ;***********TRANSMIT*************
                        ;
                        ;TRANSWIFT TRANSMISSION ROUTINE
                        ;
02E1 2E14               TRANSMIT:   LD    L,OUTLENGTH
02E3 1EAA               LEAD:       LD    E,LEADBYTE
02E5 CD0803                         CALL  WORD
02E8 2D                             DEC   L
02E9 20F8                           JR    NZ,LEAD
02EB 53                             LD    D,E
02EC 1E69                           LD    E,SYNCBYTE
02EE CD0803                         CALL  WORD
02F1 2100F8                         LD    HL,BOTRAM
02F4 5E                 BYTE:       LD    E,(HL)
02F5 7A                             LD    A,D
02F6 AB                             XOR   E
02F7 57                             LD    D,A
02F8 CD0803                         CALL  WORD
PAGE  19

02FB 23                             INC   HL
02FC 3EFC                           LD    A,TOPRAM/256
02FE BC                             CP    H
                        ;
02FF 20F3                           JR    NZ,BYTE
0301 5A                             LD    E,D
0302 CD0803                         CALL  WORD
0305 3EFF                           LD    A,OFFH
0307 C9                             RET
                        ;
                        ;
                        ;*************WORD****************
0308 0608               WORD:       LD    B,8
030A 3E0B               LOOP:       LD    A,HIGH
030C D303                           OUT   (PORTD),A
030E CD1D03                         CALL  DELAY2
0311 3E0A                           LD    A,LOW
0313 D303                           OUT   (PORTD),A
0315 CD1D03                         CALL  DELAY2
0318 CB0B                           RRC   E
031A 10EE                           DJNZ  LOOP
031C C9                             RET
                        ;
031D 00                 DELAY2:     NOP
031E 3E19                           LD    A,LOWTIME
0320 CB43                           BIT   0,E
0322 2802                           JR    Z,LOOP1
0324 3E32                           LD    A,HIGHTIME
0326 3D                 LOOP1:      DEC   A
0327 20FD                           JR    NZ,LOOP1
0329 C9                             RET
                        ;
                        ;
```

```
                         ;*************RECEIVE***************
                         ;TRANSWIFT RECEIVE ROUTINE
                         ;
032A 2100F8   RECEIVE:       LD   HL,BOTRAM
032D 3E98                    LD   A,098H
032F D303                    OUT  (PORTD),A
0331 061E     SETLEAD:       LD   B,INLENGTH
0333 CD5903   INTRO:         CALL GETBIT
0336 3EAA                    LD   A,LEADBYTE
0338 BB                      CP   E
0339 2805                    JR   Z,LEADER
033B 3E55                    LD   A,ANTIBYTE
033D BB                      CP   E
033E 20F3                    JR   NZ,INTRO
                         ;
0340 10F1     LEADER:        DJNZ INTRO
0342 CD5903   STARTER:       CALL GETBIT
0345 3E69                    LD   A,SYNCBYTE
0347 BB                      CP   E
0348 20F8                    JR   NZ,STARTER
                         ;
034A 0608     GETWORD:       LD   B,8
034C CD5903   GETBYTE1:      CALL GETBIT
034F 10FB                    DJNZ GETBYTE1
0351 73                      LD   (HL),E
0352 23                      INC  HL
0353 3EFC                    LD   A,TOPRAM/256
0355 BC                      CP   H
0356 20F2                    JR   NZ,GETWORD
0358 C7                      RST  0
                         ;
PAGE  20

                         ;
                         ;*************GETBIT****************
0359 DB02     GETBIT:        IN   A,(PORTC)
035B CB6F                    BIT  TAPEBIT,A
035D 20FA                    JR   NZ,GETBIT
035F DB02     WAITHI:        IN   A,(PORTC)
0361 CB6F                    BIT  TAPEBIT,A
0363 28FA                    JR   Z,WAITHI
0365 3E28                    LD   A,WAITTIME
0367 3D       RLOOP:         DEC  A
0368 20FD                    JR   NZ,RLOOP
036A DB02                    IN   A,(PORTC)
036C CB6F                    BIT  TAPEBIT,A
036E 37                      SCF
036F 2001                    JR   NZ,ONE
0371 3F                      CCF
0372 CB1B     ONE:           RR   E
0374 C9                      RET
                         ;
                         ;
```

EXECUTION OF PROGRAMS BY SOFTY'S OWN MPU

Programs may be written for the INS8060 (SC/MP) and run from any point in addressing space where there is EPROM or RAM. In fact one of the virtues of SC/MP is that all programs are essentially relocatable - only the values assigned to the pointer registers are absolute. There are four different areas where programs may be placed:

1) You might write your program into a 2716 EPROM and use this to replace SOFTY's own firmware. This could make SOFTY into quite a different sort of tool. There is a provision at the keyboard for copying back firmware into the RAM buffer in case it is decided to retain some of SOFTY's functions but rewrite others.

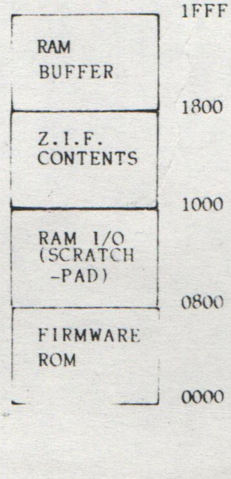
2) The new program could be written into EPROM and run directly from the ZIF socket. There is a keyboard function which transfers program control directly to the ZIF socket for this purpose.

3) The program can be written in screen RAM with the benefit of SOFTY's EDITING FACILITIES. The key marked EXECUTE starts program execution with the location following the cursor. If the program meets a 3D byte there will be a return to monitor and the SUBROUTINE POINTER (P3), ACCUMULATOR, EXTENSION REGISTER AND STATUS REGISTER contents will be printed in the STATUS LINE. It is important that the user's program does not reassign P1 or P2 if the editing facility is desired.

When the program is restarted with EXECUTE there is no loss of status because these internal register values are reloaded from the STATUS LINE. The user's program may employ P2 to gain access to PORTS and SCRATCHPAD. It does not matter if the user's program reassigns P3 because the monitor restores it to the KEYBOARD SCAN subroutine. The user's program may use the combination 3F 00 which will halt until a word is fetched from the keyboard. This KEYWORD may be read from P2 7B in the STATUS LINE.

4) It is also possible to call a block which has been transferred to SCRATCHPAD with the STORE function - GOSUB does this. There is no need to add any information to return to the program from this "custom keyfunction" - this information is added automatically when the block is stored.

An example of a User-programmed keyfunction: write C4 AA C9 00 which means load AA and store it at the location of POINTER 1 (CURSOR). Use the define function to enclose these four bytes. Then use the STORE function to transfer this little routine to SCRATCHPAD. Now you have a (pretty useless) extra key function. When GOSUB is called the byte AA will be inserted at the CURSOR location.



13 ADDRESS LINES DECODED

SOFTY MEMORY MAP