Microbaud
Developments

MACHINE LANGUAGE INTERFACE

Dear Customer,

Unfortunately we have discovered three errors in the Construction Guide of which you should be aware. Please amend your copy as follows:-

The fourth paragraph below "CONSTRUCTION" should be changed to read:-

..... in the following section. Take M4b and M5b and bend pins 9 to 17 (EXCEPT PIN 12) out horizontal on both RAMs. Now take .....

The second paragraph below the heading "EPROM CURRENT DRAIN" contains two mistakes. It should read as follows:-

First, cut the track . . . . . the track joining U28 pin 10 to ground.* Connect pin one of U28 to the ground rail above it (on the rear of the PCB). Then, connect U26 pin 10 to U17 pin 9 using a piece . . . . . .

* This must be done prior to the insertion of the IC - it can still be cut if the rest of the EPROM CURRENT DRAIN modification is not carried out at this stage.

Good luck with the construction of your MLI, and we wish you every success with your future machine code programming.

Yours sincerely,

Michael Taze

for MICROBAUD DEVELOPMENTS.

D I S C L A I M E R

Whilst every effort has been made to ensure that this Machine Language Interface will operate as indicated efficiently and properly if constructed according to the attached details, or purchased assembled, no responsibility will be accepted in respect of failure for any reason at all of the aforesaid Interface to operate effectively or at all due to any fault in design or otherwise and no responsibility is accepted for any damage caused by the Interface to any connected equipment. Further, no responsibility is accepted in respect of any injury or damage of any kind caused by any fault in the design, construction or otherwise of the Interface as aforesaid.

SEMICONDUCTORS

2 CD4011 Quad 2-input NAND (U14,U27)
1 CD4012 Dual 4-input NAND (U22)
2 CD4013 Dual D Flip-Flop (U1, U13)
2 CD4017 Decoded Decimal Counter (U2, U3)
2 CD4021 8-stage Parallel Input Shift Register (U11, U12)
2 CD4023 Triple 3-input NAND (U16, U17)
2 CD4028 BCD to Decimal Decoder (U25, U26)
1 CD4049 Hex Inverter (U33)
1 CD4052 Differential 4-channel Multiplexer (U8)
1 CD4068 8-input AND (U21) ?
1 CD4073 Triple 3-input AND (U18)
1 CD4078 8-input NOR (U20)
2 CD4081 Quad 2-input AND (U15, U19)
6 CD4094 8-bit Shift and Store Register (U4, U5, U6, U7, U9, U10)
1 CD4503 (or 40097) 3-state Hex Buffer (U28)
2 MM74C85 (or 40085) 4-bit Magnitude Comparator (U23, U24)
4 6116 4Kx4 CMOS RAM (M4a, M4b, M5a, M5b)

1 2716 2Kx8 EPROM (M2)
1 2732 4Kx8 EPROM (M3)

These need to be pre-programmed with at least the WROM instruction. A suitable set is available from Microbaud Developments.

1 2N3906 transistor
1 Miniature red LED
1 1N4002 rectifier diode

CAPACITORS

1 100uF/16VW axial electrolytic
9 0.1uF bypass capacitors ("redcaps")

RESISTORS (1/4W, 5%)

13 100K
1 4K7

MISCELLANEOUS

Solder, spacers, nuts, bolts, etc.

OPTIONAL AC POWER SUPPLY

1 7805 regulator
2 0.1uF bypass capacitors ("redcaps")
2 1N4002 rectifier diodes
1 100uF/16VW axial electrolytic

CONSRUCTION

The first task is to drill the four mounting holes on the PCB, and, if you intend using the optional AC Power Supply, drill the mounting hole for the regulator. Next, fix the 16 pin DIP plug onto one end of the ribbon cable (this is done using a special clamping tool or a hammer[!!]). Solder the other end of the cable to the connector of an old module, having first removed and discarded the module PCB. Ensure that pin 1 goes to V+, pin 16 to B4 etc. **This lead is crucial to the whole MLI**, so it would be best to triple check the lead for continuity and accuracy.

Start construction on the PCB by placing the resistors and diodes into position. Check the orientation of the diodes and solder these components into the board. Next solder in the IC sockets, noting that no sockets are required for M1, I/O A, I/O C and 'SPARE', and the 6 pole DIP switch. If you are not using sockets be sure the ICs are correctly oriented, and solder the power supply pins first, with the barrel of the soldering iron connected to the ground track of the PCB. **N.B. DO NOT solder in the CMOS RAMs yet!** Place the capacitors, transistor and LED into the board, once again checking the orientation, and solder these and the battery clip to the board.

INTRODUCTION

The Microbaud Developments Machine Language Interface has two major sections. Firstly, it provides up to 4K of EPROM which can hold routines dedicated to the MLI and routines which are frequently used and are best kept in a permanent form, such as CODE, DECODE, an assembler and disassembler.

Secondly, the MLI contains 4K of CMOS RAM which is totally independant of the calculator's main memory and is under your total control. This memory is treated as an applications ROM by the calculator, and it is in this area that you may write your own machine language fuctions and routines.

There is also provision in the MLI for an external set of input/output ports, although these are not fully implemented on the board.

TECHNICAL OVERVIEW

The HP-41C/CV uses ten bit words to communicate data and instructions in a serial format both internally and with external devices such as applications ROMs or the MLI. On the block diagram you can see that data shifted out on the ISA line from the MLI to the HP-41 may come from the EPROM or the CMOS RAM (or the Input Port if it is implemented). One of these sources is selected by the Address Bus lines AB(C-F).

During any given 56-bit HP-41 machine cycle, sixteen bits of ROM address information is shifted out through the ISA line. They arrive at the MLI ISA Input Shift Register which latches them. ISA Enable (from the Control Signals Generator) gates this address onto the Address Bus. At this stage comparators check to see if the address on the Bus matches either the EPROM address or the RAM address. If either of these addresses match,the appropriate control signals (e.g. RAMO*, PORTO*, PROMO*) are asserted and the ISA Output Shift Register has the ten bit word on IOS(0-9) dumped into it at the correct time, and this word is subsequently shifted out.

The MLI decodes a special instriction in order to write to the CMOS RAM. The instruction is WROM (or WRITE SAM), hex code 040, and is a NOP in the HP-41 so that it does not affect the subsequent 56-bit machine cycle. When the write has been decoded, the data bits are used to determine the Note that the Data Input Shift Register latches the 'C' register in the HP-41 CPU (**not** the user-level 'c' register) whenever the least significant bit of C becomes DL0, thus the ten bit word loaded into RAM is the lowest ten bits of the C register.

Each instruction which arrives at the MLI is first latched into the ISA Input Shift Register, and then goes to the Control Signals Generator via the Address Bus AB(6-F). The WROM instruction is decoded in the Control Signals Generator and the most significant bits of the Data Input Shift Register are placed in the Address Bus AB(0-F), while the Data Lines DL(0-9) are gated to the RAM inputs. When both READ and RAMCE* are low (i.e. READ is **not** asser ted and RAMCE* is asserted), the ten bit word on the Data Line DL(0-9) is written into the RAM.

PARTS LIST

1 PCB "Microbaud Developments Macine Language Interface", 245x125mm
1 case, 250x145x30mm, e.g. custom-built marviplate steel box available from Microbaud Developments
1 16 pin DIL plug
1 6 pole DIP switch
1 battery holder (4xAA NiCd or 3xAA Alkaline)
1 battery clip (to suit battery holder)
1 length 16 core ribbon cable, about 35cm
1 old HP module

SOCKETS (if required)
4 24 pin DIL
18 16 pin DIL (note that at least 2 24 pin and
12 14 pin DIL 1 16 pin socket is required)

At this stage it is good practice to visually check your soldering, cleaning away blobs and strands of solder which may short tracks. Once the board has been cleaned and checked, all the ICs except the CMOS RAMs may be plugged into their sockets. Ensuring that a suitable set of EPROMS has been plugged into the board, disable the RAM by switching the RAM Enable switch (AND the XABC switch) to 'off', plug the ribbon cable into I/O B and a calculator and press 'ON'. Execute a CAT 2 and you should see the EPROM functions displayed. Try assigning 'CODE' to a key. Place the following string into ALPHA: "10414243444546". XEQ "CODE" (or press your assigned key. If "ABCDEF" appears in the X-register everything is fine.

The next step is to solder a piece of thin wire (wire wrap wire is recommended) between pin 15 of I/O C and the hole at the top right of the 'L' EPROM. Now comes the tricky part! We have four RAMs which we will call M4a, M4b, M5a and M5b. It may be best to tag the RAMs so that you don't get too confused in the following section. Take M4b and M5b and bend pins 9 to 17 out horizontal on both RAMs. Now take M4b and carefully bend pins 9, 10, 11, 13, 15 and 17 right up out of the way. Do the same to pins 10, 13, 14, 15, 16 and 17 of M5b. Place M4b on top of M4a piggy-back style and solder the remaining unbent pins of M4b to the corresponding pins directly below on cM4a. Give M5a and M5b the same treatment. Now plug the M4 and M5 RAM sets into their appropriate sockets (or solder them into the board).

Pin 16 of M4b should be overlapping pin 9 of M5b, as should pin 14 of M4b with pin 11 of M5b. Solder these two sets of overlapping pins so that there are two 'bridges' between the two RAM sets. Solder a piece of wire to each 'bridge' and connect one to each of the two adjacent holes on the right side of I/O B. The order in which they are connected is not important.

Switch the RAM Address switches to 8000, that is RA0 on, RA1, RA2 and RA3 off. To zero the entire RAM memory space you may use either of two methods:

1) With "OK" in the ALPHA register (a precaution against accidental clearing), run 'CLROM' in the Microbaud EPROM, or
2) Run 'ZR' from Appendix 2 with start address 8000 and end address 8FFF.

Now switch REN (RAM Enable) on (leave XABC off) and set up a null Function Address Table using "The ROM Adress Space" in Appendix 1 as a guide. You may like to give your pseudo-ROM RAM a name which will appear first in the CAT 2 listing. An example of this is also found in Appendix 1. You can now write, load and run your own machine language routines! To try out the MLI you may like to load some of the machine language routines which appear in Appendix 2.

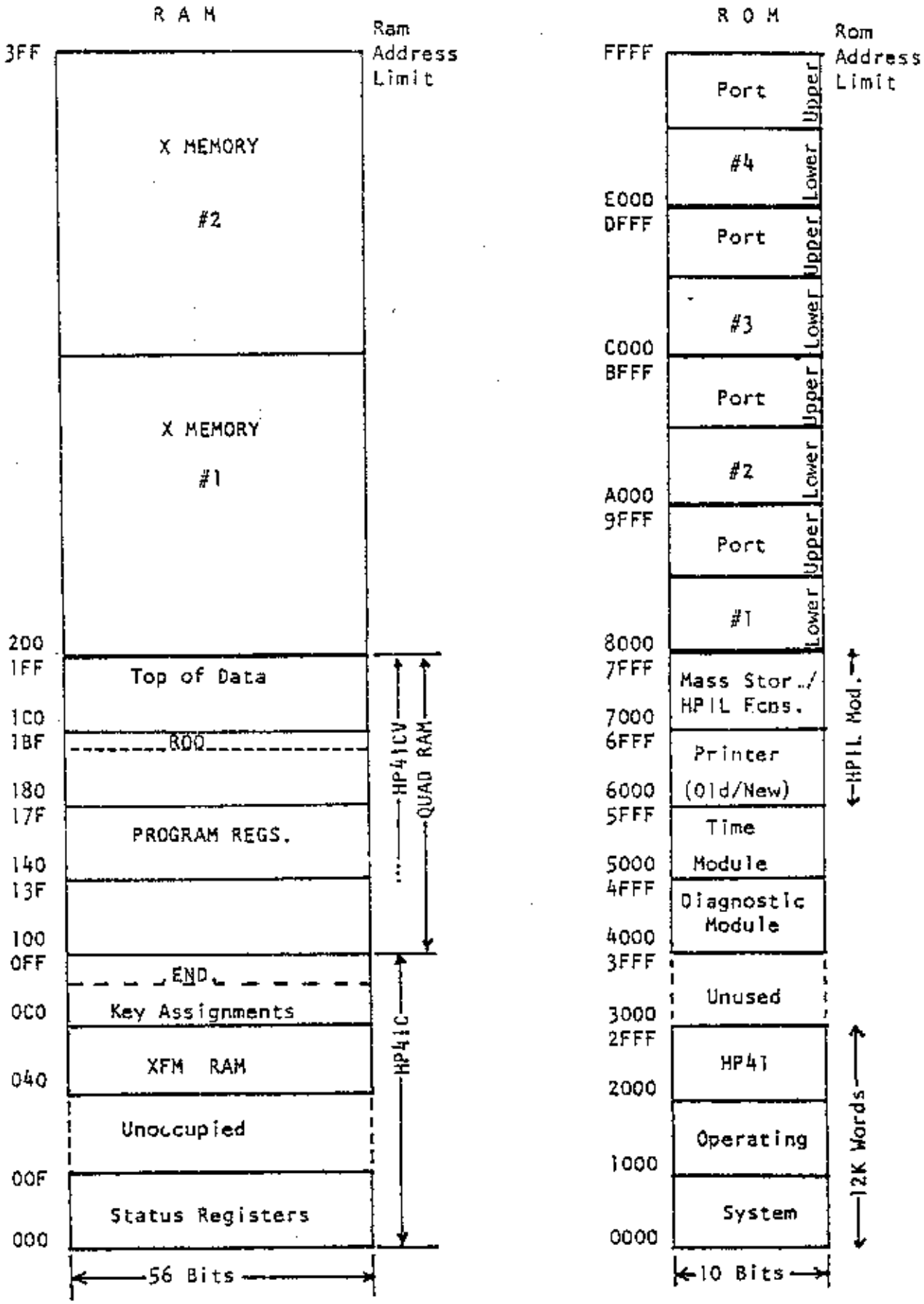
EPROM CURRENT DRAIN

Some brands of EPROM we have tried draw excessive amounts of current even when switched off - they have a very low impedance $\overline{OE}$ input which is high even when the MLI is unplugged, and so sinks about 50mA. Typically your MLI should draw around 100uA when unplugged. If you have an excessive current drain it can be fixed using the directions below.

First, cut the track coming from the pad at the top right of the 'L' EPROM. Connect the EPROM side of the cut track to the perimeter ground track using a piece of insulated wire. Next, cut three tracks, these being the track to the lower (cathode) lead of the LED, the track joining U28 pin 1 to +5V (on the rear of the PCB) and the track joining U28 pin 10 to ground.* Then, connect U28 pin 10 to U16 pin 9 using a piece of thin wire. This is best done by joining the two pads on the rear of the PCB. Connect U28 pin 9 to the lower lead of the LED in the same way. Check to see that the EPROMS are still working normally.

APPENDIX 1

The ROM Address Space and The Function Address Table

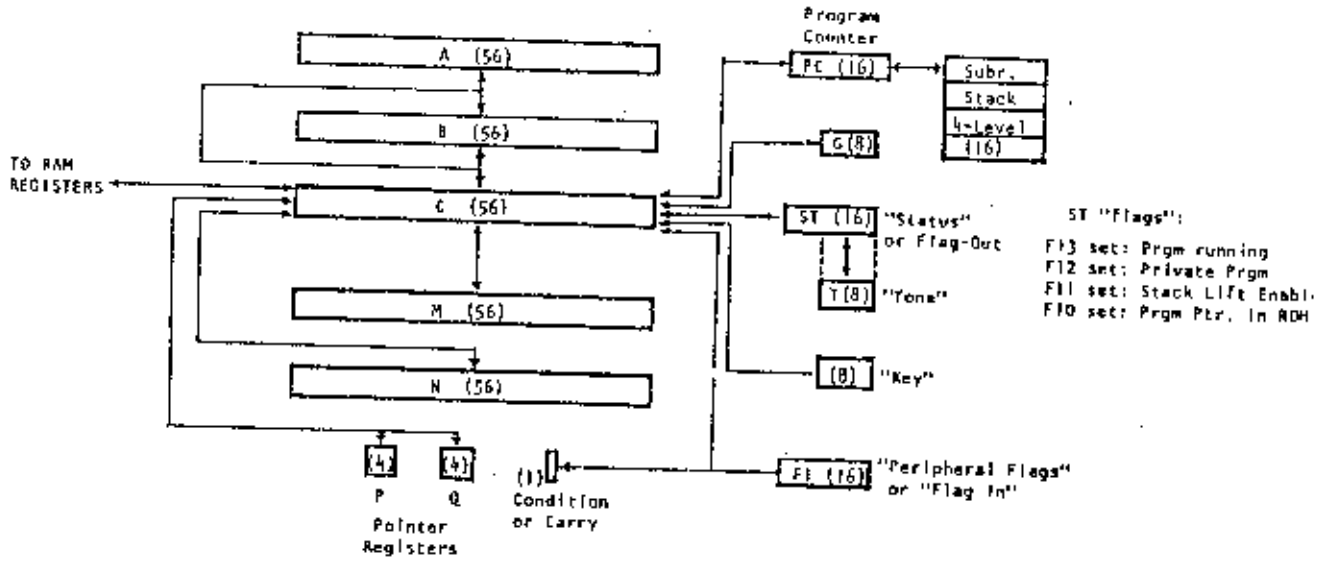


ROM and RAM are separate entities, mutually exclusive.

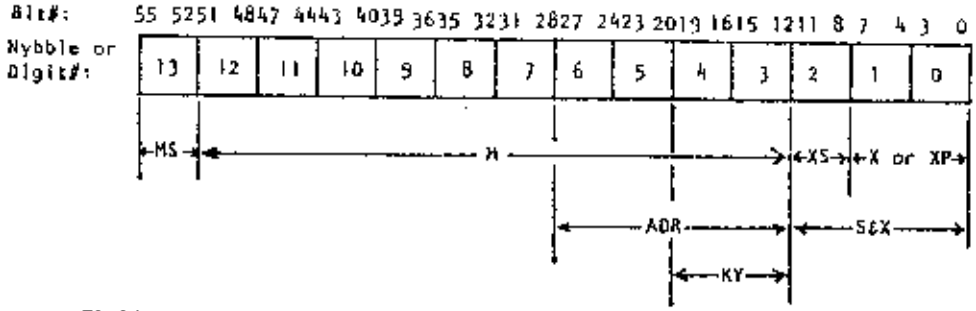
RAM is addressed by 10-bit addresses for each 56-bit register (000 to 3FF).

ROM is addressed by 16-bit addresses for each 10-bit ROM word (0000 to FFFF).

Relative Address	Function
X000	XROM Number
X001	Number of functions in the catalog (including module name)
X002, X003	Address of ROM module name
X004, X005	Address of first program
X006, X007	Address of second program
. . .	. . .
X080, X081	Address of 63rd program (if used)
Next 2 bytes after last addr	2 Nulls (000, 000)
. . .	. . .
Addr. of ROM name	Name of ROM (11 characters maximum)
. . .	. . .
Addr. of Fcn. #1	Function #1
. . .	. . .
Addr. of Fcn. #2	Function #2
. . .	. . .
XFF4 to XFFA	Special interrupt jump locations
XFFB to XFFE	ROM Name abbreviation and revision (i.e. CR1D, WD1E)
XFFF	8-bit checksum (for diagnostic module use)



HP41 CPU Registers. Arrows indicate possible paths of data flow. Note that the C register (56 bits long) is the main register through which information passes between RAM and CPU, as well as most CPU registers.



Field:		Nybbles:	Bits:
MS	(Mantissa Sign)	13	52-55
M	(Mantissa)	3-12	12-51
XS	(Exponent Sign)	2	8-11
X or XP	(Exponent)	0-1	0-7
ADR	(Address)	3-6	12-27
S&X	(Exp. & Exp. Sign)	0-2	0-11
ALL	(Entire Register)	0-13	0-55

R	Selected pointer (P or Q)
@R	At Nybble pointed to by R
R→	From Nybble 0 up to nybble
P-Q	From Pth nybble to Qth nybl
@R;+	At nybble R and R+1
LD@R,-	Load at nybble R and decre
SLCT P, SLCT Q	Select which pointer is ac

56 - BIT REGISTER FIELDS AND POINTERS IN THE HP41 INSTRUCTION SET. (S. Jacobs mnemonic

The RAM name which appears first in the CAT 2 listing is simply a function name followed by a return, placed anywhere in the in the RAM. However, the name must be the first 'function' listed in the Function Address Table. Also note that all the function names are listed in reverse, as shown in this sample FAT and the Wand listing excerpt.

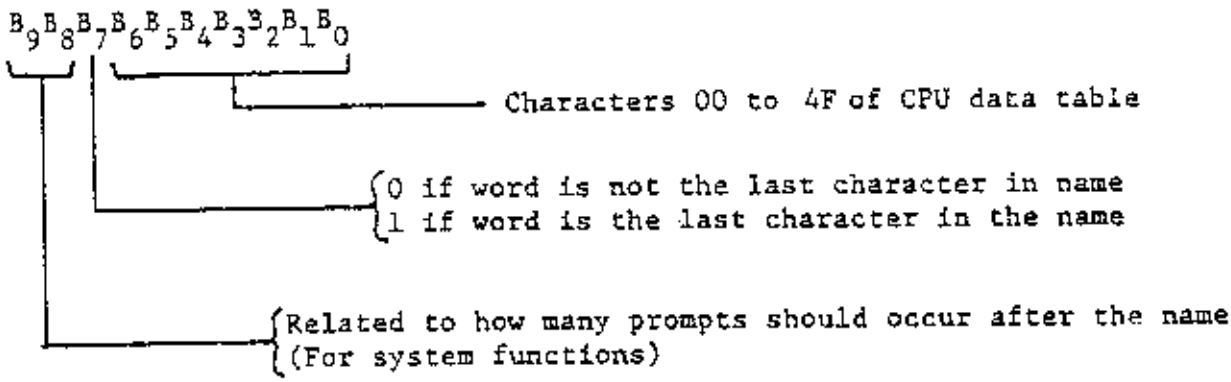
A Sample FAT

```
8000 013    XROM 19
8001 00F    15 FUNCTIONS
8002 001
8003 068    00: 8168 -MICROBAUD-
8004 000
8005 092    01: 8092 CSST
8006 001
8007 014    02: 8114 LEFT
8008 001
8009 025    03: 8125 GOOSE
800A 001
800B 02F    04: 812F +1
800C 001
800D 050    05: 8150 RESETFL
800E 001
800F 072    06: 8172 TPR
8010 001
8011 083    07: 8183 THX
8012 001
8013 00E    08: 818E SF50
8014 201
8015 0A2    09: 81A2 'DEMO
8016 001
8017 0E6    10: 81E6 UPDFAT
8018 202
8019 022    11: 8222 '+1R
801A 202
801B 042    12: 8242 'VER?
801C 002
801D 08D    13: 828D CFX
801E 002
801F 092    14: 8292 SFX
8020 000
8021 000
```

A Sample RAM Name

```
8160  0AD  "_"
8161  004  "D"
8162  015  "U"
8163  001  "A"
8164  002  "B"
8165  00F  "O"
8166  012  "R"
8167  003  "C"
8168  009  "I"
8169  00D  "M"
816A  02D  "-"
816B  3E0  RTN
```

XROM Machine-Code Function-Name Words:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
01	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
02	←	!	"	#	$	%	&	'	<	>	*	+	←	-	→	/
03	0	1	2	3	4	5	6	7	8	9	▩	,	∠	=	≥	?
04	-	a	b	c	d	e	—	⊤	⅂	⅃	⅄	⅄	⊔	≠	Σ	∠

CPU Data Table for
Non-last Characters of
Machine Code Function Names

(For last character, add 080 hex to the code in the above table.)

XROM User-Code Function Names:

If the program in question is user code, then the two words pointing to the program's address point to the first word in the program, which continues forward from that point. (This first instruction in the program would be an ALPHA label.) The two words previous to the pointed address may contain information related to the program's size for copying purposes, if this label is line 01 of the program.

Example:

	Addr.	Code:	Function:
	00D	0BE	
	00E	204	Addr. 402
	00F	002	for fcn.
	.	.	
	.	.	
	400	009	9 Regs to copy
	401	220	
→	402	1C2	LBL
	403	001	
	404	0F7	r
	405	000	

Addr.	Code:	Function:
406	057	W
407	04E	N
408	044	D
409	054	T
40A	053	S
40B	054	T
40C	19C	FIX
40D	000	0
40E	1A6	XROM
40F	0C5	27,05 (WNDSC
.	.	.

APPENDIX 2

This Appendix contains some routines which are useful examples of machine language programming, its applications, speed and operation. The following is a brief description of each routine listed later in this section. We have included a smattering of various mthods of operation (for example immediate execute, programmable and non-programmable functions).

CSST - continuous single step. This routine is non-programmable and when executed it continuosly single-steps through the current program, beginning at the current line. If flag 0 is set, CSST will step backwards, or this can be effected while you are stepping by pressig the 'ON' key. The 'ON' key toggles the stepping direction forwards or backwards. Each key apart from 'ON' and the backarrow (which exits the program) also has a stepping speed associated with it, moving down each column with Σ+ being the fastest down to R/S, then ALPHA, PRGM and finally USER being the slowest.

SF50, LEFT, GOOSE - These three routines are used by the user-level program "DEMO". LEFT rotates the display left by one character, GOOSE places a left-facing goose in the display and SF50 sets flag 50 (surprise, surprise!), the message flag.

"DEMO" - This user-level program shows some of the somewhat unusual (even in the realms of synthetics!) things which may be achieved using machine language. Running "DEMO" yields a left-facing goose flying right to left (minus the Wickes goose dropping). Local labels 10 to 14 have the following results:

- LBL 10 - left goose flies left
- LBL 11 - ALPHA register rotates left
- LBL 12 - right goose flies left
- LBL 13 - left goose flies right
- LBL 14 - right goose flies right (as is normal)

+1 - This is an extremely simple routine which adds up by ones in machine language. Use it to impress your friends when compared to "+1R" run for the same length of time!

"+1R" - As for '+1', but in user code. Much, much slower!

RESETFL - Resets flags to MEMORY LOST status except flag 27 (USER) and flag 31 (TIME module DMY) are both set.

TPR - Prompting tone. This routine prompts for a 3-digit number and excutes the corresponding TONE. This is not programmable.

TNX - Tone from X. A programmable funtion the same as "TN" in the PPC ROM.

UPDFAT - This updates the Function Address Table when new routines are inserted. Key the address of the new function (that is the address of the first executable word in the routine) into ALPHA, execute 'CODE', execute 'UPDFAT'. The FAT is updated by incrementing the number of functions and adding the routine address to the end of the FAT. If you had just keyed in the routine "TNX" at addresses 8180 to 8187, then to incorporate this routine in the FAT, type:

"8183", XEQ "CODE", XEQ "UPDFAT"

"VER?" - This routine returns the current mainframe ROM revisions.

CFX - Clears the flag given by the value in X which can range from ØØ to 55.

SFX - Sets the flag specified by the number in the X-register. As for CFX, this can be any flag in the system.

```
808E 094 "T"
808F 013 "S"
8090 013 "S"
8091 003 "C"
8092 000 NOP
8093 22D
8094 0A8 ?NCXQ 2A88
8095 34C ?FSET 12
8096 211
8097 087 ?C=0 2184
8098 05A C=0 M
8099 05C R= 4
809A 050 LDAR 1
809B 170 PUSH ADR
809C 023 JNC 80A0 +04
809D 3F8 READ 15(e)
809E 381
809F 0A8 ?NCXQ 28E8
80A0 3C4 ST=0
80A1 104 CLRF 8
80A2 106 A=C S&X
80A3 086 B=A S&X
80A4 1C5
80A5 014 ?NCXQ 8571
80A6 04E C=0 ALL
80A7 130 LDI S&X
80A8 0C3
80A9 1BC RCR 11
80AA 10E A=C ALL
80AB 1B0 POP ADR
80AC 170 PUSH ADR
80AD 130 LDI S&X
80AE 01F
80AF 3CC ?KEY
80B0 0C7 JC 80C8 +18
80B1 266 C=C-1 S&X
80B2 3EB JNC 80AF -03
80B3 27A C=C-1 M
80B4 3CB JNC 80AD -07
80B5 3B8 READ 14(d)
80B6 1FE C=C+C MS
80B7 337 JC 809D -1A
80B8 149
80B9 0A4 ?NCXQ 2952
80BA 3F8 READ 15(e)
80BB 144 CLRF 6
80BC 2E6 ?C#0 S&X
80BD 3BD
80BE 0A9 ?CXQ 2AF7
80BF 0DB
80C0 08C ?NCXQ 2337
80C1 3F8 READ 15(e)
80C2 226 C=C+1 S&X
80C3 14C ?FSET 6
80C4 013 JNC 80C6 +02
80C5 046 C=0 S&X
80C6 3E8 WRIT 15(e)
80C7 2CB JNC 80A8 -27
80C8 220 C=KEY KY
80C9 37A ?A#C M
80CA 07F JC 80D9 +0F
80CB 130 LDI S&X
80CC 3C0
80CD 3C8 CLRKEY
80CE 3CC ?KEY
80CF 71A
80D0 002 ?NCGO 80C7
80D1 266 C=C-1 S&X
80D2 3DB JNC 80CD -05
80D3 181
80D4 070 ?NCXQ 1C6C
80D5 03C
80D6 261
80D7 000 ?NCXQ 0098
80D8 273 JNC 80A6 -32
80D9 11A A=C M
80DA 01C R= 3
80DB 1E2 C=C+C AR
80DC 127 JC 8100 +24
80DD 05A C=0 M
80DE 05C R= 4
80DF 050 LDAR 1
80E0 1DA A=A-C M
80E1 31A ?A<C M
80E2 023 JNC 80E6 +04
80E3 08A A<>C M
80E4 170 PUSH ADR
80E5 38B JNC 80D6 -0F
80E6 210 LDAR 8
80E7 1DA A=A-C M
80E8 319 ?A<C M
80E9 3D7 JC 80E3 -06
80EA 05C R= 4
80EB 110 LDAR 4
80EC 010 LDAR 0
80ED 1DA A=A-C M
80EE 05C R= 4
80EF 050 LDAR 1
80F0 31A ?A<C M
80F1 018 JNC 80F4 +03
80F2 08A A<>C M
80F3 05B JNC 80FE +0B
80F4 1DA A=A-C M
80F5 31A ?A<C M
80F6 018 JNC 80F9 +03
80F7 08A A<>C M
80F8 020 JNC 80FB +05
80F9 05C R= 4
80FA 002 A=0 AR
80FB 08A A<>C M
80FC 1FA C=C+C M
80FD 1FA C=C+C M
80FE 1FA C=C+C M
80FF 328 JNC 80E4 -18
8100 3B8 READ 14(d)
8101 2FC RCR 13
8102 3D8 C<>ST XP
8103 00C ?FSET 3
8104 018 JNC 8107 +03
8105 004 CLRF 3
8106 013 JNC 8108 +02
8107 008 SETF 3
8108 3D8 C<>ST XP
8109 33C RCR 1
810A 3A8 WRIT 14(d)
810B 171
810C 01C ?NCXQ 875C
810D 248 JNC 80D6 -37
```

```
8110 094 "T"
8111 006 "F"
8112 005 "E"
8113 00C "L"
8114 130 LDI S&X
8115 010
8116 270 RAM SLCT
8117 130 LDI S&X
8118 0FD
8119 3F0 PRPH SLCT
811A 000 NOP
811B 3F8 READ 15(e)
811C 04E C=0 ALL
811D 3F0 PRPH SLCT
811E 270 RAM SLCT
811F 3E0 RTN
```

```
8120 085 "E"
8121 013 "S"
8122 00F "O"
8123 00F "O"
8124 007 "G"
8125 3C1
8126 0D8 ?NCXQ 2CF8
8127 130 LDI S&X
8128 02C
8129 3A8 WRIT 14(d)
812A 14D
812B 024 ?NCXQ 0953
812C 3E0 RTN
```

```
812D 0B1 "1"
812E 02B "+"
812F 04E C=0 ALL
8130 2A0 SETDEC
8131 270 RAM SLCT
8132 23A C=C+1 M
8133 3CC ?KEY
8134 3F3 JNC 8132 -02
8135 130 LDI S&X
8136 009
8137 10E A=C ALL
8138 35C R= 12
8139 1A6 A=A-1 S&X
813A 3FA LSHFA M
813B 342 ?A#0 AR
813C 3EB JNC 8139 -03
813D 0AE A<>C ALL
813E 0E8 WRIT 3(X)
813F 3C8 CLRKEY
8140 3CC ?KEY
8141 3F7 JC 813F -02
8142 3E0 RTN
```

```
8149 08C "L"
814A 006 "F"
814B 014 "T"
814C 005 "E"
814D 013 "S"
814E 005 "E"
814F 012 "R"
8150 04E C=0 ALL
8151 270 RAM SLCT
8152 29C R= 7
8153 000 LD@R 3
8154 350 LD@R D
8155 05C R= 4
8156 110 LD@R 4
8157 210 LD@R 8
8158 3A8 WRIT 14(d)
8159 3E0 RTN
```

```
816F 092 "R"
8170 110 "P"
8171 114 "T"
8172 000 NOP
8173 0A6 A<>C S&X
8174 38D
8175 05A ?NCGO 16E3
```

```
8180 093 "X"
8181 00E "N"
8182 014 "T"
8183 0F8 READ 3(X)
 184 38D
8185 008 ?NCXQ 02E3
8186 38D
8187 05A ?NCGO 16E3
```

```
818A 000 "0"
818B 035 "5"
818C 006 "F"
818D 013 "S"
818E 3A8 READ 14(d)
818F 3D8 C<>ST XP
8190 088 SETF 5
8191 3D8 C<>ST XP
8192 3A8 WRIT 14(d)
8193 3E0 RTN
```

```
81E0 094 "T"
81E1 001 "A"
81E2 006 "F"
81E3 004 "D"
81E4 010 "P"
81E5 015 "U"
81E6 0F8 READ 3(X)
81E7 046 C=0 S&X
81E8 226 C=C+1 S&X
81E9 18C RCR 11
81EA 330 FETCH S&X
81EB 186 A=C S&X
81EC 226 C=C+1 S&X
81ED 040 WROM
81EE 03C RCR 3
81EF 0A6 A<>C S&X
81F0 226 C=C+1 S&X
81F1 1E6 C=C+C S&X
81F2 18C RCR 11
81F3 046 C=0 S&X
81F4 18E A=C ALL
81F5 0F8 READ 3(X)
81F6 05E C=0 MS
81F7 05A C=0 M
81F8 31C R= 1
81F9 0EA C<>B R<
81FA 04A C=0 R<
81FB 23C RCR 2
81FC 21A C=C+A M
81FD 040 WROM
81FE 04E C=0 ALL
81FF 0CA C=B R<
8200 23A C=C+1 M
8201 21A C=C+A M
8202 040 WROM
8203 046 C=0 S&X
8204 23A C=C+1 M
8205 040 WROM
8206 23A C=C+1 M
8207 040 WROM
8208 3E0 RTN
```

```
01*LBL "+1R"
02 1
03 ENTER↑
04 ENTER↑
05 ENTER↑
06 "PUSH R/S"
07 PROMPT
08*LBL 01
09 +
10 GTO 01
11 END
```

```
01*LBL "DEMO"
02*LBL 10
03 GOOSE
04 SF50
05*LBL 01
06 LEFT
07 0
08 GTO 01
09*LBL 11
10 AVIEW
11*LBL 02
12 LEFT
13 0
14 GTO 02
15*LBL 12
16 SF50
17*LBL 03
18 LEFT
19 0
20 GTO 03
21*LBL 13
22 GOOSE
23*LBL 04
24 0
25 GTO 04
26*LBL 14
27 0
28 GTO 14
29 END
```

```
01*LBL "VER?"
02 "A+"
03 RCL (
04 ENTER↑
05 ENTER↑
06 ENTER↑
07 "+"
08 RCL [
09 "ROM 012: "
10 XEQ 01
11 XEQ 01
12 XEQ 01
13 AVIEW
14 RTN
15*LBL 01
16 ENTER↑
17 ROM>X
18 BIN>BCD
19 64
20 +
21 X>A
22 CLX
23 RCL T
24 RDN
25 X+Y
26 RTN
27 END
```

SYNTHETIC TEXT LIN

```
02  F2 E0 00
07  F2 8F FE
```

```
280 098 "X"
288 086 "F"
290 083 "C"
298 244 CLRF 9
29E 02B JNC 8293 +05
2A0 098 "X"
2A8 086 "F"
2B0 013 "S"
2B8 248 SETF 9
2BE 0F8 READ 3(X)
2C4 38D
2CC 008 ?NCXQ 82E3
2D4 06E A=0 ALL
2DC 106 A=C S&X
2E4 130 LDI S&X
2EC 038
2F4 306 ?A<C S&X
2FC 381
304 00A ?NCGO 82EB
30C 130 LDI S&X
314 008
31C 0EE C<>0 ALL
324 04E C=0 ALL
32C 270 RAM SLCT
334 22E C=C+1 ALL
33C 23C RCR 2
344 186 A=A-B S&X
34C 3F3 JNC 82A3 -02
354 013 JNC 82A8 +02
35C 1EE C=C+C ALL
364 166 A=A+1 S&X
36C 3F3 JNC 82A7 -02
374 0EE C<>0 ALL
37C 3B8 READ 14(d)
384 0AE A<>C ALL
38C 24C ?FSET 9
394 02B JNC 82B3 +05
39C 0EE C<>0 ALL
3A4 370 C=C OR A
3AC 000 NOP
3B4 023 JNC 82B6 +04
3BC 0CE C=0 ALL
3C4 2AE C=-C-1 ALL
3CC 380 C=C AND A
3D4 3A8 WRIT 14(d)
3DC 149
3E4 05A ?NCGO 1652
```

BYTE LISTING OF -YER?-

```
8240 009 "α"
8241 270 "ρ"
8242 1C6 "F"
8243 000 "+"
8244 0F5 "u"
8245 000 "+"
8246 056 "V"
8247 045 "E"
8248 052 "R"
8249 03F "?"
824A 1F2 "r"
824B 010 "0"
824C 000 "+"
824D 190 "0"
824E 075 "u"
824F 183 "+"
8250 183 "+"
8251 183 "+"
8252 1F2 "r"
8253 00F "+"
8254 0FE "Σ"
8255 190 "0"
8256 075 "u"
8257 1F9 "y"
8258 052 "R"
8259 04F "O"
825A 04D "M"
825B 020 " "
825C 030 "0"
825D 031 "1"
825E 032 "2"
825F 03A ":"
8260 020 " "
8261 1E0 "`"
8262 00A ""
8263 081 "x"
8264 1E0 "`"
8265 007 "↓"
8266 081 "x"
8267 1E0 "`"
8268 004 "α"
8269 001 "x"
826A 17E "Σ"
826B 185 "δ"
826C 182 "X"
826D 183 "+"
826E 1A5 "%"
826F 062 "b"
8270 1A5 "%"
8271 049 "I"
8272 116 "ä"
8273 014 "ä"
8274 140 "@"
8275 1A5 "%"
8276 047 "G"
8277 177 "w"
8278 190 "0"
8279 070 "p"
827A 175 "u"
827B 1A5 "%"
827C 06A "j"
827D 185 "δ"
827E 1C8 "H"
827F 000 "∧"
8280 22F "/"
```

BYTE LISTING OF -+1R-

```
8220 004 "α"
8221 270 "ρ"
8222 1C6 "F"
8223 000 "+"
8224 0F4 "t"
8225 000 "+"
8226 02B "+"
8227 031 "1"
8228 052 "R"
8229 111 "Q"
822A 183 "+"
822B 183 "+"
822C 183 "+"
822D 1F8 "x"
822E 050 "P"
822F 055 "U"
8230 053 "S"
8231 048 "H"
8232 020 " "
8233 052 "R"
8234 02F "/"
8235 053 "S"
8236 18E "♪"
8237 182 "X"
8238 140 "@"
8239 182 "2"
823A 004 "α"
823B 1C8 "H"
823C 003 "+"
823D 22F "/"
```

BYTE LISTING OF -DEMO-

```
81A0 008 "∧"
81A1 210 "0"
81A2 1C4 "D"
81A3 001 "+"
81A4 0F5 "u"
81A5 000 "+"
81A6 044 "D"
81A7 045 "E"
81A8 04D "M"
81A9 04F "O"
81AA 10B "λ"
81AB 1A4 "$"
81AC 0C3 "C"
81AD 1A4 "$"
81AE 0C8 "H"
81AF 102 "X"
81B0 1A4 "$"
81B1 0C2 "B"
81B2 110 "0"
81B3 1B2 "2"
81B4 006 "Γ"
81B5 19C "ν"
81B6 17E "Σ"
81B7 183 "+"
81B8 1A4 "$"
81B9 0C2 "B"
81BA 110 "0"
81BB 1B3 "3"
81BC 006 "Γ"
81BD 18D "∠"
81BE 1A4 "$"
81BF 0C8 "H"
81C0 184 "α"
81C1 1A4 "$"
81C2 0C2 "B"
81C3 110 "0"
81C4 1B4 "4"
81C5 006 "Γ"
81C6 18E "♪"
81C7 1A4 "$"
81C8 0C3 "C"
81C9 185 "δ"
81CA 110 "0"
81CB 185 "5"
81CC 004 "α"
81CD 18F "∔"
81CE 110 "0"
81CF 1BF "?"
81D0 004 "α"
81D1 1CA "J"
81D2 006 "Γ"
81D3 22F "/"
```

APPENDIX 3

Other sources of information

This Machine Language Interface is based upon the Machine Language Development Laboratory designed by Lynn A. Wilkins (7344). This design was published in the PPC Calculator Journal V 9 N 3. This article gives more details on the construction of the MLDL, although this is intended for use with a wire-wrapped board, as opposed to the printed wiring board type of construction used for the MLI.

The available literature on the subject of microcode, is very extensive, but almost inaccessible outside the major programmable calculator user group in the world, PPC. This group was formed in 1974 when the first handheld programable calculator, the HP-65, was released by Hewlett-Packard. It was then known as the 65 User's club, changed its name to the PPC Club in 1978, and was incorporated in 1982. Amongst a wide range of other activities, it publishes what was originally called 65 Notes, but became the PPC Calculator Journal. There was no information on microcode (the name originally given by HP to the various assembly language, or machine codes of their generations of hand held calculators, both programmable and non-programmable) outside of the pages of the PPC Journal (and the 65 Notes), until 1980. The Melbourne Chapter of the PPC Club, PPC Melbourne, started the publication of its own user's publication, **PPC Technical Notes**, in 1979, and has printing information on the subject from 1980 on.

The PPC Journal is available only to members of the PPC Club, at an annual subscription which varies around the world. For a sample copy, and membership application forms, write to:

PPC, 2545 West Camden Place, Santa Ana, California, U.S.A. 92704,

enclosing an A4 sized self-addressed stamped envelope (two ounces, airmail, or International stamp vouchers for the appropriate amount). The same material is available from PPC Melbourne.

Subscriptions to PPC Technical Notes are A$20 annually, and back issues are available at A$10 per set, plus postage. (Airmail rates on back issues run to well over A$10, depending on the destination.) Send Bank Cheques or money orders, made payable to PPC Melbourne, to:

R.M. Eades, Box 15, Hampton, Victoria, Australia 3188

Only the essential references are given here. For further reading relating to the subject, see back issues of the PPC Journal, and PPC Technical Notes. Anyone wanting to program in the HP-41C/CV microcode should have a good grasp of the operating system of that machine. Suitable texts are:

Wickes, W.C. Synthetic Programming on the HP-41c.
(Larken Publications, 4517 NW Queens Ave., Corvallis, Oregon, U.S.A. 97330.)

Jarett, K. HP-41 Synthatic Programming Made Easy.
(SYNTHETIX, 1540 Mathaws Ave., Manhattan Beach, California, U.S.A., 90266.)

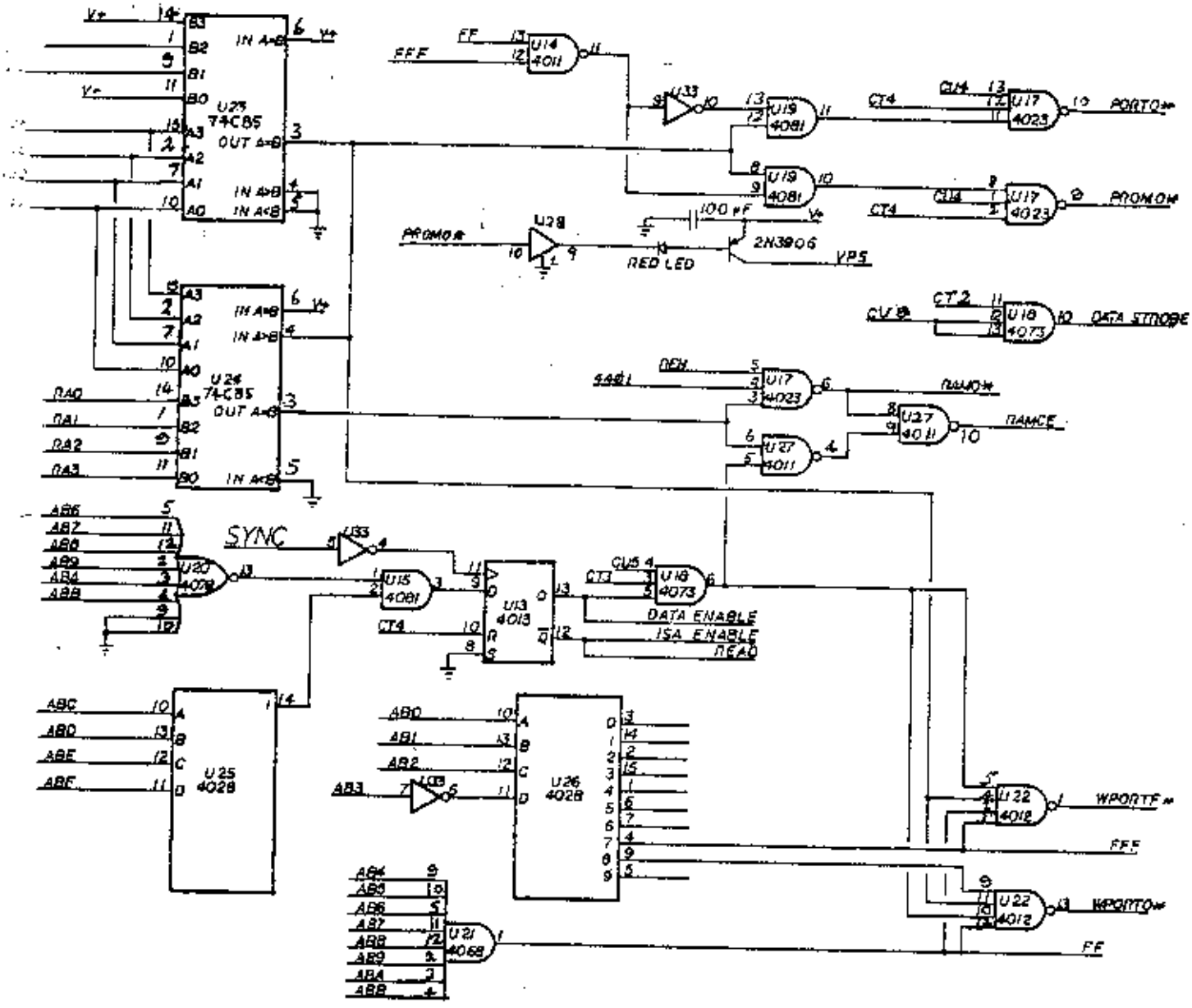
Dodin, J-D. Au Fond de la HP-41c. (In French.)
(J-D. Dodin, 77 Rue du Cagire, 31100 Toulouse, France.)

The latter is (to date) the only publication to give an elementary introduction to the operating system of the HP-41c, and to the subject of synthetic programming, a knowledge of which is essential to serious microcode work, outside the walls of Hewlett-Packard.

APPENDIX 4

CONTROL SIGNALS GENERATOR

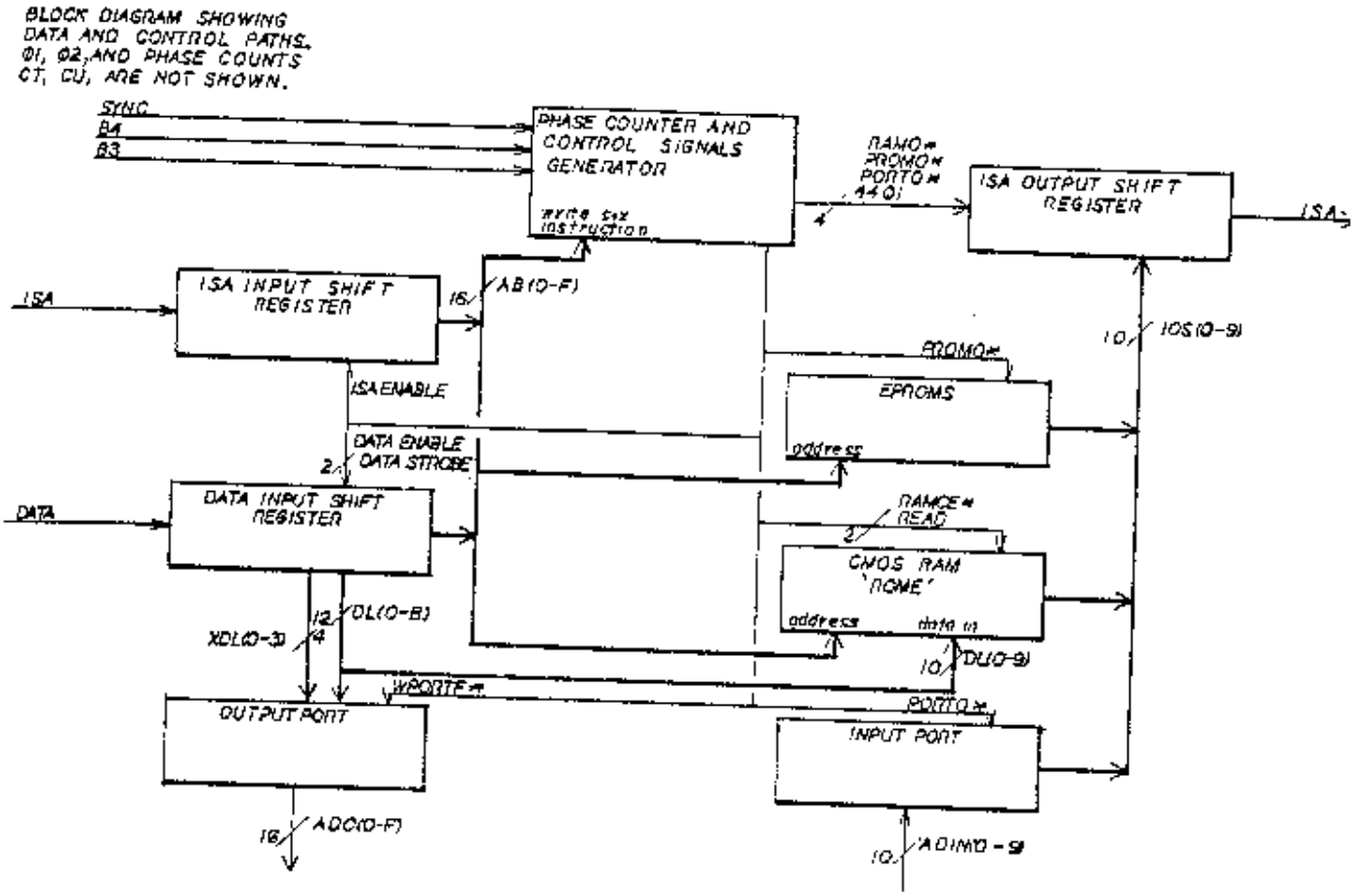
(After EPROM Power Supply modifications.)

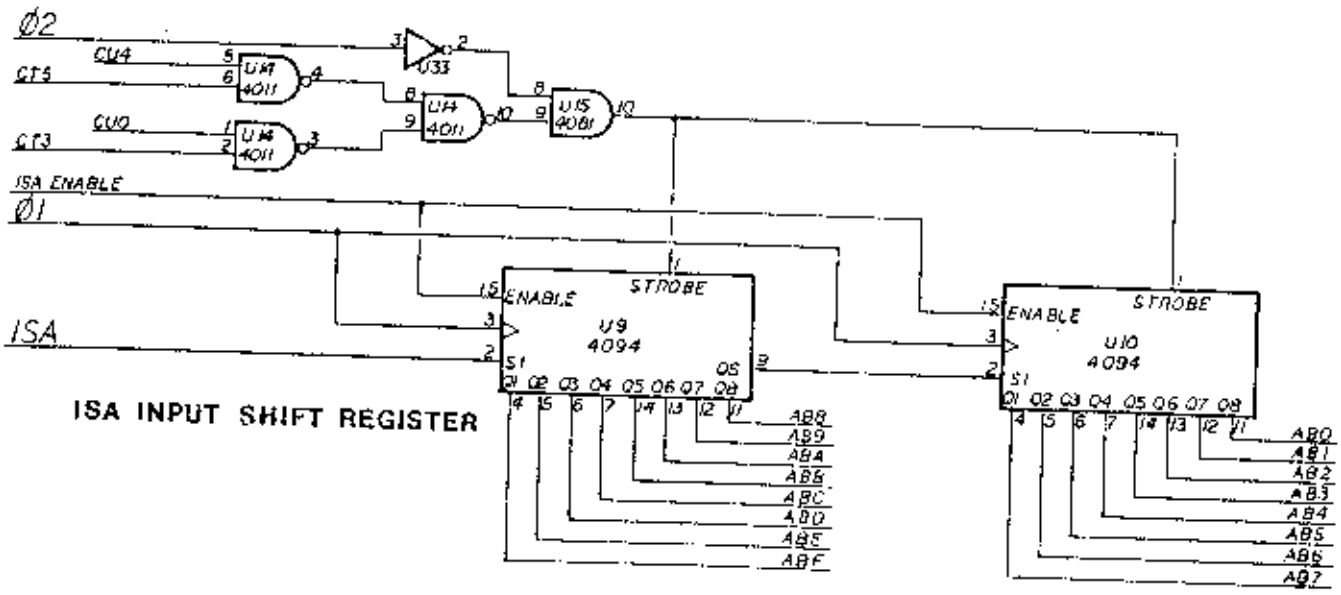


Microbaud

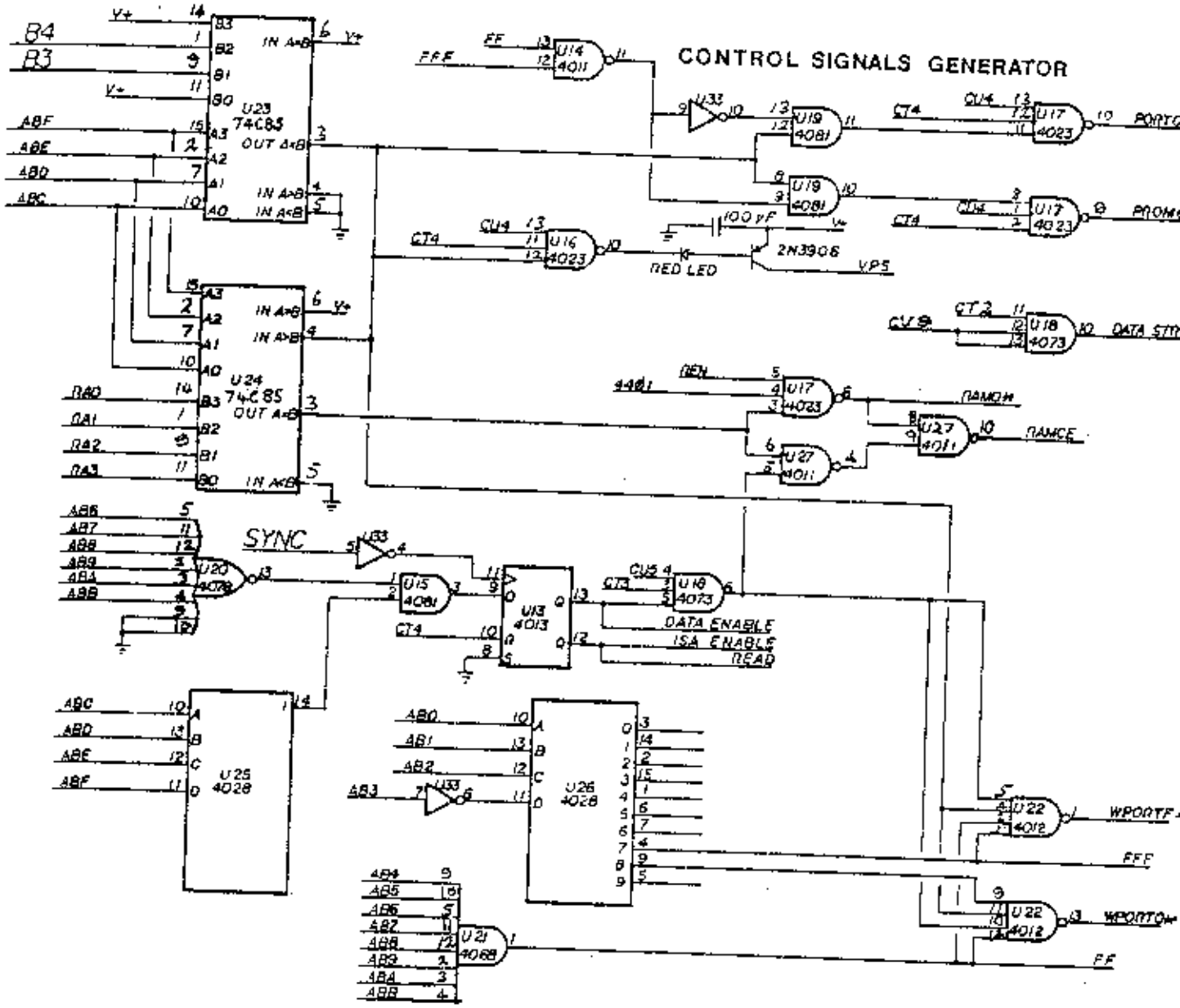
Developments

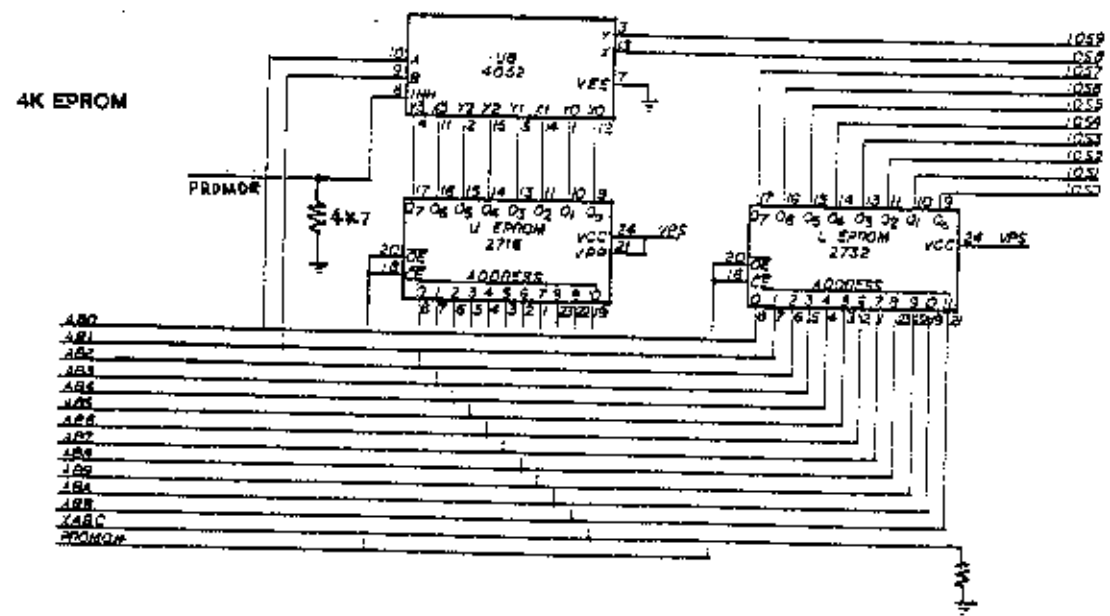
MACHINE LANGUAGE INTERFACE

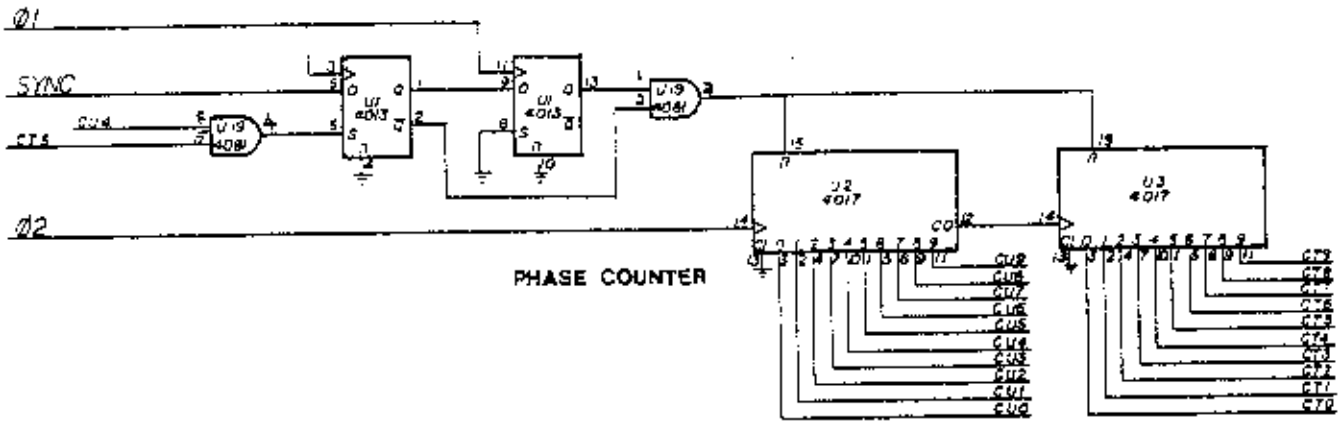




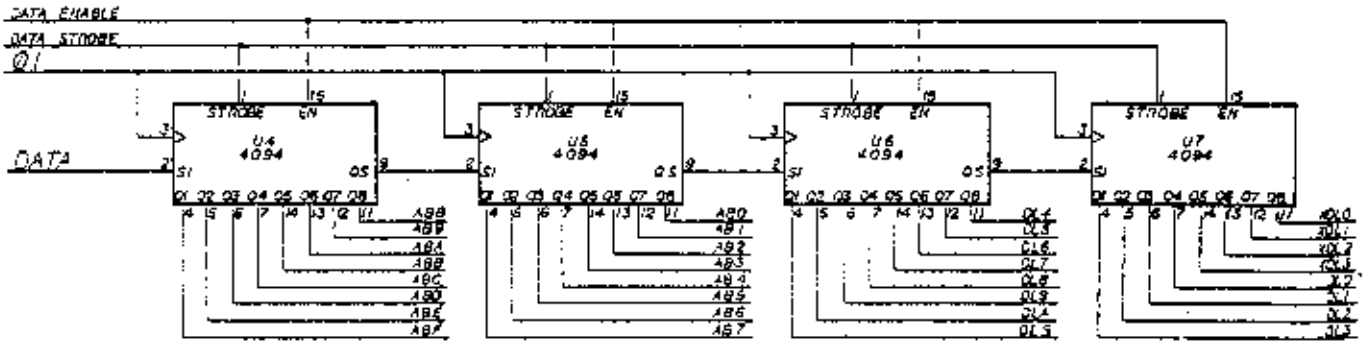
ISA INPUT SHIFT REGISTER

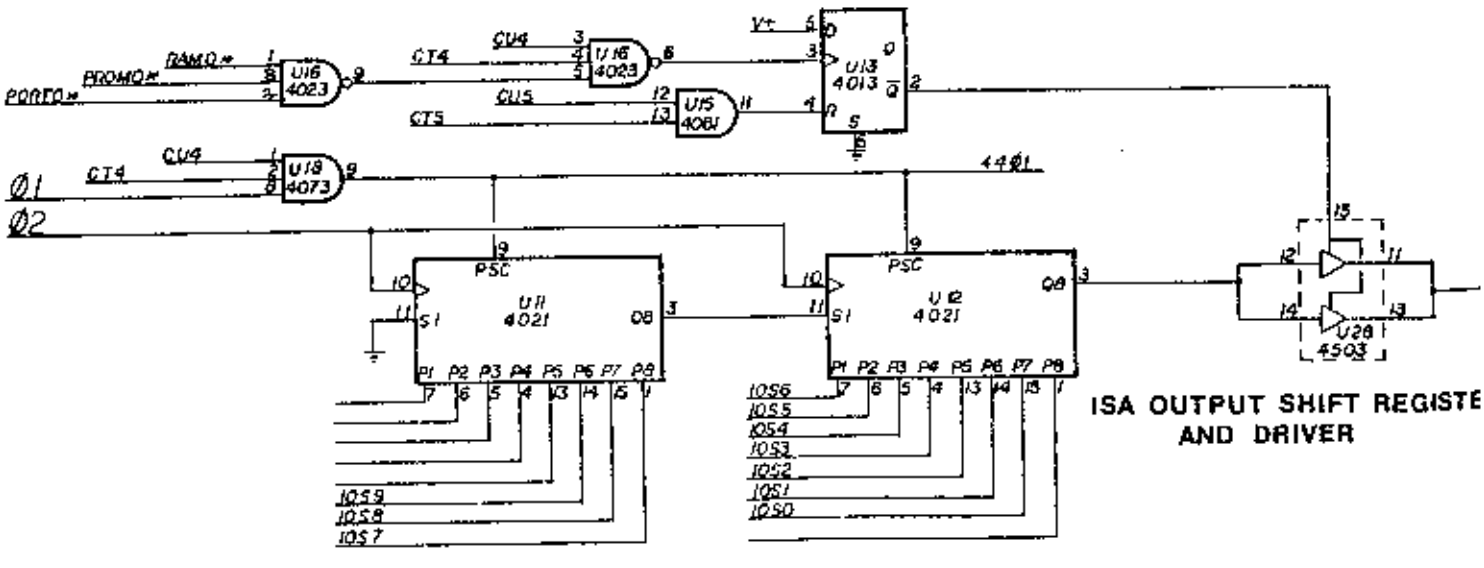

CONTROL SIGNALS GENERATOR
B4
B3
U23 74C85
U24 74C85
IN A>B
OUT A>B
IN A<B
ABF
ABE
ABD
ABC
DA0
DA1
DA2
DA3
SYNC
U20 4078
U15 4081
U3 4013
U16 4023
U14 4011
U33
U19 4081
U17 4023
U18 4073
U27 4011
U22 4012
U25 4028
U26 4028
U21 4068
100 pF
2N3906
RED LED
VPS
DATA ENABLE
ISA ENABLE
READ
PROMOE
PROMCE
WPORTF
WPORTG
FFF
FF


4K EPROM
U8 4032
U EPROM 2716
L EPROM 2732
4K7
PROMOE
ADDRESS
VPS



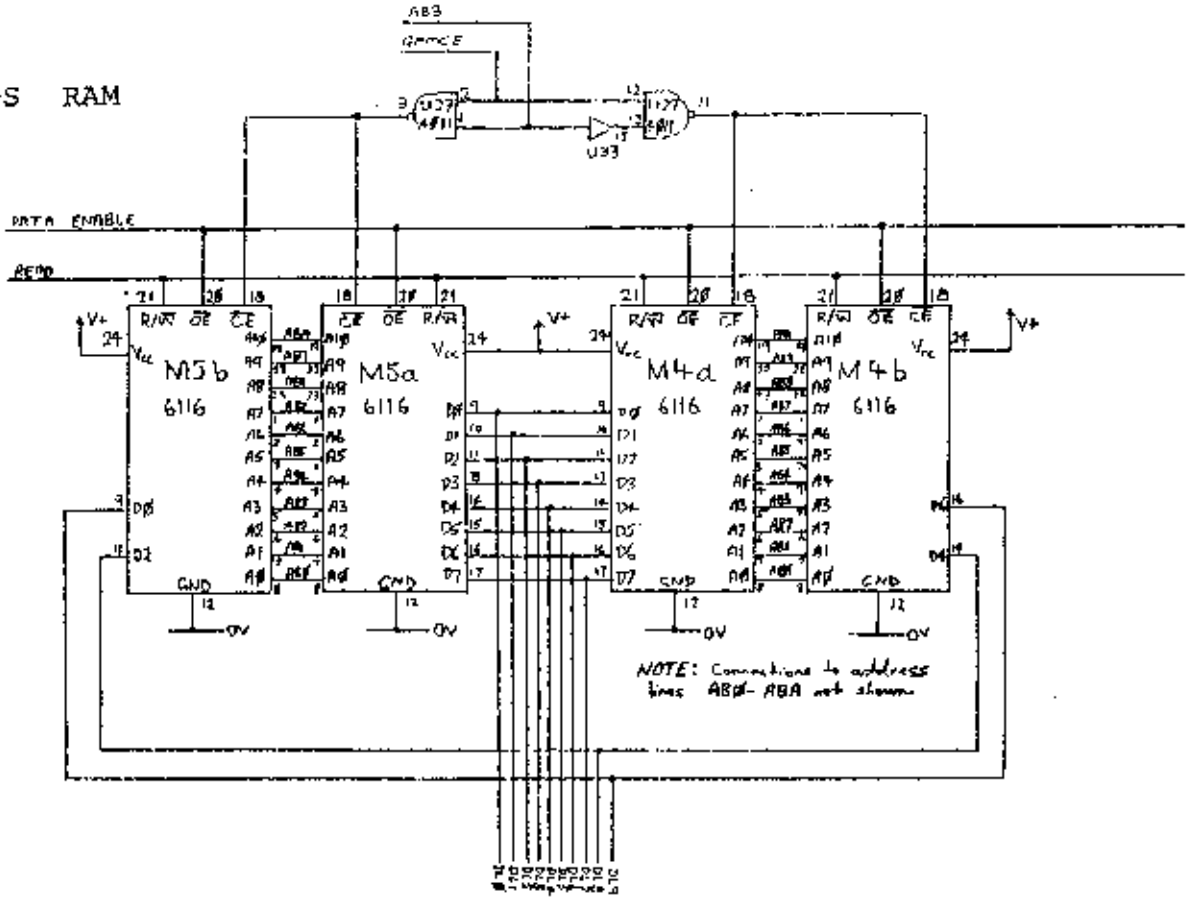
DATA INPUT SHIFT REGISTER



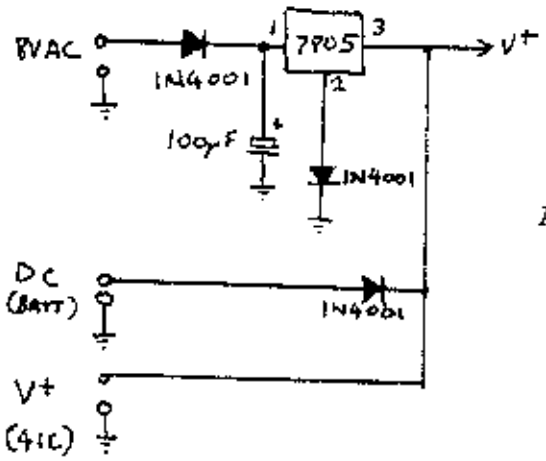


ISA OUTPUT SHIFT REGISTE
AND DRIVER

4K CMOS RAM

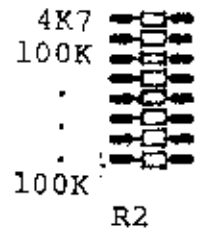


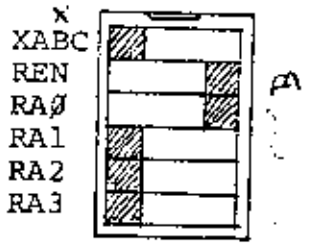
BITS AND PIECES

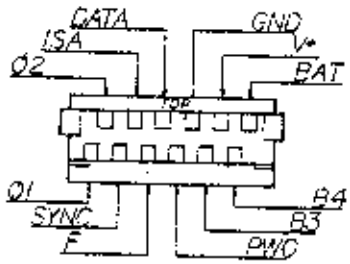


POWER SUPPLY

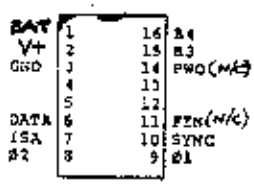
R1
ALL 100K
5% ¼W



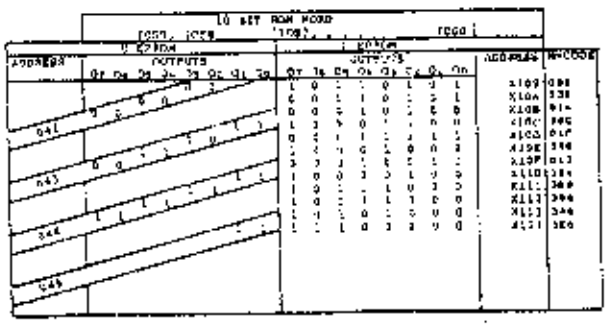




CABLE CONNECTOR



I/O B



EPROM FORMAT

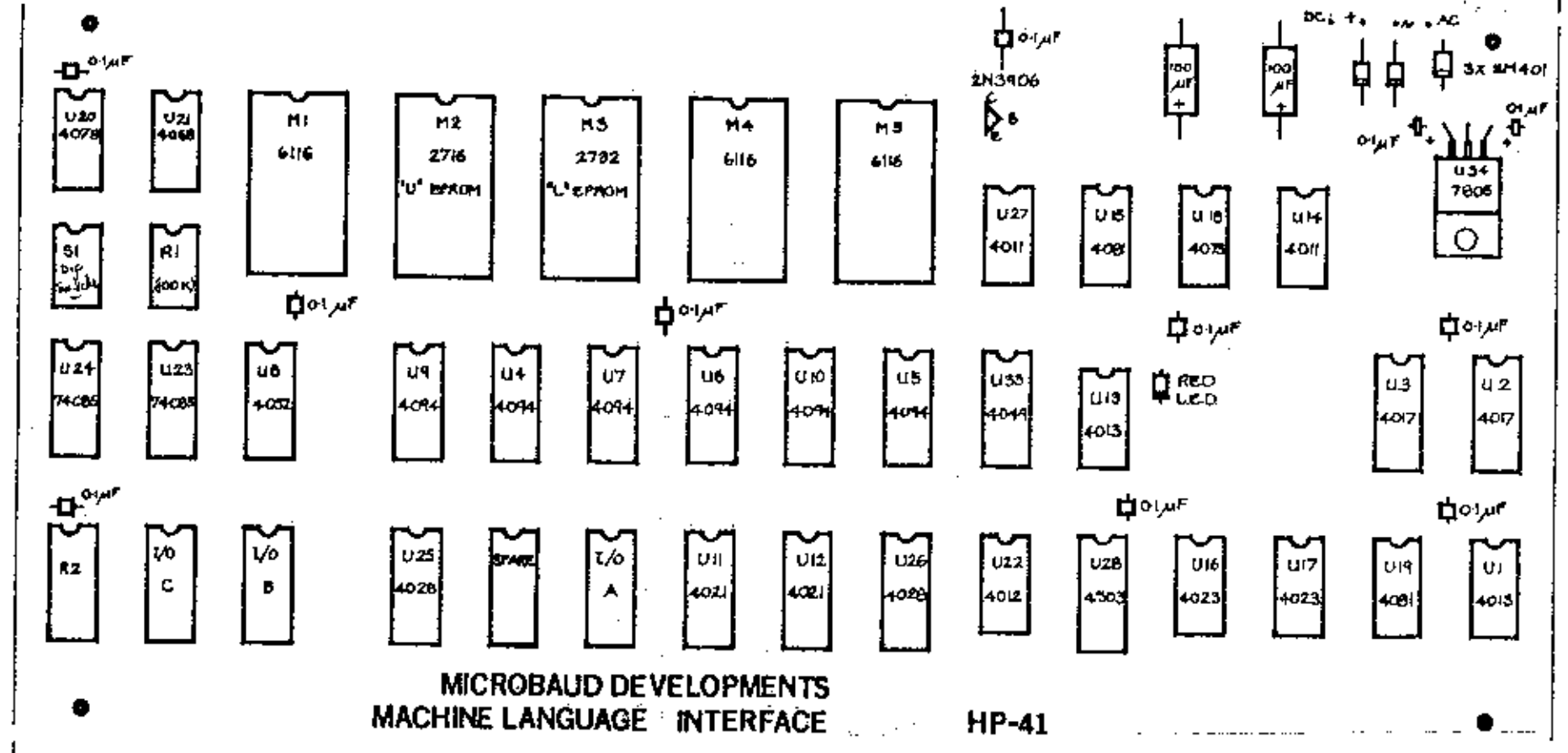


Fig. 1

'ML1' STUFFING DIAGRAM

HP-41 MACHINE LANGUAGE DEVELOPMENT LAB - MLDL

INTRODUCTION TO THE MLDL

I built the Machine Language Development Lab for a toy so that I could play with the inner language of the HP-41. When it turned out that other members of PPC also wanted to toy with the calculator's inner secrets (surprise, surprise!) I decided to release the design for noncommercial use. The introduction of EPROM boxes also spured the MLDL project. Some PPCers will want their very own machine code in EPROM. But it is very difficult to develop machine code without a reasonable way to test it. The MLDL is small. It is portable. It is easy to use. And it allows the immediate testing of machine language code.

There are other, more powerful, ways to develop code. For instance, Paul Lind (6157) uses a micro-computer and a dual port RAM card. The computer sees the shared RAM as a pair of I/O ports. The HP-41 sees the RAM as if it were a ROM module. Paul can use all the power of his computer to design code for the HP-41. He has an editor, assembler, disk storage, trace routines, ... the works. (A package of schematics is available for a nominal fee. For the details send a SASE to Paul c/o Puget Sound Programing, P.O. Box 5929, Seattle WA 98105)

The MLDL provides three functions. There is an EPROM array to store the special functions useful to the MLDL. There is an I/O port which is to be used with an EPROM programmer. And there is a CMOS memory array. The CMOS array is called ROME for ROM Experimental. ROME is where machine code can be loaded and tested.

The 4K words of EPROM in the current design are adequate for a small assembler and support functions. More EPROM could be added. 16K total would be no problem. But the commercial EPROM boxes are probably more convenient.

The I/O port may prove to be unnecessary when someone starts selling an EPROM programmer that runs on the HP-IL. Don Robinson (8021) tells me that he has such a device. Also, there may be some users who will run their code from the MLDL. They may never make hard copies in EPROM.

The 4K words of ROME is a lot. Most machine code routines will be one hundred to two hundred words long. There is room for a lot of programing in four thousand words!

The MLDL is a convenient, inexpensive tool to help write and test HP-41 Machine Language code. There is probably only a handful of PPC members outside of HP who have been able to run their own machine code. Well, now is your chance too.

MACHINE CODE, MICRO-CODE, OR M-CODE?

Let me identify the three types of code used in the HP-41.

- USER CODE
- TRANSLATED USER CODE
- MACHINE CODE

User Code is what HP describes in the HP-41C/CV user manual. It is the code that sane, rational, ordinary calculator users use. User Code will satisfy 99% of user needs. Synthetic Programming, since it has been accepted by HP, is User Code that will satisfy another 0.9% of user needs. User Code is good, useful stuff.

Translated User Code is one type of code usable in ROMs and EPROMs. T-Code is User Code which has been modified to fit the ten bit ROM word format. The PPC ROM was initially User Code. It was part of HP's service when producing the custom ROM to provide the modification. T-Code can be identified as ROM code which can be COPYed into user memory.

Machine Code is the other type of code usable in ROMs and EPROMs. Machine Code is the native language of the micro-processor used within the HP-41C/CV. The code is not properly called micro-code. Micro-code or 'micro-programme' is a sequence of operations used to control that part of a computing "...machine which supplies the pulses for operating the gates associated with the arithemitcal and control registers." (M.V. Wilkes, The Best Way To Design An Automatic Calculating Machine.) Calling Machine Code 'micro-code' is wrong. But almost everybody does it. I shall use the compromise term M-Code.

UNDERSTANDING THE MLDL

This is a 'broad brush' description of how the MLDL works, just as the Block Diagram is a large scale picture. Details of the individual blocks are given in the BUILDING AND CHECK-OUT section of this article. The necessary background to understand the MLDL/HP-41 interface is in two PPC Journal articles, DeArras' PPC V7N3 and McClellan's PPC V6N6.

It is all very simple. The purpose of the logic chips is to shift out the proper bits at the proper time.

The ten bit M-Code words which are shifted out the ISA line to the HP-41 originate in the EPROM, ROME, or the Input Port. The Address Bus lines A#(C-F) determine which is selected. During a machine cycle sixteen bits of ROM address is shifted out of the HP-41 on the ISA line. The Isa Input Shift Register takes in the address bits and latches them. ISA ENABLE gates the

captured address onto the Address Bus. Comparator circuits in the Control Signals Generator look for a match with either the HPROM address, the ROME address, or the Input Port address. If a match is detected one of the control signals RAMO*, PROMO*, or PORTO* is asserted and at the proper time the Isa Output Shift Register is loaded with the ten bit M-Code which is on IOS(0-9). Then the word is shifted out. The source of the M-Code is enabled by the control signals RAMCE* ANDed with READ, or PROMO*, or PORTO*.

It is obvious how the ten bit Machine Code words get into the EPROM and the Input Port. The ROME isn't so obvious. The ROME is loaded in response to a special WRITE S&X instruction, 040. The instruction is a NOP to the HP-41 so it does nothing during the following machine cycle. The MLDL interprets the instruction and uses the DATA bits as a source of address and code word. The MLDL writes the code word at the address. The C register out of the HP-41 CPU is repeatedly latched into the Data Input Shift Register. It is latched at the phase where the least significant bit of C becomes DL0. Thus the lowest ten bits of C become the ten bit word loaded into the ROME. Every instruction out of the HP-41 is latched into the Isa Input Shift Register and sent via the Address Bus AB(0-F) to the Control Signals Generator. The WRITE S&X is decoded and the most significant bits of the Data Input Shift Register are gated onto the Address Bus AB(0-F). Data Lines DL(0-9) are also gated to the ROME inputs. READ is not asserted, a write, and when RAMCE* is asserted the ten bit word on DL(0-9) is written into the ROME.

BUILDING AND CHECK-OUT

It is not difficult to build the MLDL. The builder should have some experience constructing electronic projects. If you intend to build the MLDL yourself, 'perfboard,' 'wire wrap,' and 'bypass cap' should be familiar terms. If you don't know which side of an IC is up, find somebody who does. A familiarity with digital logic will almost guarantee a working MLDL. Most of the ICs used are detailed in the RCA COS/MOS Integrated Circuits Data Book, SSD-250B. The details may also be in data books from other IC manufacturers.

Let's establish some ground rules. ICs are identified by the U number or generic part number. For example, U16 is a 4029. Particular IC pins are identified by appending the pin number to the associated U number. Pins 2 and 4 of U16 are called out as U16-2,4. Signal names are in capital letters. For example, ISA or PROMO*. The associated function is logically asserted when the signal voltage is high. But, if the name is followed by the asterisk the function is asserted when the signal voltage is low. PROMO* is asserted when U16-10 is a low voltage. For a signal group I may use the short notation AB or AB(0-F) instead of ABO thru ABF.

The MLDL was designed for easy modular construction. An oscilloscope will facilitate check-out, but should not be necessary. A simple logic probe will suffice (unless you've really got your wires crossed.) The instructions proceed in sections. Each section will call for the wiring and check-out of a modular block of circuits. The check-out also goes into the 'how and why' of each block's operation. Because some ICs are used in more than one block not all the inputs to the IC will be wired. For example, U13 is used in the Isa Output Shift Register and the Control Signals Generator. Wire the unused inputs of all ICs to ground. I use a contrasting wire color for this. It helps in finding the inputs later. Install only the ICs which are wired. Proceeding block-by-block will help catch miswires before they are hopelessly buried. The approach also provides positive (I hope) reinforcement and confidance.

Each section calls for:

1. Organizing the parts layout.
2. Wiring the circuits together as shown on the schematic.
3. Double checking the wiring. Make sure the unused inputs are tied low.
4. Installing the ICs.
5. Verifying the block's operation.

Steps 1 and 5 are the thinking steps. The steps between demand care and attention. The most likely problems will be miswires or shorts. The next most likely problem is a misunderstanding of how a block is supposed to operate. The block may work perfectly but you might think otherwise. Talk it over with a friend. The last thing to assume is bad parts. I do know how very frustrating it can be to search out the causes of problems. I sympathize with your difficulty when the verify step doesn't, but it is impractical for me to personally aid you. Do not call me. You are on your own. I would appreciate a letter describing in detail any problem you may have when building the MLDL. I will keep a record of such problems and if common problems develop I will try to assess the causes and publish fixes. I would also like to read about your successes.

Section 1. The Plug. The plug which connects the HP-41 to the MLDL is vital to the success of the project. The finished connector/cable assembly must be rugged. You will be plugging and unplugging the MLDL often. I recommend and old memory module which can be opened at the seam. Unsolder and throw out the circuit board. Use ribbon cable or single wires to connect to the MLDL. I recommend a DIP plug at the MLDL end. I used a sixteen pin plug into a wire wrap socket on the prototype. The socket had the following pin outs.

V+	1	16	B4
VCC	2	15	B3
GND	3	14	PWO
	4	13	
	5	12	
DATA	6	11	PIN
ISA	7	10	SYNC
Ø2	8	9	Ø1

The length of the cable is important. Don't try to compete with Ma Bell. Be warned that the longer the wires the more likely will be electro-static caused crashes. The HP-41s that have been used with the prototype MLDL had no trouble driving eighteen inches of wire. The verification of this section is to double check the wiring for continuity and mechanical integrity. Be sure to identify the wires at the MLDL end. Glue the memory module back together and set the cable assembly aside.

Section 2. The ROME battery and the EPROM power switch. ODDS AND ENDS shows a battery and diode circuit. The batteries are there to maintain the CMOS memory while the MLDL is not connected to the calculator. Three AA cells should keep the MLDL alive for the shelf life of the batteries. Replace them when the total voltage to the memories drops to 2.1 volts. The 100 micro-farad capacitor will supply the MLDL while you change the batteries. Just don't take all day. The Control Signals Generator shows a VPS control circuit comprising a 2N3906 transistor, a red light-emitting-diode and the 100 micro-farad capacitor. VPS is the power for the EPROMs. The transistor is the switch which turns the VPS on for the time that the EPROMs are used. Because VPS is on for only one phase time, speeded up HP-41s may not work with the MLDL. Without the switch the EPROMs would consume hundreds of milli-amps and overload the calculator's power supply. The red LED provides a certain voltage drop which helps match the output voltage/current characteristic of the CMOS gate, U17, with the drive requirements of the 2N3906. Any simple red LED will work. I used a MV50 type. By the way, don't expect to see the LED light whenever the EPROM is accessed. If it does light, something is wrong. The battery and VPS control circuits are simple enough to verify by simply checking the continuity, checking the diode polarity, and checking the emitter-base-collector connections on the transistor.

Section 3. The Phase Counter. Recall that the HP-41 uses a fifty-six phase machine cycle. The Phase Counter maintains synchronism between the MLDL and the HP-41. For example, ANDing the output of the Tens Counter CT5, U3-1, with the output of the Units Counter CU4, U2-10, generates a pulse in synch with the fifty-fourth phase. The pulse is used to reset the Phase Counter whenever a SYNC is missing. The HP-41 generates a SYNC only when fetching a ROM instruction opcode. SYNC is suppressed when the HP-41 fetches an immediate operand or executes a FETCH S&X. The last falling edge of SYNC resets the phase counter and forces synchronism with the calculator. Note that the phase numbers used in the MLDL are shifted from those used in the DeArras article. It is a good exercise to determine which way. Double check the wiring of U1, U2, U3, and U19 (This is the last warning about the unused inputs. Tie them low.) The time has come to connect the MLDL to the HP-41. Don't expect a lot yet. Press and hold down the ON key. This will keep the calculator generating clock pulses. You may wish to write an endless loop routine in User Code to do the same. Use a logic probe or oscilloscope to verify that there is a periodic reset pulse at U19-4, U1-15, and U3-15. The higher outputs

of the Tens Counter, U3-5,6,9,11, should remain low. You should also check that the HP-41 signals, Ø1, Ø2, SYNC, ISA, DATA, and VCC are there at the end of the MLDL cable assembly.

Section 4. The Data Input Shift Register. Recall that the DATA line is the contents of the HP-41 CPU's C register. U4, U5, U6, and U7 latch the serial bit stream at the phase which stops the least siginificant bit of the C register at pin seven of U7, DL0. The DATA STROBE signal is produced at U18-10. The 3-input AND gate is shown with the Control Signals Generator. You must wire U18 along with U4, U5, U6, and U7. Verify the wiring by running the calculator and checking for a periodic DATA STROBE. Enable the outputs of U4 thru U7 by temporarily wiring DATA ENABLE, high. Check that none of the outputs of U4 thru U7 are stuck high or low.

Section 5. The Isa Input Shift Register. U14 and U15 decode two strobe pulses each machine cycle. One strobe pulse latches the instruction into U9 and U10. The next strobe pulse latches the next instruction address into U9 and U10. Temporarily tie the ISA ENABLE high and verify periodic strobe pulses at U9-1 and U10-1. Also check that none of the U9 or U10 outputs are stuck high or low.

Section 6. The Control Signals Generator. Be very, very careful when wiring the control block. It is the heart of the MLDL. Mistakes here will be hard to find. U23 and U24 are comparator circuits which select the EPROM or ROME in response to the upper four Address Bus lines, AB(C-F). U23 compares AB(C-F) with the upper half of the address space of whatever port the MLDL is plugged into.

Port 1 = 9XXX
Port 2 = BXXX
Port 3 = DXXX
Port 4 = FXXX

The output of U23 is used to select the EPROM. U24 compares AB(C-F) with the ROME address switches and selects the ROME whenever there is a match. You can map the ROME into any 4K block you wish. But watch out. Blocks 0, 1, and 2 are used by the HP-41.

U20, U25, U15, and U13 decode and latch the WRITE S&X instruction. WRITE S&X is actually an HP-41 NOP, hex code 040. U20, U25, and U15 decode the instruction field from the Isa Input Shift Register and U13 latches the output, setting up the MLDL for a write to ROME sequence as described in UNDERSTANDING THE MLDL. U21 and U26 are used to decode certain of the highest sixteen addresses in the EPROM space. For example, the I/O port is at XFFF.

Connect the MLDL to Port 4 and press ON. The calculator should function since nothing more has been connected to the port signal lines. Verify that a sequence of periodic pulses are produced at U17-10 whenever ON is pressed. The EPROM output signal at U16-10, PROMO*, should be a similar sequence of negative pulses. Set the ROME address switches to the lower half of Port 4, EXXX. (RA1, RA2, and RA3 should be high. RA0 should be low.) REN must be high to enable ROME output to the calculator. Verify that U17-6, RAMO*, and U15-4, RAMCE*, produce a sequence of negative pulses similar to PROMO*. The RAM pulses will be a bit narrower if you are using a 'scope, since they are gated by Ø1.

Section 7. The Isa Output Shift Register. U11 and U12 comprise the Isa Output Shift Register. At phase forty-four the shift register is loaded with the data on the input lines, IOS(0-9). The data is shifted out to the HP-41 during the last ten phases of a machine cycle. U28 is the MLDL ISA line driver. The flip-flop, U13-2, only enables the MLDL output during the last ten phases of a machine cycle in which the EPROM, ROME, or INPUT PORT was addressed.

Wire the Isa Output Shift Register and Driver, but don't connect the output of the Driver, U28-11,13, to ISA. We are not yet ready to send things to the HP-41. Check that U13-2 pulses when ON is pressed. Ground the input pins to U11 and U12, IOS(0-9). Verify that no pulses are output at U12-3 while the calculator is running. The next verification steps take time, but they are worth it. Start with IOS0. Wire the input line high and verify a pulse out at U12-3. Rewire the input line low. Do the same test for IOS1 thru IOS9. Unwire IOS(0-9) from the ground. Connect U28-11,13 to ISA.

Section 8. The EPROM Array. The 2732 is the L EPROM and the 2716 is the U EPROM. U8 is a switch which directs the correct pair of L U EPROM outputs to IOS(0, 9). The pair is selected by the lowest address lines, AB0 and AB1. You will need to install EPROMs which contain ROM>X and X>ROM. Run a CATALOG 2. You should see a complete EPROM catalog. Assign X>ROM to a key. Use BN from the PPC ROM or the machine code CODE to transfer the alpha word FFFF000 to the X register. Key X>ROM and one pulse should be produced at U22-1, WPORT*. If you wish to install U31 and U32 you can now write to the Output Port. Check that REN is low. Use the User Code routine WROME to write to the ROME. Verify that each X>ROM asserts RAMCE* but not READ.

Section 9. The ROME. Install the 5504s with the MLDL disconnected from the HP-41 and the MLDL batteries removed. Install the MLDL batteries and connect up the HP-41. Check that the ROME address switches are still set for block E. Use the User Code routine ZROME to zero the ROME. ZROME isn't a fast routine. Enable the ROME output by switching REN high. Use VROME to view ROME contents. There should be nothing but zeros. That's it. You're done.

Use WROME and VROME to poke and peek around in the HP-41. Use WROME to load a FAT and A>X or YSS in ROME. Have fun, try some of your own M-Code!

ZROME

ZROME fills sequential ROME words with zeros.

XEQ ZROME. The first prompt is START=? Enter the hex address of the first ROME word to be zeroed. The calculator is in the alpha mode. R/S. The second prompt is END=? Enter the hex address of the last ROME word to be zeroed. R/S. Every ROME word from the start address to, and including, the end address will be zeroed.

A FFF size block takes about eight minutes on my HP-41.

Non HP functions used are X<>Y, RXL, X>ROM, CODE.

```
01*LBL "ZROME"          18 .LB
02 AON                  19 ARCL 01
03 "START=?"            20 CODE
04 PROMPT               21 RXL
05 ASTO 01              22 RXL
06 "END=?"              23 RXL
07 PROMPT               24 XOY
08 AOFF                 25 STO L
09 CODE                 26 ROM
10 "!"                  27*LBL 01
11 CODE                 28 X>ROM
12 X<>Y                 29 LASTX
13 XXL                  30 X<>Y
14 XXL                  31 X<Y?
15 XXL                  32 GTO 01
16 "8888"               33 "ROME"
17 CODE                 34 TONE 9
                        35 .END.
```

VROME and WROME

VROME displays the ROME contents.
WROME displays the ROME contents and can change ROME contents.

XEQ VROME. The prompt is "ADDRESS?" Enter the first ROME address to be displayed. R/S. The HP-41 shows the address and the contents. R/S. The next address and contents are shown. Step through the memory by pressing R/S. Select a new starting address by entering it into the alpha display. Step backwards through memory by setting flag 1 true.

XEQ WROME. The prompt is "ADDRESS?" Enter the first ROME address to be investigated. R/S. The HP-41 shows the address and contents. Pressing R/S will step through memory. Change the contents of a displayed memory word by entering the new contents into the alpha display. R/S. The address and the new contents will be displayed.

Non HP functions used are Y-X, OR, ROM>X, RXL, X>ROM, CODE, and DECODE.

```
01*LBL "WROME"          20 ASHF                 40*LBL "WROME"
02 AON                  21 ASTO 01              41 AON
03 "ADDRESS?"           22 X<>Y                 42 "ADDRESS?"
04 PROMPT               23 ROM#X                43 PROMPT
05*LBL 04               24 RCL L                44 CODE
06 FS?C 22              25 XOY                  45 ENTER^
07 CODE                 26 "10000000"           46 ENTER^
08 FC? 01               27 CODE                 47*LBL 01
```

```
 6 GTO 6:          28 OR          48 XFC 04
 7 "2"             29 DECODE      49 $10F
 8 CODE            30 ASMP        50 [illegible]
12 Y=X             31 ASTO 02     51 STO 03
13+LBL 41          32 " "         52 RDM
14 ENTER↑          33 ARCL 01     53 RXL
15 ENTER↑          34 "F "        54 RXL
16 "1000000000     35 ARCL 02     55 RXL
17 CODE            36 RDM         56 CODE
18 OR              37 AVIEW       57 OR
19 DECODE          38 RTN         58 XROM
                   39 GTO 04      59 ROM
                                  60 STO 03
                                  61 END
```

THE IC LIST

```
 2   CD4013 Dual D Flip-Flop
     U1, U13
 2   CD4017 Decoded Decimal Counter
     U2, U3
 8   CD4094 8-Bit Shift and Store Register
     U4, U5, U6, U7, U9, U10
 1   CD4052 Differential 4-channel Multiplexer
     U8
 2   CD4021 8-stage Parallel Input Shift Register
     U11, U12
 2   CD4011 Quad 2-input NAND
     U14, U27
 2   CD4081 Quad 2-input AND
     U15, U19
 2   CD4023 Triple 3-input NAND
     U16, U17
 1   CD4073 Triple 3-input AND
     U18
 1   CD4078 8-input NOR/OR
     U20
 1   CD4068 8-input NAND/AND
     U21
 2   CD4012 Dual 4-input NAND
     U22
 2   CD4063 4-bit Magnitude Comparator
     U23, U24
 2   CD4028 BCD-to-decimal Decoder
     U25, U26
 6   CD4049 Hex Buffer/Inverter
     U33
 2   CD4503 3-state Hex Buffer
     U28, U29
 2   74C373 8-bit Latch
     U31, U32
10   TC5504AP-2 4Kx1 CMOS RAM
     M0, M1, M2, M3, M4, M5, M6, M7, M8, M9
 1   2716 2Kx8 EPROM
     C EPROM *
 1   2732 4Kx8 EPROM
     L EPROM *
```

*Puget Sound Programing sells a MLDL EPROM set. For $25 you get ROM>X, X>ROM, CODE, DECODE, ROM>REG, REG>ROM and some more. Plus the documentation that tells what the routines do. Add $5 for mailing outside of the U.S. and CANADA.

THE ROM ADDRESS SPACE

The "PPC HP-41 Assembly Language Listings Number 2" gives a short description of the HP-41 XROM organization. That information is repeated here. The addresses are given as four hex digits. For example, X000 represents the first address in any 4K block. In each 4K block above 5000, the only blocks allowed for XROMs, there are a number of preassigned address not used for instruction code. The following table lists the preassigned addresses and describes their use.

X000 The XROM number in hex. This is the first of the two numbers you see when viewing a programmed ROM function after the ROM module has been removed. For example, the function XROM 16,07 would come from a ROM with 010 stored at the first address.
X001 The number of functions, in hex, which are displayed in a CATALOG 2.
X002 thru
X0MM The Function Address Table, the FAT. MM is the hex equivalent of 2*N +1, where N is the number of functions. Sixty-four is the maximum number.

```
X0KK 000        KK= MM +1
X0LL 000        LL= MM +2
```

The two null words mark the end of the FAT.

Each entry in the FAT requires two words. There is a high-order word at the even numbered address and a low-order word at the odd numbered address. The high-order word contains the most significant byte of the functions starting address. The low-order word contains the least significant byte of the functions starting address.

If the function is written in Machine Code the FAT address is the address of the first executable instruction, and bit nine of the high-order word is zero. If the ROM function is in T-Code, the FAT address points to the first byte of the desired USER Label instruction, and bit nine of the high-order FAT word is one.

In most ROMs the ROM name is the first function listed in the FAT. This is just a machine code function with one instruction, return. This is a simple trick to list a ROM name first in a CATALOG 2. There is no requirement for a ROM to have a name.

```
X0JJ thru       JJ= MM +3
XFF3 This is where the Machine Code goes.
XFF4 thru
XFFA These are the seven scanned entry points. Unless
     you know exactly what these addresses are for,
     set them to 000. Otherwise the calculator will
     crash or be psychotic.
XFFB thru
XFFE These words are for the ROM revision number. The
     space is available for the MLDL user. It will be
     useful to identify your own revisions!
XFFF This is the ROM check sum. It is used by the
     Diagnostic ROM to verify the contents of HP ROMs.
```

The address space between 3000 and 4FFF is available. It should only be used by the experienced Machine Code programmer.

A FAT EXAMPLE

The following is the Function Address Table from a hypothetical ROM or EPROM which contains the the "Useful M-Code Routines" and a User Code routine, TEST, whose label is at X573. The XROM number is nineteen.

```
X000 013        ;The XROM number, 19.
X001 005        ;The number of functions in the FAT.
X002 000        ;
X003 0EA        ;ROM>X
X004 000        ;
X005 0FC        ;X>ROM
X006 001        ;
X007 002        ;X>A
X008 001        ;
X009 00C        ;TEST
X00A 205        ;
X00B 073        ;"TEST"
X00C 000        ;
X00D 000        ;The required nulls
```

SOME USEFUL M-CODE ROUTINES

These routines were written by Paul Lind (6157). The addresses are for example only. The code is position independent. Just don't cross any 4K boundaries. Be sure to add the address of the first executable instruction to the function table.

ROM>X

Given a 16 bit binary address in the lower 16 bits of X, ROM>X returns the 10 bit ROM word in X and the incremented address in L.

```
X0E5  090  X          ;Routine name
X0E6  03E  >          ;
X0E7  00D  M          ;
X0E8  00F  O          ;
X0E9  012  R          ;
X0EA  0F8  READ 3     ;Read the X register
X0EB  18C  RCR 11     ;Rotate left 3 digits
X0EC  33D  FETCH S&X  ;Get the ROM word
X0ED  10E  A=C ALL    ;Copy to A
X0EE  05A  C=0 M      ;
X0EF  05E  C=0 MS     ;Clear address register
X0F0  0E8  WRITE 3    ;Copy the instruction to X
X0F1  0AE  A<>C ALL   ;
X0F2  046  C=0 S&X    ;Clear the instruction
X0F3  23A  C=C+1 M    ;Increment the address
X0F4  03C  RCR 3      ;Rotate the address to 4 LSDs
X0F5  128  WRITE 4    ;Write the address to L
X0F6  3ED  RTN        ;Return
```

X>ROM

Given a 16 bit address followed by a 12 bit word in the lowest bits of the X register, X>ROM will write the lowest 10 bits (The M-Code) to the ROME address. Address portion is not

altered. L does not contain the incremented address.

```
X0F7  08D  M          ;Routine name
X0F8  08F  O          ;
X0F9  012  R          ;
X0FA  03E  >          ;
X0FB  018  X          ;
X0FC  0FB  READ 3     ;Read the X register
X0FD  040  WRITE 56X  ;040 is the MLDL write
X0FE  3E0  RTN        ;Return
```

X>A

Append character code in X to the left of the alpha register.

```
X0FF  061  A          ;Routine name
X100  03E  >          ;
X101  018  X          ;
X102  0FB  READ 3     ;Read the X register
X103  380  ?NC XQ     ;Convert to binary BCDBIN
X104  008   'BCDBIN'  ;is a native routine*
X105  39C  R=0        ;Set pointer to zero
X106  058  G=C WR, +  ;Low byte of C into G
X107  051  ?NC GO     ;Append character in G to
X108  086   'APPEND'  ;alpha register.  APPEND is
                      ;a native routine*
                      ;Returns through APPEND
```

* A native routine is a routine which is built into the HP-41.

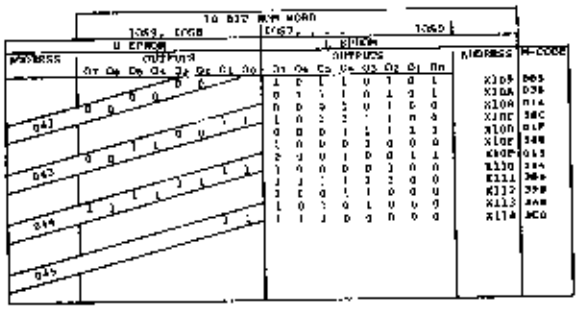
TSS

Toggle flag 55 to disable the printer.

```
X109  0B5  S          ;Routine name
X10A  035  5          ;
X10B  014  T          ;
X10C  38C  ?FSET 0
X10D  01F  GC +3
X10E  388  SETF 0
X10F  013  JNC +2
X110  384  CLRF 0
X111  388  READ 14
X112  398  C=ST X
X113  3A8  WRITE 14
X114  3E0  RTN
```

THE EPROM FORMAT

The types of EPROMs used in the MLDL and other EPROM boxes are all eight bit word formats. The HP-41 uses a ten bit ROM word format. The MLDL uses one EPROM, the L EPROM, for the lower eight bits of the ROM word. The least significant bit of the L EPROM is the least significant bit of the ROM word. The upper two bits of the ROM word are stored in the U EPROM. Each word in the U EPROM stores the upper two bits for four consecutive ROM words. The bits are paired as indicated in the example calculation below. The following EPROM MAP is for the M-Code routine TSS.



The next example shows how to calculate the U EPROM address and position for a given ROM word address.

Write the ROM word address out in binary. X109 becomes

XXXX000100001001,

Shift the address right by two bits,

XXXX000100001001.01

The value to the left of the binary point is the U EPROM address and the value to the right of the point is the U EPROM pair number,

.00 is pair O1, O0
.01 is pair O3, O2
.10 is pair O5, O4
.11 is pair O7, O6

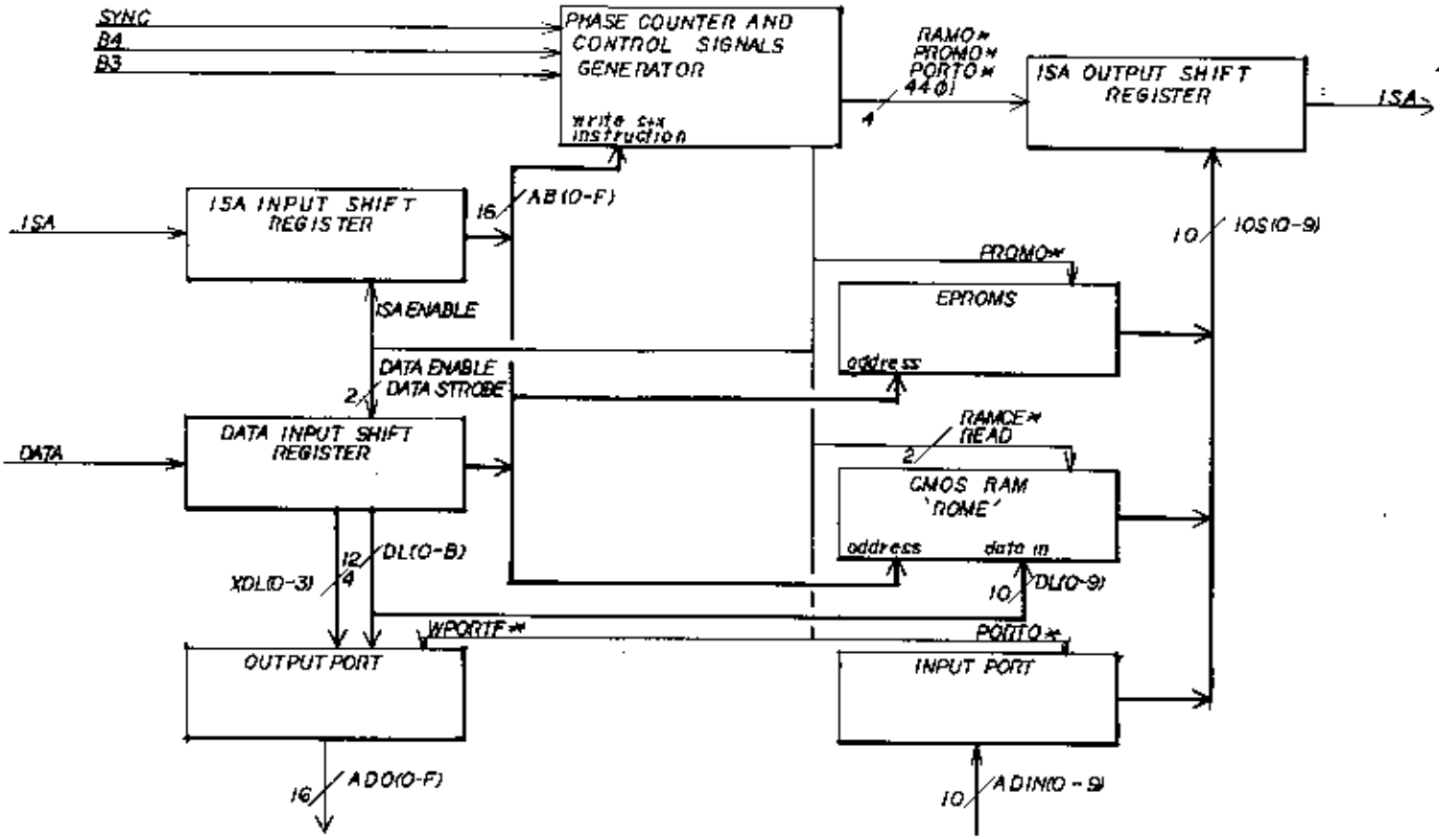
The ROM word at address X109 is stored in the L EPROM address X109 and the U EPROM address 042 outputs O3, and O2.

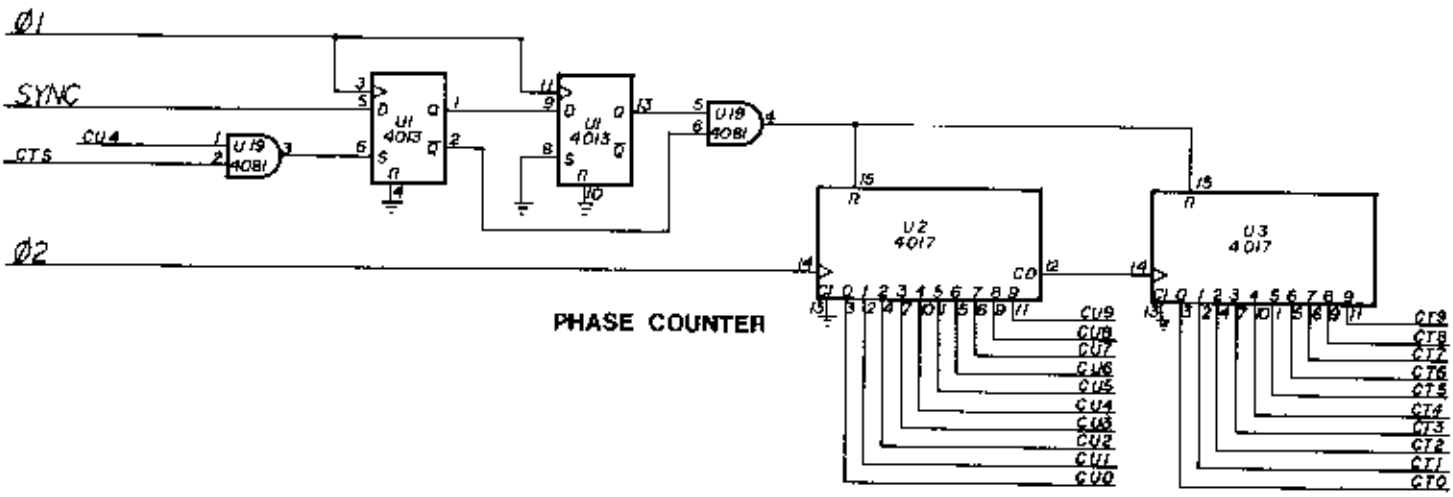
Lynn A. Wilkins (7344)
1912 171 Place NE
Redmond, WA 98052

See pages 32 thru 34 for schematics.

folks at HP know!

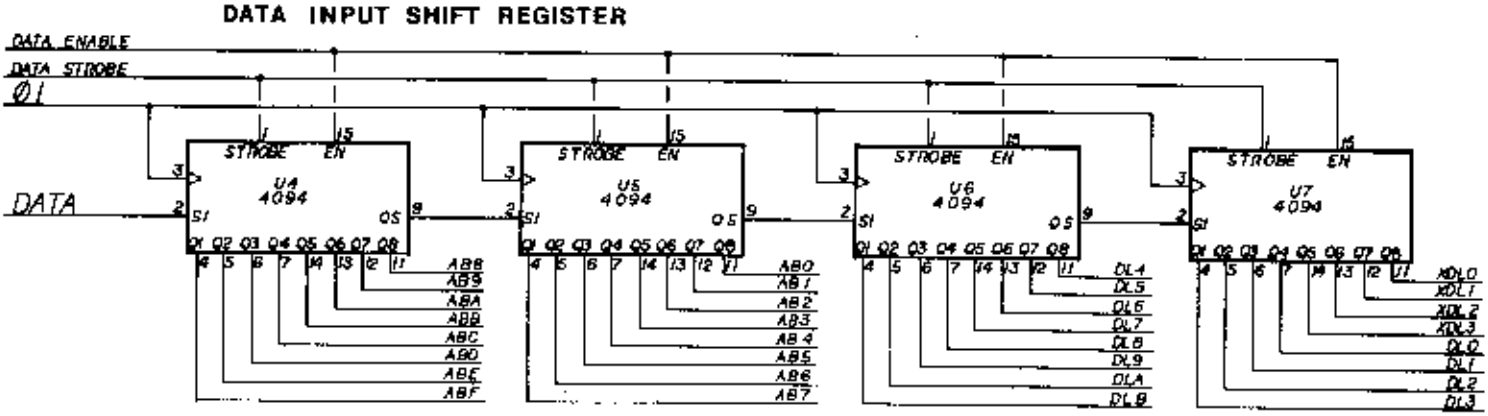
BLOCK DIAGRAM SHOWING DATA AND CONTROL PATHS, Ø1, Ø2, AND PHASE COUNTS CT, CU, ARE NOT SHOWN.

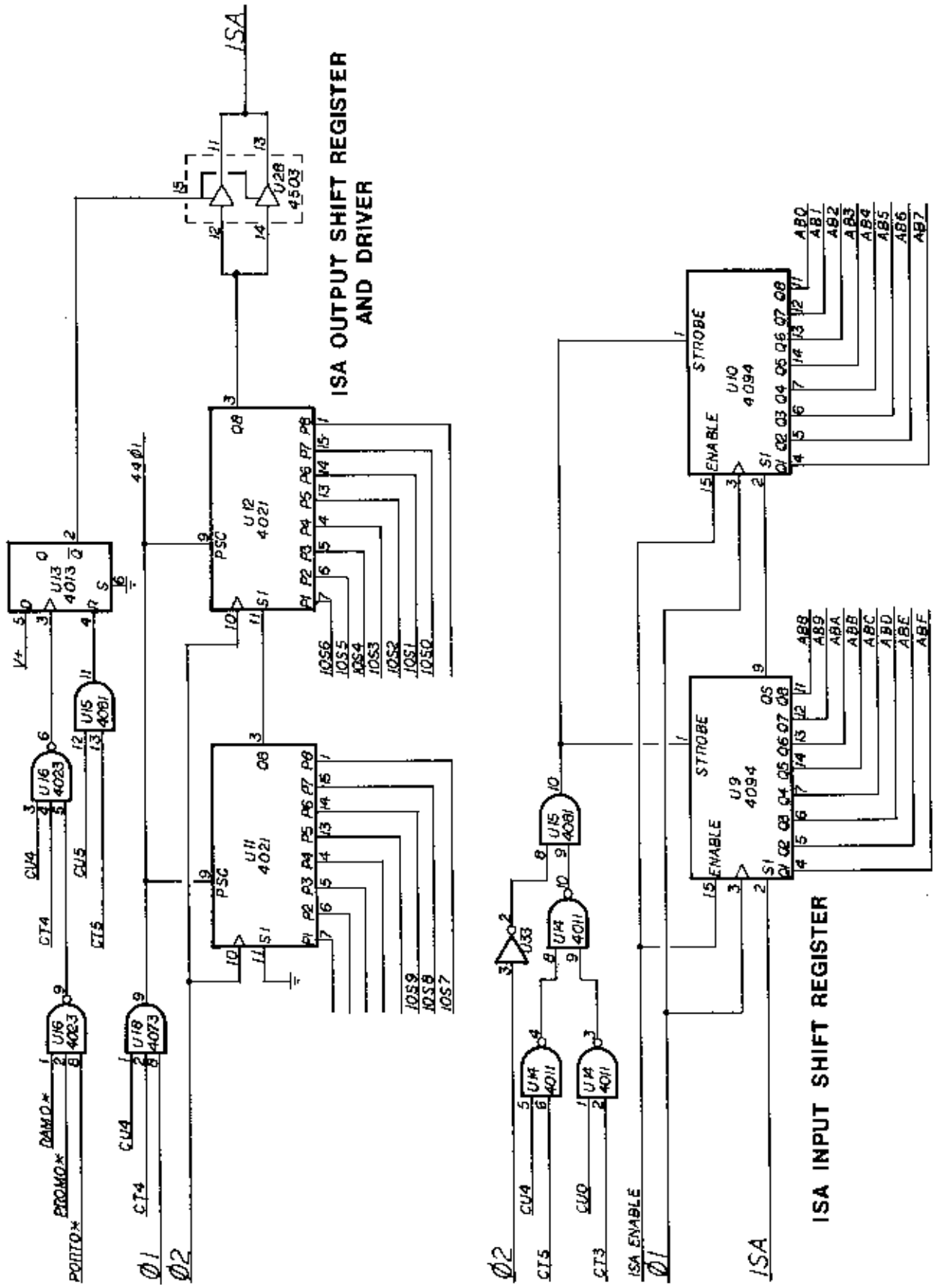


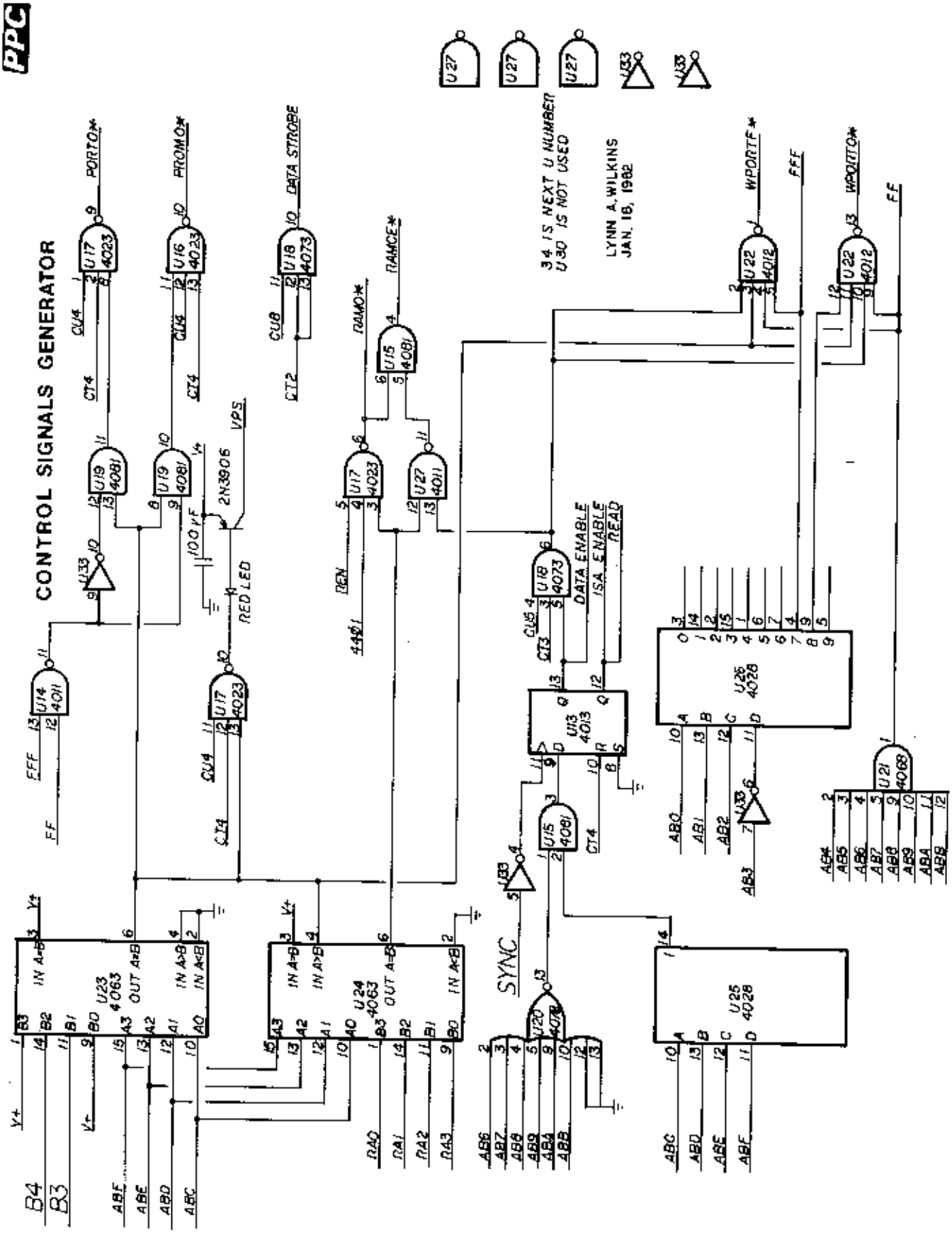


PHASE COUNTER

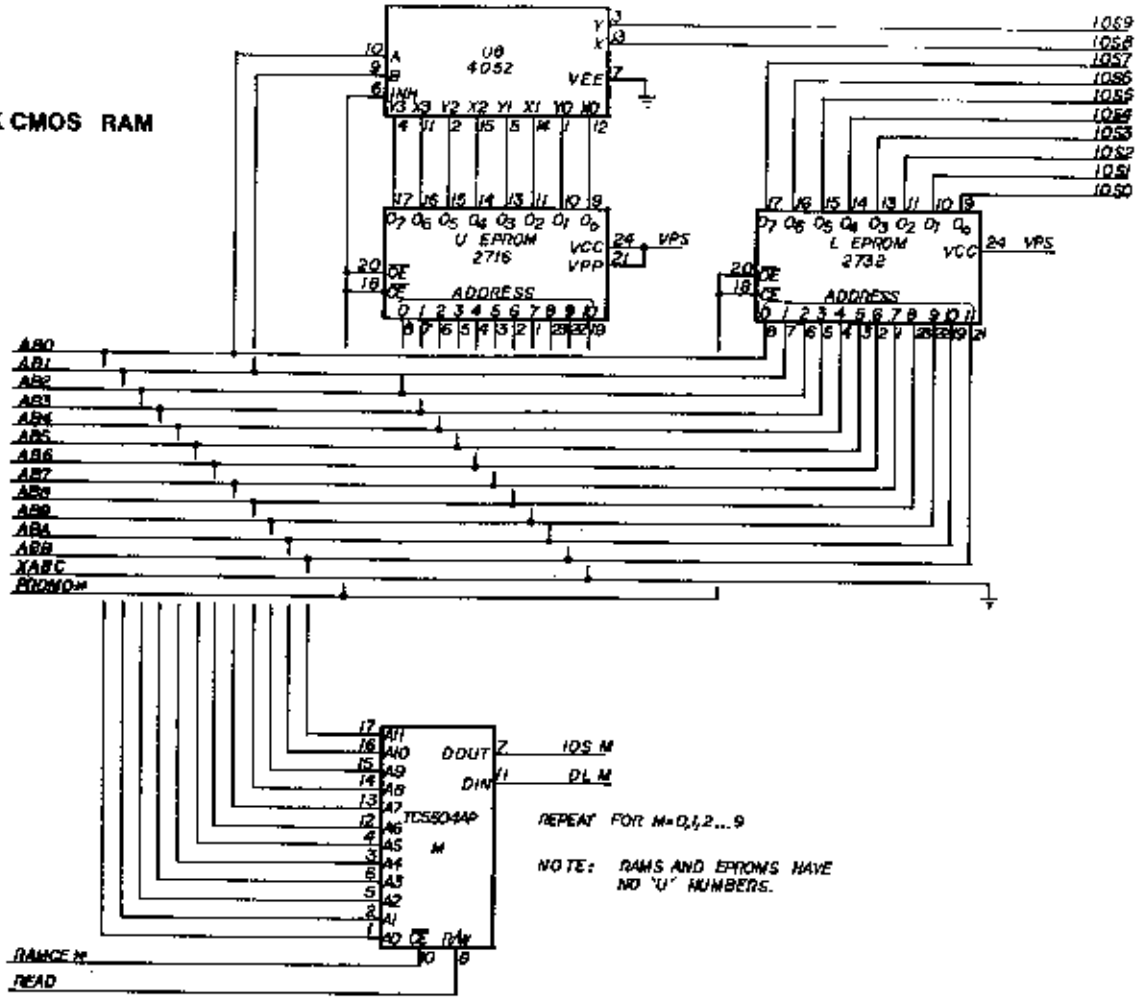
DATA INPUT SHIFT REGISTER




ISA OUTPUT SHIFT REGISTER
AND DRIVER
ISA INPUT SHIFT REGISTER
ISA
ISA ENABLE
STROBE
ENABLE
U10
4094
U9
4094
U11
4021
U12
4021
U13
4013
U14
4011
U15
4081
U16
4023
U18
4073
U6
4023
U28
4503
Ø1
Ø2
CT4
CT5
CT3
CU4
CU5
CU0
DAMO*
PROMO*
PORTO*
V+
IOS0
IOS1
IOS2
IOS3
IOS4
IOS5
IOS6
IOS7
IOS8
IOS9
AB0
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
ABA
ABB
ABC
ABD
ABE
ABF

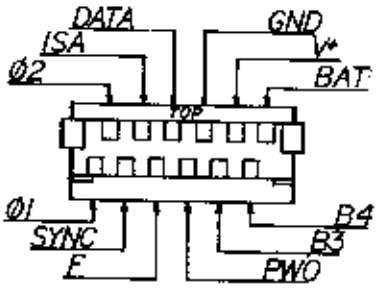

CONTROL SIGNALS GENERATOR
34 IS NEXT U NUMBER
U30 IS NOT USED
LYNN A. WILKINS
JAN. 18, 1982
PORTO*
PROMO*
DATA STROBE
RAMCE*
RAMO*
WPORTF*
WPORTO*
DATA ENABLE
ISA ENABLE
READ
SYNC
RED LED
100 pF
2N3906
VPS
FFF
FF
FE
B4
B3
U23 4063
U24 4063
U25 4028
U26 4028
U3 4013
U15 4081
U16 4073
U17 4023
U18 4073
U19 4081
U20 4078
U21 4068
U22 4012
U27 4011
U33
U35

4K EPROM AND 4K CMOS RAM



REPEAT FOR M=0,1,2...9

NOTE: RAMS AND EPROMS HAVE NO 'U' NUMBERS.

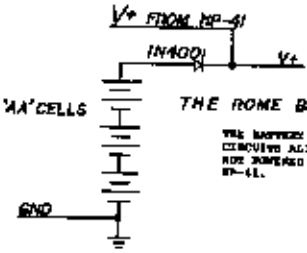


ODDS AND ENDS

CABLE CONNECTOR

USE 240K PULL DOWN RESISTORS ON DATA, ISA, Ø2, ØI, SYNC, B3, AND B4.

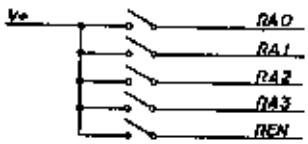
BAT, E, and PWO ARE NOT USED.



THE ROME BACK-UP BATTERY

THE BATTERY KEEPS ALL CIRCUITS ACTIVE WHEN NOT POWERED FROM THE HP-41.

THE ROME ADDRESS SWITCHES

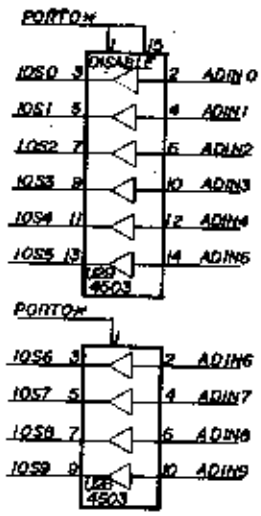


USE 100K PULL DOWN RESISTORS ON THE ROME ADDRESS AND ENABLE LINES.

! IMPORTANT NOTE !
REN ENABLES THE OUTPUT OF ROME DATA AND INSTRUCTIONS TO THE HP-41. WHEN POWER IS REMOVED FROM THE CMOS ROME IT GOES. ON POWER UP THIS ROM INTERFACES IS UNKNOWN. ALL REN MUST BE LOW UNTIL ROME HAS BEEN SET UP WITH RAMS.
THEREFORE:

SWITCH REN TO GND WHENEVER ROME IS TRANSKED.

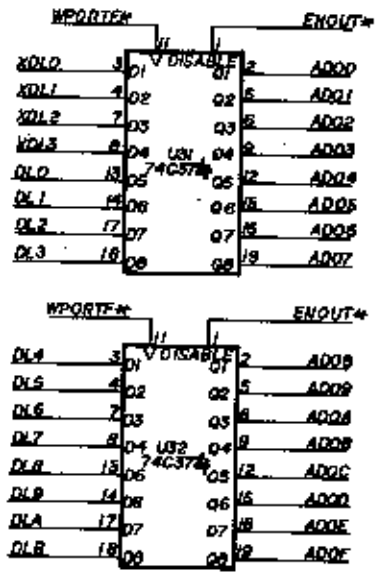
TEN BIT INPUT PORT



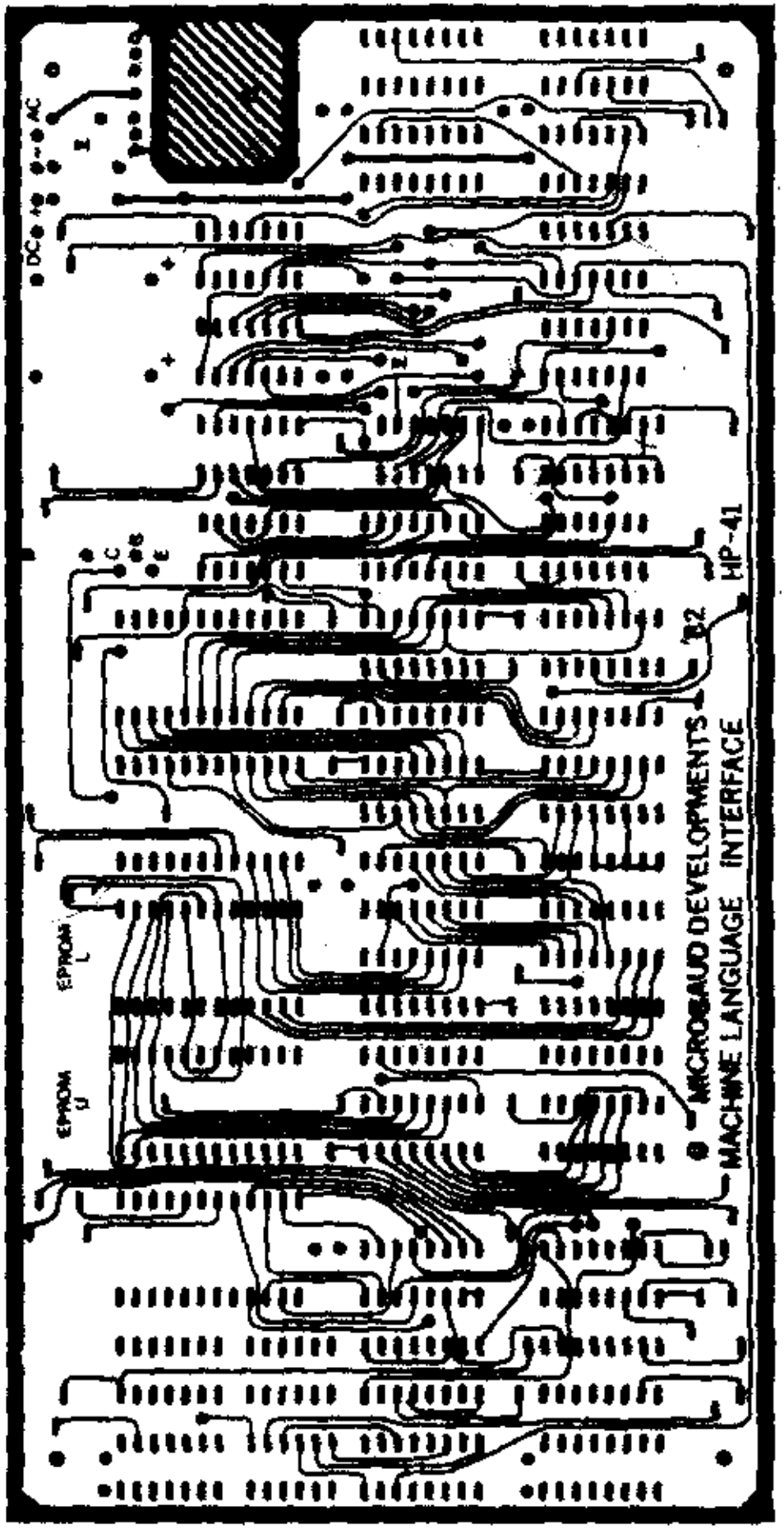
OTHER HALF OF 4503 IS USED FOR THE LED DRIVER.

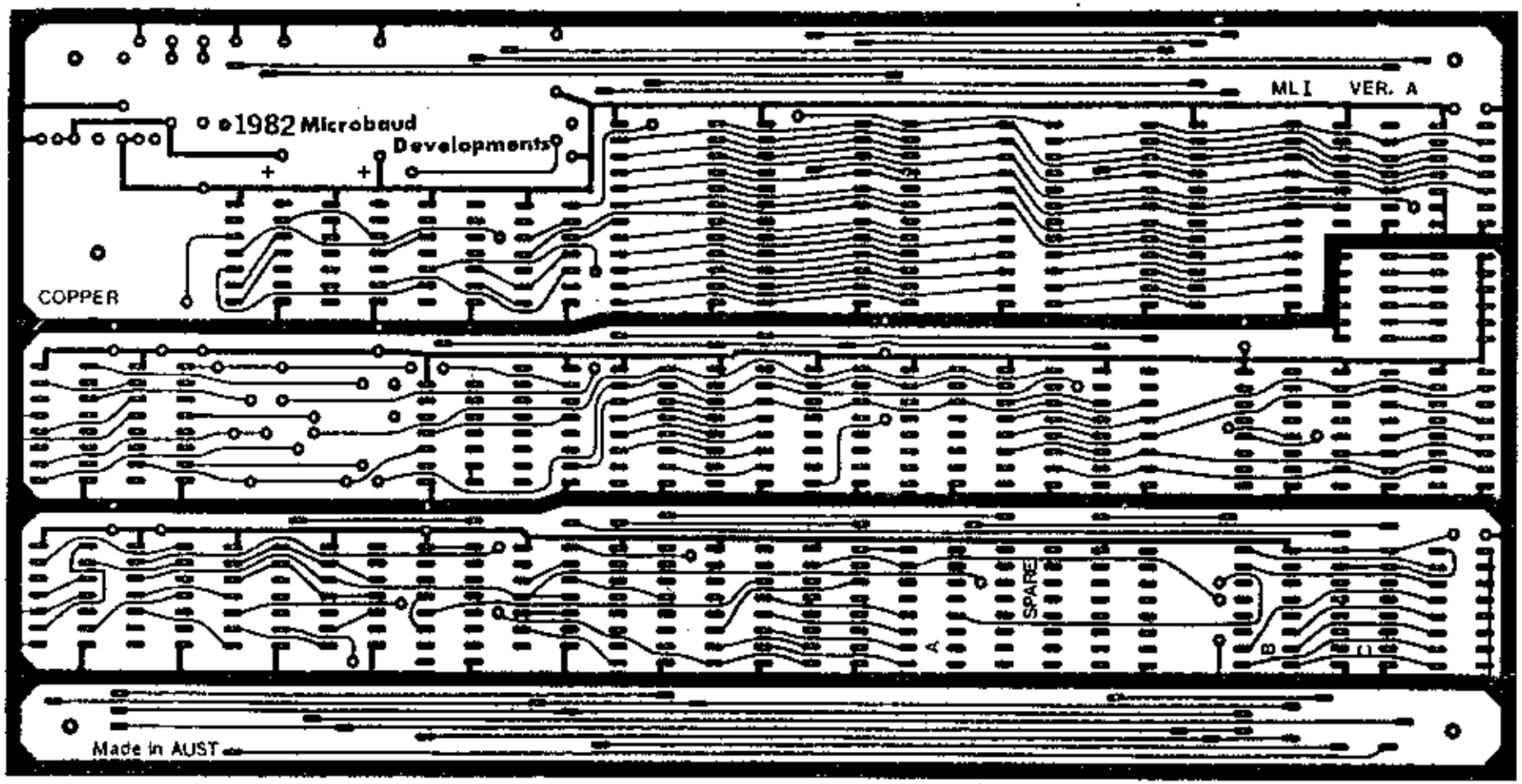
ADIN'S SHOULD HAVE PULL UPS AND SHOULD BE DRIVEN BY OPEN COLLECTOR TYPE DRIVERS.

SIXTEEN BIT OUTPUT PORT



ENOUT* SHOULD HAVE A PULL UP, ADD'S SHOULD SEE CMOS OR 74LS TYPE BUFFERS.


AC
DC
EPROM
U
EPROM
U
MICROSAUD DEVELOPMENTS
MACHINE LANGUAGE INTERFACE
HP-41
J2


©1982 Microbaud
Developments
MLI VER. A
COPPER
SPARE
Made in AUST