IMPROVING THE MARK-8

by Terry F. Ritter
25248 Glen Springs Way
Austin, Texas 78741

Several weeks ago, I mentioned to Hal that I would write up a description of the hardware PUSH/POP stack I had implemented on my Mark-8. At that time the stack itself was fully operational, but I planned further modifications to establish a buss system that would reduce the number of wires between the Mark-8 and and my peripheral breadboard. As it turns out, such a system is not trivial.

It was my intention to extend the Mark-8 data buss (pins 25-18) some ten feet to my breadboard; timing signals would be generated by circuitry added to the Mark-8 CPU board, and be similarly extended to the breadboard. This scheme would appear to allow address data to be latched into the peripheral registers during T1 and T2, as well as making the buss contents availiable during T3 and T4, all on 12 wires (plus power) instead of 26. While I am quite sure that such a system can be implemented, my attempts, using ribbon cable, and standard TTL as line drivers were not successful. Although many timing windows were configured, most giving apparently-suitable waveforms to the distant latches, latched data was rarely correct. It seems probable that transient pulses on the timing lines were latching transitional data, but this is not confirmed. In any case, this simple project quickly got out of hand time-wise, and was terminated prior to success.

For some time, I have been planning some rather complex software systems for my home computer, including a machine-language trace/editing system, and an interpreting language suitable for extensive complex-algebra manipulations. Very soon, however, I became aware of several of the serious deficiencies in the 8008 instruction set. In particular, the 8008 allows no true interrupt service, since it is normally impossible to fully save the processor state at the time of the interrupt and restore it after the interrupt has been answered. There is also no normal access to the program counter or the PC stack. These particular deficiencies are now corrected in some degree with external hardware. In addition, construction of the stack system, as well as other peripherals, implied use of input ports not availiable on the Mark-So Accordingly, an input-port buss system was developed that made input ports 1-7 availiable on the peripheral breadboard. The presented hardware modifications and additions thus consist of additional input port capability, a PUSH/POP stack designed for interrupt service (including flag latch), and program counter access.

The Input Port System

Since it was desired to place input ports 1-7 on the external breadboard, one approach that seemed reasonable was to place the appropriate data on port 1 input of the Mark-8. This appeared to reduce external complications, since port timing was still accomplished within the Mark-8 itself.

Input data from ports 1-7 is accepted on port 1 of the Mark-8 after modification of the select circuitry associated with Ie?, the port-select 7442 on the input board. This modification consists of the addition of seven germanium diodes (which form a seven-input negative-logic OF gate) between the 1-7 outputs of the 7442 and pin 6 of IC6, the 7402 controlling the 8263 input multiplexers.

The input-port structures are configured from six sets of 8-946 quad DTL nand-gates each, with gate outputs in parallel (alternatel:;, 12-7403'5 could be substituted, with eight pull-up resistors on the common buss). This results in a low-impedence inverted-data buss to the computer, where eight inverters (two more 946's) correct the data intc the port 1-7 channel.

en the peripheral board it is necessary to develop signals similar to those to and from our (now modified) 7442 on the input board. This requires all high-address lines (bits 9-15) for development of Den (I/O enable), INPUT, as well as the port-select signals. Six of the 7442 output lines are inverted (complements of one 7404) and used to enable the appropriate input ports (one non-inverted line is used for the POP instruction in the stack, yet to come).

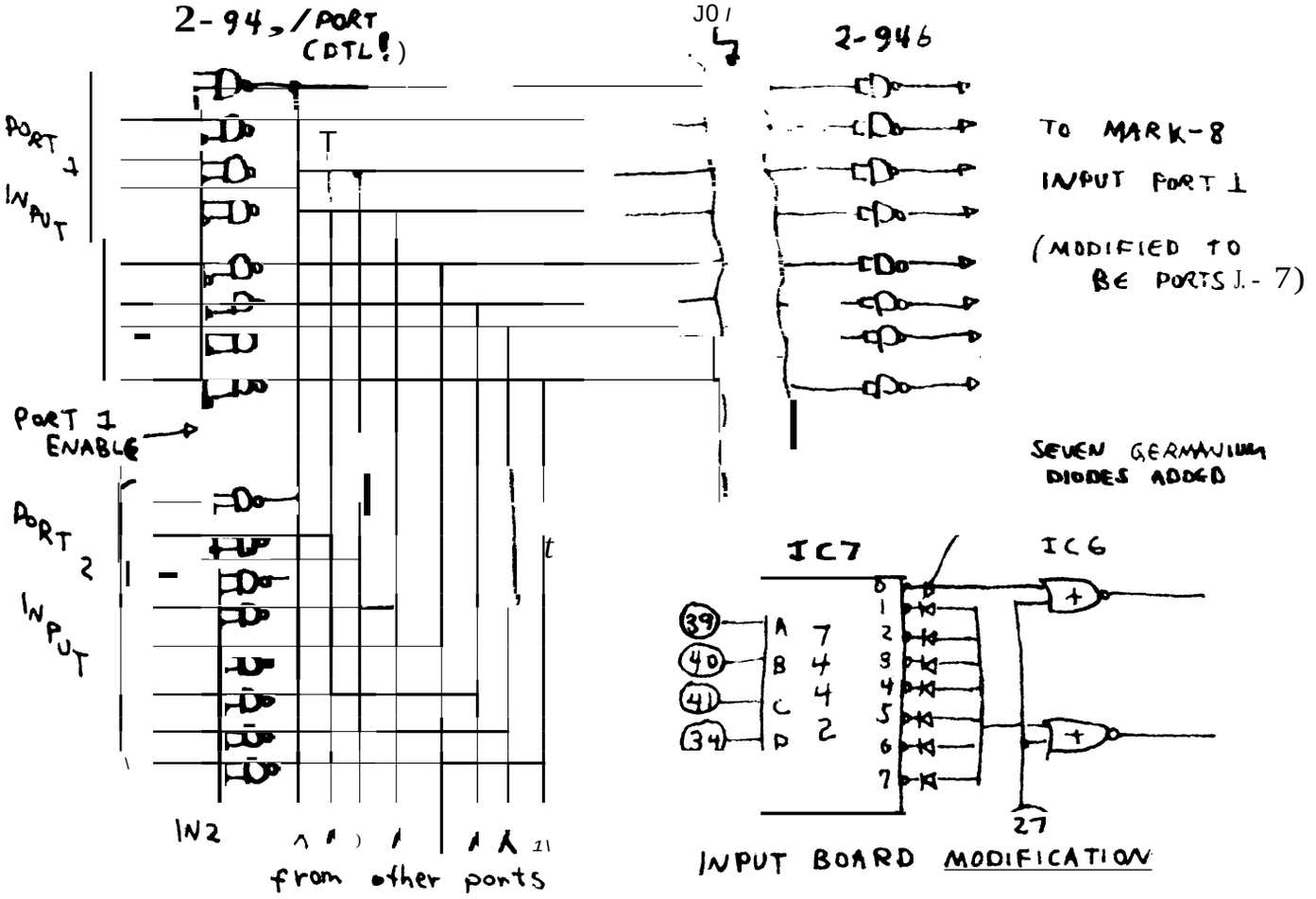
The Hardware PUSH/POP Stack

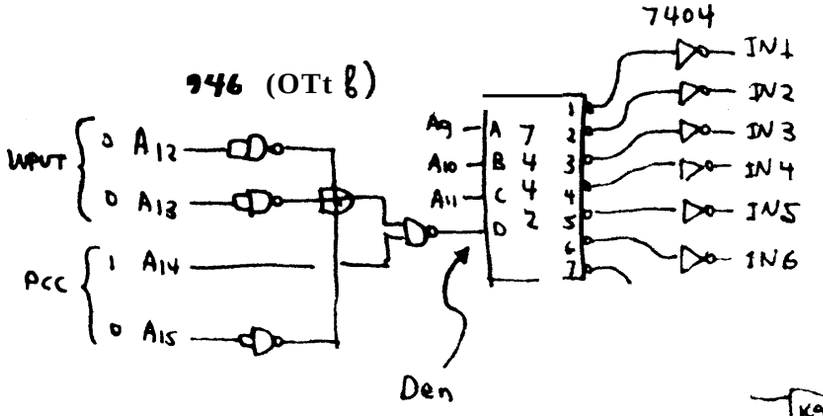
The Push/Pop stack is designed to store the complete status of the CPU prior to execution OF an interrupt routine. this includes all registers and flags. Although a software system could be used to identify and store the flags, in this implementation an INPUT 7 instruction causes the flags to be latched externally during what is effectively an OUTPUT-type command (PUSH). These flags are then brought into the accumulator (by another PUSH) and stored on the stack on top of the registers. When status is to be recovered, flag values are easily POPed from the stack and used as an address into a data table in the last sixteen words of a particular page. The appropriate data is then loaded into register A which is added to itself to set the flags to their original valu.es. The registers are then recoverer] from the stack the accumulator to restore the original CPU status.

The concept for the PUSH/POP stack is taken directly from an article by Tom Pitman writing in Electronic Design for November 22, lq74 (p. 202), although the logic Is re-designed for perhaps more-availiable IC's. Two 7489 RAM's are used as a 16-1evel stack with a 74193 as level-counter. Two 946's (and another diode) provide appropriate timing signals, while a single 7475 latches the flags from the data buss. Timing signals T3 and 1"4 are required by the stack implementation and are obtained vIa rihbon-cable from Ie17 pin 3 and Iell pin 10, respectively, on the CPU board. The necessary input commands (IN6 and NOT IN?) a.re already availiable from the input circuitry. Four data-buss lines are required for the flag latch, however.

The original stack level need not be set to any particular value, since tho 74193 is a mod-16 counter and will traverse the stack as a circular list. Hence, the original level-value is meaningless unless it is desired to detect a stack overflow. In the absence of such hardware modifications, of course, it is the responsibility of the programmer not to exceed the stack capacity.

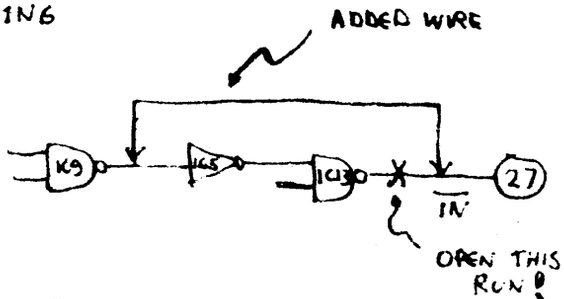
EIGHT INPUT PORTS





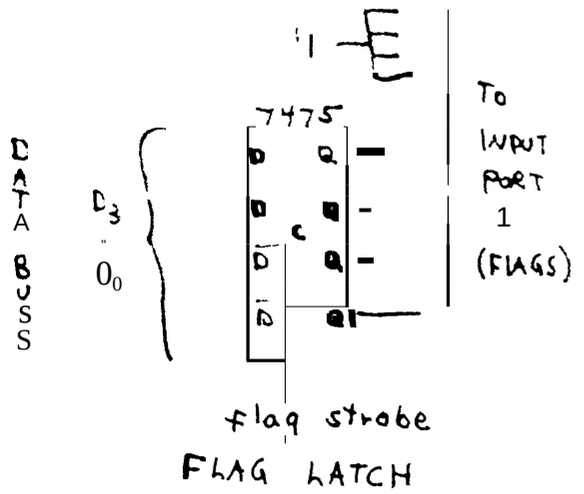
PORT SELECT CIRCUITRY

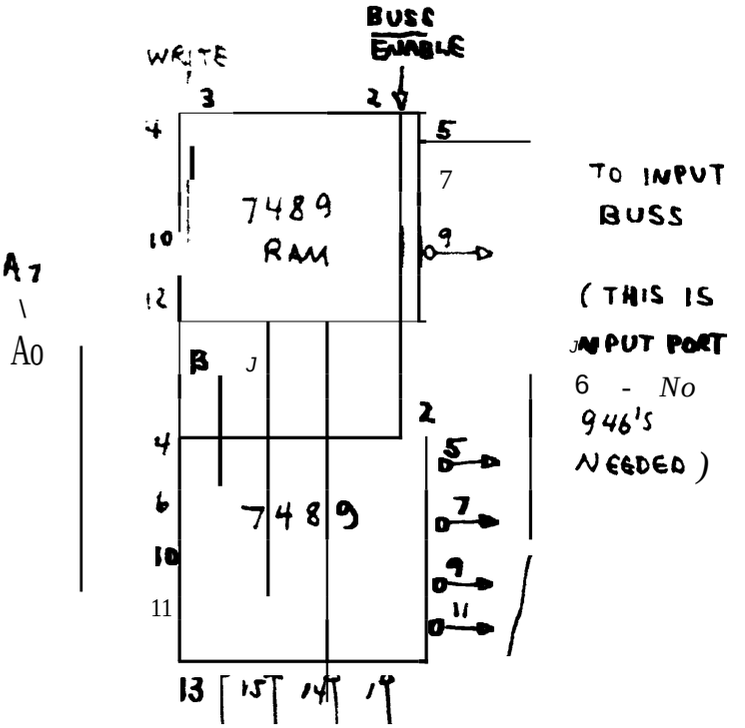
(on peripheral board)



CPU BOARD MODIFICATION

PUSH / POP STACK !

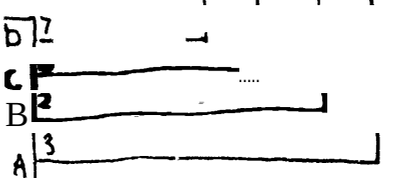




INPUT 7 = 117
PUSH

T1: T1 LATCH ← A
T2: T2 LATCH ← INSTRUCTION
T3: A ← FLAG LATCH
T3→T4: LEVEL ← LEVEL + 1
T4: STACK ← 1 J LATCH
T4: FLAG LATCH ← FLAGS

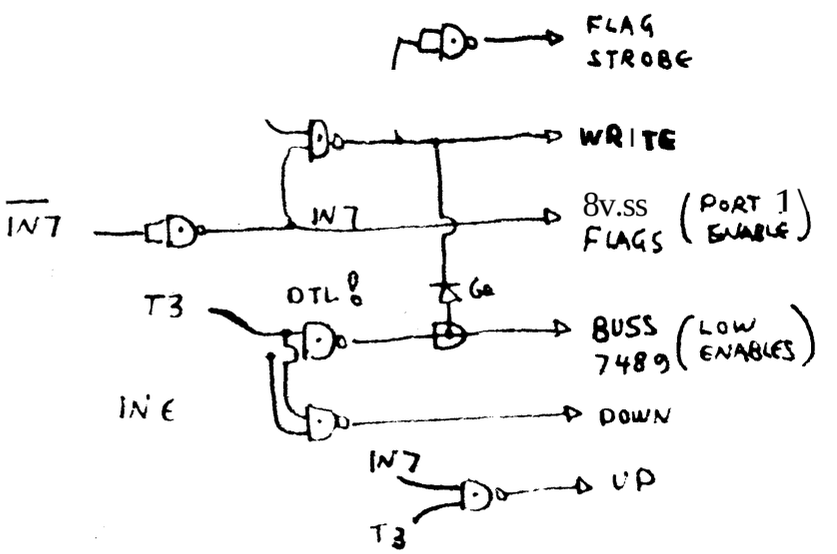
uP DOWN



LEVEL COUNTER + STACK

INPUT 6 = 115
POP

T1: T1 LATCH ← A
T2: T2 LATCH ← INSTRUCTION
T3: A ← STACK
T3→T4: LEVEL ← LEVEL - 1



STACK CONTROL LOGIC

Simple Stack Test

```
00 100  301   ←   push
00 101  117       sequential
00 102  010       data
00 103  110
00 1C4  100   ←
00 1C5  000
00 106  121   ↓
00 107  000       HALT for
00 110  115        display
00 111  104
00 112  106   ↑
00 113  000   \   pop stack loop
```

00 102 can be all

Simple Flag Recover Test
(needs flag table at 01 360)

change 01 076 for various
flags at 01 104

```
01 075  C06
01 076  34C   <   modify at
01 077  200         will
01 100  117
01 101  117
01 102  117
01 103  121
01 104  C0E   -   display flags
01 105  250       before save
01 1C6  117
01 107  117
01 110  121
01 111  000   -   display
01 112  056       altered
01 113  001       flags
01 114  115
01 115  115
01 116  115
01 117  360
01 120  307
01 121  2CC
01 122  117
01 123  117
01 124  121
01 125  000       display
                  recovered
                  flags
```

HALT

RECOVER ←

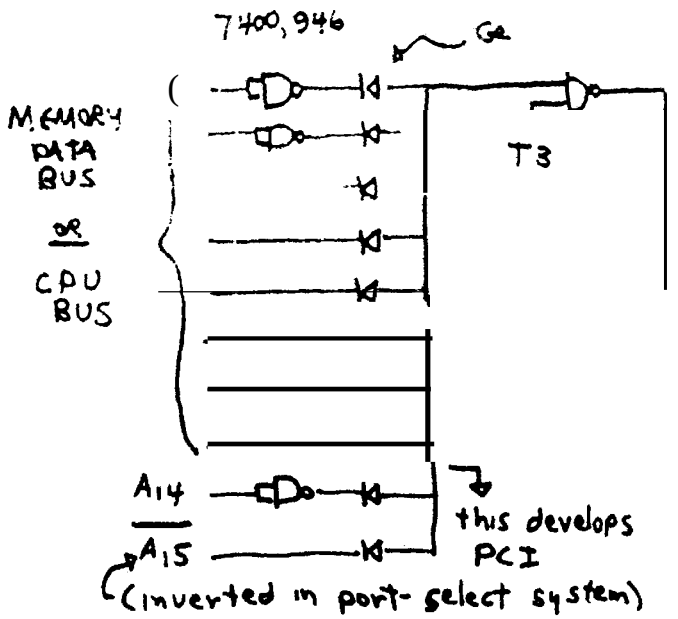
Flag table

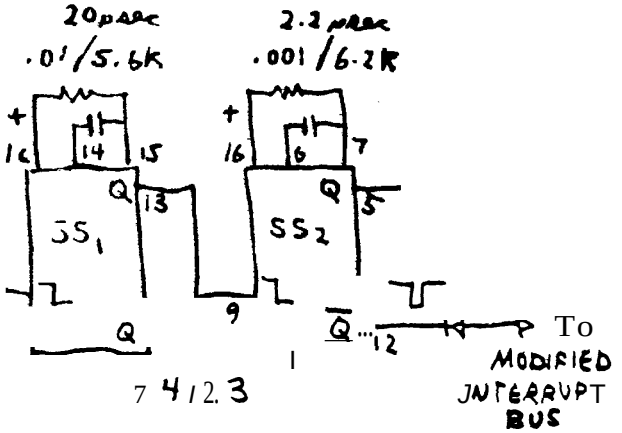
	b_3	b_2	b_1	b_0
'1'	CARRY	EVEN	ZERO	NEG.
'0'	NO-CARRY	ODD	NON-ZERO	POS.

FLAG CONDITIONS

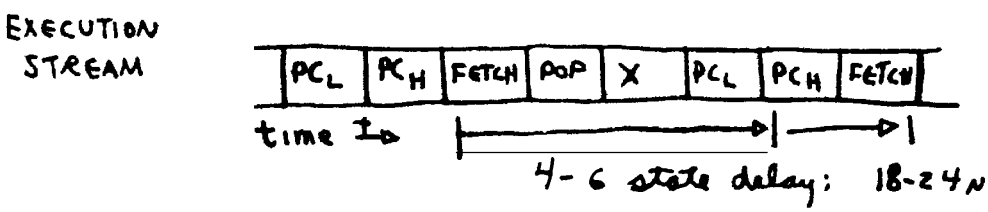
```
01 315  117
01 316  301
01 317  117
01 320  302
01 321  117
01 322  303
01 323  117
01 324  304
01 325  117
01 326  305
01 327  117
01 330  306
01 331  117
01 332  117
01 333  007
01 334  056
01 335  C01
01 336  115
01 337  360
01 340  307
01 341  200
01 342  115
01 343  360
01 344  115
01 345  350
01 346  115
01 347  340
01 350  115
01 351  330
01 352  115
01 353  320
01 354  115
01 355  310
01 356  115
01 357  007
01 360  040
01 361  100
01 362  000
01 363  000
01 364  060
01 365  140
01 366  000
01 367  000
01 370  240
01 371  200
01 372  000
01 373  000
01 374  260
01 375  340
01 376  200
01 377  000
```

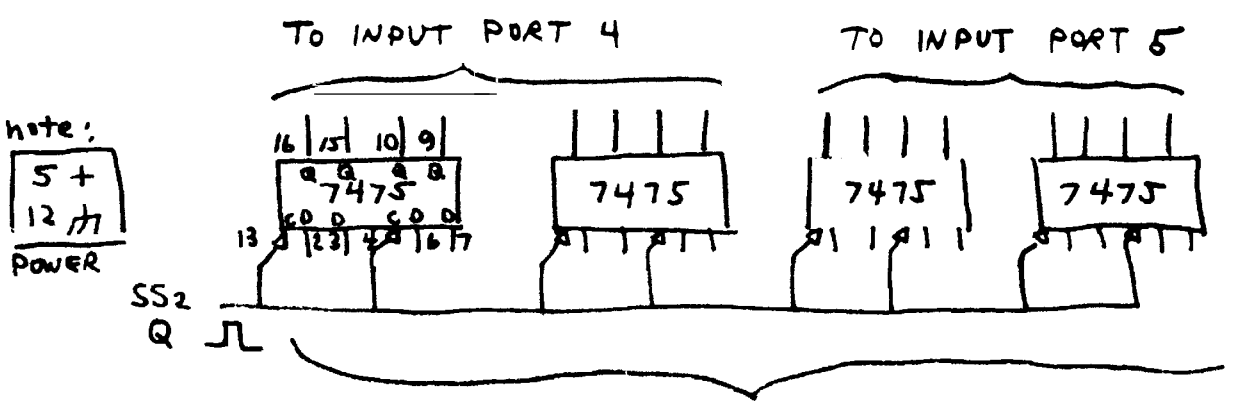
077 DELAYED AUTO-INTERRUPT AND p.c. ACCESS



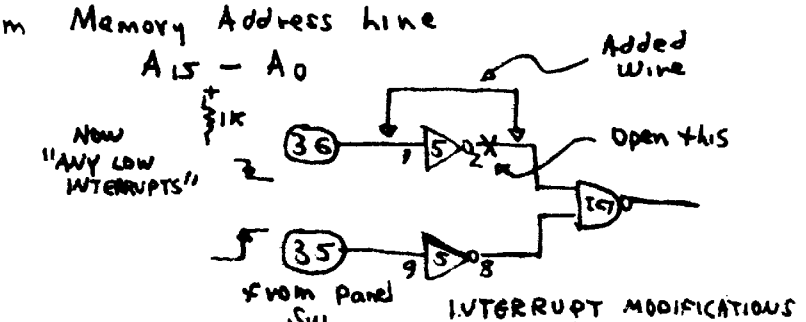


delays through RETURN execution and past start of next instruction cycle





THIS HARDWARE SYSTEM SAVES THE PC ADDRESS OF THE LAST EXECUTED STEP OF THE TRACED OPERATION



Program Counter Access

Certain types of programs, in particular TRACE systems, require more than just valid interrupt-handling capabilities; they also may require access to the last value of the program counter before the interrupt. My implementation is to use a special RETURN code to activate a delayed interrupt. As the last instruction of an interrupting system, control passes back to the original program; the delayed interrupt allows one complete instruction to be executed before the interrupting program is again entered. The interrupting program can include octal display of the state of the CPU, or test to determine if display is necessary. Typically, an ASCII SPACE might return processing to the original program until the interrupting program determines that display is again desirable.

Of course, access to a 14-bit address implies that two input ports must be used to regain the data. This paper thus embodies the following input port assignments: 4:PCH; 5:PCL; 6:PUSH; 7:POP.

Back to the Stack

The Stack could alternately have been arranged to store register data without channeling it through the accumulator, by using 3xx ($x \neq 7$) NO-OP instructions. This method would preclude the use of the 3xx NO-OP in a timing loop, although an output instruction could be used instead. But, although register PUSH operations would be quite convenient, POP would still need to come in through the accumulator, and a new software system would be necessary to detect and store the flags in a register, so this method is not as attractive as it first appears.

The POP instruction does not destroy stack data, so repeated POP's may be used to gain access to the stored previous state of the CPU. Of course, 16 POP's are necessary to rotate the stack to its original position. The stack may also be used for temporary storage (for, say, EXCHANGE H,L -type instructions) if care is taken not to overflow the stack and thus destroy stored state-recovery data.

TERRY SAYS THE INCLUDED ARTICLE PLUS HIS OSCILLOSCOPE SYSTEM YET TO COME REPRESENT WELL OVER 2 MONTHS OF SPARE TIME WORK. HE SAYS SARDO IS FAST AT DELIVERING 1702A AND 5314 PROMS AND THEY ARE JUST A LITTLE SLOW (SPEC AT 1.7 MICROSEC) BUT SHOULD WORK OK. MINI MICRO MART IS VERY VERY SLOW AND THEIR BOARD DESIGNER IS JUST PLAIN INCOMPETENT. HE'LL SUPPLY MORE ON THIS LATTER. HE LIKE PARTICIPANT'S ADVICE ON WHETHER THE COMPUTER HOBBYIST AND DIGITAL GROUP NEWSLETTERS ARE WORTHWHILE. HE GETS 20 MAGAZINES PER MONTH NOW BUT WILL SUBSCRIBE IF PARTICIPANTS RECOMMEND THEM HIGHLY ENOUGH.

BILL FULLER, 2377 DALWORTH 157. GRAND PRARIE, TX 75050 AND L. G. WALKER ARE TRYING TO GET A GROUP GOING IN THE NORTH TEXAS AREA AROUND JUNE 30. CONTACT THEM IF YOUR ARE INTERESTED.