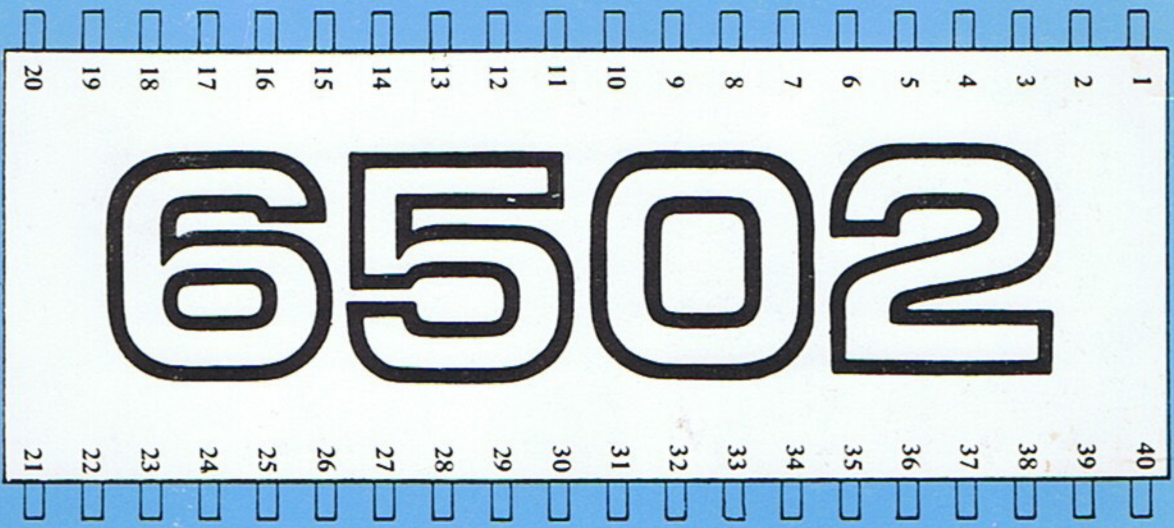
The BEST of MICRO™

AIM 65 SUPER JOLT PET KIM-1 Apple II



SYM-1 DATA HANDLER Challenger II DATAC 1000

Volume 1

The BEST of MICRO™

MICRO is a publication devoted to the world of the 6502 microprocessor: the 6502 based microcomputers, peripheral hardware, software, ideas, applications and so forth.

MICRO began publication with the Oct/Nov 1977 issue and was published regularly on a bimonthly basis for the first year. This volume, "The BEST of MICRO - Volume 1", contains all of the significant material from the first year of MICRO. Only the advertising, a few minor articles, and a few dated articles have been omitted. Any errors which were discovered after the initial publication of the articles have been corrected in this collection.

MICRO obtains most of its material from it's readers: users of 6502 based systems - hobbyists and professionals alike. Authors are paid a modest fee for articles which appear in MICRO, and will obtain additional royalities for reprinting such as this collection.

MICRO is interested in promoting the use of the 6502 and feels that this can best be accomplished by presenting material that is of a useful, informative nature as opposed to lots of games or vague "think" pieces.

MICRO has, in its first year which is covered in this volume, focused primarily on the KIM, PET, and APPLE microcomputers. This is because the material we received was about these three systems. We would welcome material about the OSI systems, or any of the myriad of other 6502 based systems which are not as popular. We also anticipate broad coverage of the new 6502 systems that are just becoming available at the end of the first year: the SYM-1 and the AIM 65.

MICRO covers all of the 6502 based systems because we feel that ideas generated on one system may often be useful to users of other related systems. Therefore, do not just read the stuff in the section on your particular machine, but find out about the other machines as well, and see what you can adapt to your own uses.

MICRO is now published monthly by MICRO Ink, Inc. For information on subscriptions and back issues, write to:

MICRO
P.O. Box 64
So. Chelmsford, MA 01824
USA

Editor/Publisher
Robert M. Tripp

COMBINED TABLE OF CONTENTS

GENERAL Pages 119 to 176

KIM-1

Commodore Business Machines, Inc.
901 California Avenue
Palo Alto, CA 94304
415/326-4000

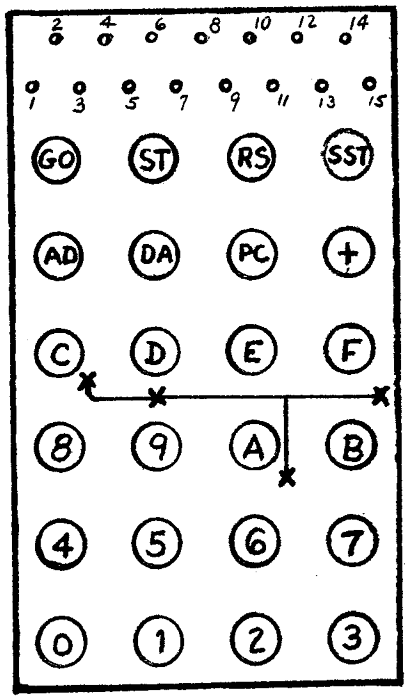
* two perforated "tear-out" reference cards

IMPROVING KIM-1 KEYBOARD RELIABILITY

KIM Application Note
MOS Technology
950 Rittenhouse Road
Norristown, PA 19401

The keyboard on some KIM-1's has a "bouncy" key problem and the "9", "D", or "C" keys may fail entirely. The problem is due to the use of the outer edge of the snap-action discs to jump over the center contact line on the keyboard pc. Since the discs are only held against the pc board with tape, the contact is poor. There are five of these jump-overs in series for the "C" key (four for the "9" key), thereby compounding the problem. To check for the problem, measure the resistance from keyboard pin 3 to pin 15 (numbered from left to right as shown) with the "C" key depressed. It should be less than about 10 ohms.

Fortunately, this problem can be easily corrected. The solution is to solder a thin wire jumper across these poor contacts as follows. Disassemble the keyboard by first removing the four screws on the back of the keyboard at the corners. Then remove the two remaining screws that hold the keyboard to the KIM-1 (note for reassembly that they are longer), being careful not to pull the keyboard pc board away from the KIM-1 board--it's only attached by the solder at one end. With the KIM-1 upside down, separate the black keyboard panel from the keyboard pc board. You may wish to cover the keyboard with masking tape to hold the keys in place. After cutting four small holes through the clear Mylar at the locations indicated by an X in the figure, the lines from the "C" to "9", "D" to "9", "A" to "7" and the line to "B" are exposed. Connecting these points by soldering a thin wire between them routed as shown is sufficient to bridge the five potentially poor contacts.



HYPERTAPE AND ULTRATAPE

Robert M. Tripp
P. O. Box 3
So. Chelsford, MA 01824

While the cassette tape I/O of the KIM-1 is one of its best features, it is terribly slow. Waiting a couple of minutes to load a 1K program can be a real pain. Jim Butterfield showed how to speed up the tape process by writing KIM compatible tapes which were up to six times as fast as the normal KIM ("Supertape", KIM-1 User Notes, Vol. 1, Issue 2, Page 12, or "Hypertape", The First Book of KIM, Page 119). For the COMPUTERIST HELP packages--Editor, Mailing LIst, and Information Retrieval--I doubled this rate by writing a byte of data as a byte, not converting it into two ASCII characters. This "Ultratape" is not KIM compatible and requires a special loading program. The DUMP routine presented here combines both Hypertape and Ultratape. The LOAD routine is used to load an Ultratape. These two routines, as presented here, assume that your system has a means of turning the cassette tape units on and off under program control. (See "Computer Controlled Relays", Page 122).

Dumping Hypertape

Eight locations in page zero are used to hold the arguments for DUMP. For Hypertape they are:

```
00D8 Select Hypertape Mode  02
00D9 Program Identification No.(ID) 01-FE
00DA Starting Address Low (SAL)
00DB Starting Address HIgh (SAH)
00DC Low Memory Address of Data
00DD High Memory Address of Data
00DE Low Count of Bytes to Dump
00DF High Count of Bytes to Dump
```

A feature of this version of Hypertape is that the data to be dumped does not have to reside in its normal memory locations. The Starting Address stored on the tape is provided by 00DA and 00DB independently from the actual memory address which is provided by 00DC and 00DD.

Four additional locations are used on page zero to control the rate at which the data is dumped.

```
00E8  3700 Hz Speed Control      02 = 6X
      (04 = 3X, 06 = 2X, 0C = 1X)
00E9  3700 Hz Pulse Duration        = C3
00EA  2400 Hz Speed Control      03 = 6X
      (06 = 3X, 09 = 2X, 12 = 1X)
00EB  2400 Hz Pulse Duration        = 7E
```

Locations 00EC and 00ED are used for temporary storage. Note that you must change the values of both 00E8 and 00EA to change the dump speed to three, two, or one times the normal KIM dump rate.

DUMP starts at location 0120. The first instruction is a subroutine call to turn on the cassette unit via a relay. If your system is not equiped for automatic control of the cassette, then simply put NOP's in place of this instruction (EA, EA, EA) and the matching subroutine call at location 01A0. The two NOP's at 0123 replace an instruction that was used in the HELP version but which is not required generally. Location 01A6 is the end of the DUMP. This may be either a JMP instruction (as shown) or can be an RTS instruction if DUMP is called as a subroutine.

Hypertape Format

Hypertape uses the standard KIM cassette tape format.

100 SYNCs/Start of Header/ID/SAL/SAH/2 ASCII characters for each byte of data.../Terminator/CHKL/CHKH/EOT/EOT

SYNC is the ASCII SYNC character = 16 hex.
Start of Header is the ASCII * = 2A hex.
ID is the Program Identification Number = 01 to FE hex.
SAL and SAH are the Start Address Low and Start Address High which are used by the KIM Loader.
Each byte of data is converted into two ASCII characters such that a 3F would be stored as ASCII 3 (33) and ASCII F (46).
The Terminator is an ASCII / = 2F hex. CHKL and CHKH are the Check Digit Low and Check Digit High which are generated by the KIM CHKT subroutine during the DUMP and are tested during the LOAD routine.
EOT is the ASCII character = 04 hex.

Loading Hypertape

Since Hypertape is KIM compatible, all you need to load it is the standard KIM Monitor load routine. Set your arguments in 17F5 through 17F9, make sure that the status bits in 00F1 are zero, and start the loader at 1873. That's all there is to it.

Dumping Ultratape

The same eight page zero locations that were used to hold the arguments for the Hypertape DUMP are used for the Ultratape DUMP, but 00DA and 00DB have a different usage.

```
00D8 Select Ultratape Mode                01
00D9 Program Identification Number 01-FE
00DA Low Count of Bytes Dumped
00DB High Count of Bytes Dumped
00DC Low Memory Address of Data
00DD High Memory Address of Data
00DE Low Count of Bytes to Dump
00DF High Count of Bytes to Dump
```

The Ultratape Routine produces a tape that is not compatible with the KIM Monitor. The basic difference is that it stores a byte of data directly without converting it into two ASCII characters. This results in a two-to-one data compression over the KIM method. Since any date value is valid, there must be some way to determine how much data there is in a record. The Terminator character (/=2F) cannot be used since there is no way to distinguish between it and a 2F hex data byte. The problem is solved by putting a count of the number of data bytes into the Header of the tape record. Since the LOAD routine will provide the Starting Address information, the SAL and SAH bytes are not needed. Ultratape uses these two positions in the header to store a two byte count which will be used by LOAD to know how many bytes of data to load. Because the LOAD routine uses a portion of the KIM Monitor to get into sync, to test the Program ID, and to pick up the header information (two byte counter), and does not regain control until after the first byte of data has been picked up and packed by the KIM, the first data byte of Ultratape is actually stored as two ASCII characters, just as in Hypertape. All other data bytes are stored without conversion. A Terminator character follows the last valid data byte so that LOAD can test it and make sure it has not missed or added a character. The remainder of the Header and Trailer are identical to the KIM standard.

100 SYNCs / Start of Header / ID / Count Low/ Count High / 2 ASCII characters for the first data byte / one byte for each data byte.../ terminator / CHKL / CHKH / EOT / EOT

Loading Ultratape

Since Ultratape is not KIM compatilbe, it requires a special LOAD routine. The LOAD routine uses four locations in page zero to hold its arguments:

```
  00D8 Select Load Function               00
  00D9 Program Identification Number 01-FE
  00DA Starting Address Low
  00DB Starting Address High
(00DC is used internally by LOAD)
```

Locations 00E8 to 00EB which were required to set the speed in the dump routines are not required for LOAD. LOAD starts at 0200 with a subroutine call to the routine to turn on the cassette under program control. This should be NOP'ed if you do not have your cassettes under program control. Similarly the call at location 024C should be NOP'ed. Since it is possible to get and detect an error during a LOAD, there must be some way to signal this back to the calling routine. In the HELP programs which this code comes from, a location called STEPNO is incremented on good loads and not incremented on bad loads via the instruction at location 024A. To make LOAD a more general subroutine you can change this to increment location 00D8 which should be set to zero before calling LOAD. Then upon return from LOAD this location can be tested and some action taken if an error has occurred. Location 024F is the end of LOAD. It may be either a JMP instruction (as shown) or can be an RTS instruction if LOAD is called as a subroutine.

In addition to being faster than loading via the KIM monitor, LOAD has the feature that when the load is complete control returns to the user program, not the KIM Monitor. This makes it possible to load data from the cassette under program control without ending up in the KIM Monitor with location 0000 staring you in the face. The data loaded may be ASCII data as in the HELP programs, or may be program data that is overlaying part of the RAM under program control. This feature considerably expands the the usefulness of the KIM cassette interface.

Cassette On/Off Routines

TWRITE at 0252 toggles the direction bit for PB1. This turns a relay on and off on successive calls. DUMP calls TWRITE to control the WRITE cassette. TREAD at 0256 toggles the direction bit for PB0 to control the action of a second relay. TREAD is called by LOAD to control the READ cassette unit.

HYPERTAPE/ULTRATAPE

Dump Routine

```
0120  20 52 02  DUMP    JSR    TWRITE       TURN ON RECORDER
0123  E6 E4             INCZ   STEPNO       BUMP OVER PARAMETERS
0125  A5 DC             LDAZ   PARAM4       LOW MEMORY ADDRESS
0127  8D 5D 01          STA    LOW          SAVE AS POINTER
012A  A5 DD             LDAZ   PARAM5       HIGH MEMORY ADDRESS
012C  8D 5E 01          STA    HIGH         SAVE AS POINTER
012F  20 43 19          JSR    INTCHK       INIT CHECK DIGITS
0132  A9 BF             LDAIM  BF           SET UP TAPE OUTPUT
0134  8D 43 17          STA    PBDD
0137  A9 27             LDAIM  27
0139  85 ED             STAZ   GANG
013B  A9 64             LDAIM  NSYNCS       GET NUMBER OF SYNCS
013D  85 EC             STAZ   COUNT        SET SYNC COUNTER
013F  A9 16     SYNCS   LDAIM  SYNC         OUTPUT SYNCS
0141  20 C0 01          JSR    OUTCHR       AS CHARACTERS
0144  C6 EC             DECZ   COUNT        DEC AND TEST SYNC
0146  D0 F7             BNE    SYNCS        COUNTER UNTIL DONE
****
0148  A9 2A     HEADER  LDAIM  "*           START OF HEADER
014A  20 C0 01          JSR    OUTCHR       CHARACTER
014D  A5 D9             LDAZ   PARAM1       ID NUMBER
014F  20 AC 01          JSR    OUTBT        OUTPUT BYTE, NO CHECK
0152  A5 DA             LDAZ   PARAM2       SAL OR CNTLO
0154  20 A9 01          JSR    OUTBTC       OUTPUT BYTE WITH CHECK
0157  A5 DB             LDAZ   PARAM3       SAH OR CNTHI
0159  20 A9 01          JSR    OUTBTC       OUTPUT BYTE WITH CHECK
****
015C  AD 00 00  DUMPIT  LDA    [LOW,HIGH]   DATA POINTER
015F  EE 5D 01          INC    LOW          BUMP DATA POINTER
0162  D0 03             BNE    CHECK
0164  EE 5E 01          INC    HIGH
0167  20 4C 19  CHECK   JSR    CHKT         KIM CHECK SUM ROUTINE
016A  A6 EC             LDXZ   COUNT        FIRST BYTE TEST
016C  F0 0A             BEQ    FIRST        USE KDUMP FOR FIRST BYTE
016E  A6 D8             LDXZ   PARAM0       TEST DUMP OR KIM DUMP
0170  CA                DEX                 DUMP = 1
0171  D0 07             BNE    KDUMP        KDUMP = 2
0173  20 C0 01          JSR    OUTCHR       1 BYTE/CHARACTER
0176  F0 05             BEQ    TEST         UNCONDITIONAL BRANCH
0178  E6 EC     FIRST   INCZ   COUNT        ONLY FIRST ONCE
017A  20 AC 01  KDUMP   JSR    OUTBT        2 BYTES/CHARACTER
017D  C6 DE     TEST    DECZ   CNTLO        COUNT DATA OUTPUT
017F  D0 DB             BNE    DUMPIT       CONTINUE
0181  C6 DF             DECZ   CNTHI
0183  30 02             BMI    TRAIL        DONE IF MINUS
0185  D0 D5             BNE    DUMPIT       CONTINUE IF NOT ZERO
****
0187  A9 2F     TRAIL   LDAIM  "/           TERMINATOR CHARACTER
0189  20 C0 01          JSR    OUTCHR       OUTPUT AS CHARACTER
018C  AD E7 17          LDA    CHKL         WRITE CHECK CHARACTERS
018F  20 AC 01          JSR    OUTBT        AS BYTE WITHOUT CHECK
0192  AD E8 17          LDA    CHKH
0195  20 AC 01          JSR    OUTBT
0198  A9 04             LDAIM  EOT          END OF TRANSMISSION
019A  20 C0 01          JSR    OUTCHR       WRITE EOT TWICE
019D  20 C0 01          JSR    OUTCHR       TO END DUMP
01A0  20 52 02          JSR    TWRITE       TURN RECORDER OFF
01A3  20 8C 1E          JSR    INIT1        RE-INIT THINGS
01A6  4C 04 03          JMP    NXTSTP       CONTINUE HELP
****
01A9  20 4C 19  OUTBTC  JSR    CHKT         KIM CHECK SUM
01AC  48        OUTBT   PHA                 SAVE DATA ON STACK
01AD  4A                LSRA                SHIFT DATA TO GET
01AE  4A                LSRA                MOST SIGNIFICANT CHAR.
01AF  4A                LSRA
01B0  4A                LSRA
01B1  20 B5 01          JSR    HEXOUT       OUTPUT AS HEX CHARACTER
01B4  68                PLA                 RESTORE DATA
01B5  29 0F     HEXOUT  ANDIM  0F           MASK DATA TO HALF BYTE
01B7  C9 0A             CMPIM  0A           CONVERT TO ASCII
01B9  18                CLC
01BA  30 02             BMI    HEX2
01BC  69 07             ADCIM  07           A - F CHARACTER
01BE  69 30     HEX2    ADCIM  30           0 - 9 CHARACTERS
****
01C0  A2 08     OUTCHR  LDXIM  8            OUTPUT 8 BITS
01C2  86 DC             STXZ   COUNTR
01C4  A2 02     TRY     LDXIM  2            TABLE POINTER
01C6  86 DD             STXZ   TRIB
01C8  B4 E8     ZON     LDYZX  NPUL         NUMBER OF PULSES
01CA  48                PHA                 SAVE DATA
01CB  2C 47 17  ZON1    BIT    CLKRDI       WAIT FOR READY
01CE  10 FB             BPL    ZON1
01D0  B5 E9             LDAZX  TIMG         GET TIMING VALUE
01D2  8D 44 17          STA    CLKIT        SET TIMER
01D5  A5 ED             LDAZ   GANG         GET CURRENT STATE
01D7  49 80             EORIM  80           FLIP TAPE BIT
01D9  8D 42 17          STA    SAD          OUTPUT STATUS
01DC  85 ED             STAZ   GANG         SAVE STATUS FOR LATER
01DE  88                DEY                 DEC COUNTER
01DF  D0 EA             BNE    ZON1
01E1  68                PLA                 RESTORE DATA
01E2  C6 DD             DECZ   TRIB         LOOP TEST
01E4  F0 05             BEQ    SETZ         THIRD SEGMENT?
01E6  30 07             BMI    ROUT         CURRENT BIT DONE?
01E8  4A                LSRA                SHIFT FOR NEXT BIT
01E9  90 DD             BCC    ZON          OUTPUT 1 IF SET
01EB  A2 00     SETZ    LDXIM  0            OUTPUT ZERO
01ED  F0 D9             BEQ    ZON          UNCONDITIONAL BRANCH
01EF  C6 DC             DECZ   COUNTR       BIT COUNTER
01F1  D0 D1             BNE    TRY          MORE TO DO
01F3  60                RTS                 THIS BYTE IS DONE
****
```

HYPERTAPE/ULTRATAPE

Load Routine

```
0200  20 56 02  LOAD    JSR    TREAD        TURN ON RECORDER
0203  A5 D9             LDAZ   PARAM1       GET TAPE ID NUMBER
0205  8D F9 17          STA    ID           STORE FOR KIM LOADER
0208  A9 60             LDAIM  RTS          SET RETURN INSTRUCTION
020A  85 DC             STAZ   PARAM4       HELP POINTER ROUTINE
020C  20 75 18          JSR    INTCHK       INIT CHECK SUM
****
020F  A2 8D     RETURN  LDXIM  STA          ON RETURN FROM KIM
0211  86 D9             STXZ   PARAM1       CREATE STORE ROUTINE
0213  D0 09             BNE    GET1         FIRST CHAR. IS PACKED
0215  20 24 1A  GET     JSR    RDCHT        READ NEXT CHARACTER
0218  AD EA 17          LDA    SAVX+1       GET "RAW" CHARACTER
021B  20 4C 19          JSR    CHKT         USE KIM CHECK SUM
021E  20 D9 00  GET1    JSR    PARAM1       STORE CHAR ROUTINE
0221  E6 DA             INCZ   PARAM2       BUMP LOW POINTER
0223  D0 02             BNE    GET2         TEST LOW POINTER
0225  E6 DB             INCZ   PARAM3       BUMP HIGH POINTER
0227  CE ED 17  GET2    DEC    CNTLO        VEB+1 = CNTLO
022A  D0 E9             BNE    GET          GET NEXT CHARACTER
022C  CE EE 17          DEC    CNTHI        VEB+2 = CNTHI
022F  30 02             BMI    ENDTST       DONE IF MINUS OR ZERO
0231  D0 E2             BNE    GET          MORE IF GREATER THAN 0
****
0233  20 24 1A  ENDTST  JSR    RDCHT        READ TERMINATOR CHAR.
0236  C9 2F             CMPIM  "/           SHOULD BE SLASH
0238  D0 12             BNE    ERROR        ELSE COUNTING ERROR
023A  20 F3 19          JSR    RDBYT        GET LOW CHECK DIGIT
023D  CD E7 17          CMP    CHKL         IS IT CORRECT?
0240  D0 0A             BNE    ERROR        ELSE DATA ERROR
0242  20 F3 19          JSR    RDBYT        GET HIGH CHECK DIGIT
0245  CD E8 17          CMP    CHKH         IS IT CORRECT?
0248  D0 02             BNE    ERROR        ELSE DATA ERROR
024A  E6 E4             INCZ   STEPNO       BUMP STEP ON GOOD LOAD
024C  20 56 02  ERROR   JSR    TREAD        TURN TAPE RECORDER OFF
024F  4C 04 03          JMP    NXTSTP       CONTINUE PROGRAM
****
0252  A9 02     TWRITE  LDAIM  2            TOGGLE BIT 2 FOR WRITE
0254  10 02             BPL    TAPE         UNCONDITIONAL BRANCH
0256  A9 01     TREAD   LDAIM  1            TOGGLE BIT 1 FOR READ
0258  4D 03 17  TAPE    EOR    1703         TOGGLE APPROPRIATE BIT
025B  8D 03 17          STA    1703         PBO = READ/PB1 = WRITE
025E  60                RTS
****
```

```
HYPERTAPE/ULTRATAPE

KIM MONITOR LOCATIONS

INTCHK *   $1943  INITIALIZE CHECK DIGITS
PBDD   *   $1743  PORT B DATA DIRECTION
CHKT   *   $194C  CHECK SUM ROUTINE
CHKL   *   $17E7  CHECK SUM LOW
CHKH   *   $17E8  CHECK SUM HIGH
INIT1  *   $1E8C  KIM INITIALIZE
CLKRDI *   $1747  TIMER
CLKIT  *   $1744  TIMER DONE TEST LOCATION
SAD    *   $1742  SYSTEM PORT B DATA REGISTER
ID     *   $17F9  TAPE ID LOCATION
KLOAD  *   $1875  KIM LOAD INITIALIZATION
RDCHT  *   $1A24  READ TAPE CHARACTER
SAVXA  *   $17EA  CHARACTER SAVE LOC.
KCNTLO *   $17ED  KIM LOW COUNTER
KCNTHI *   $17EE  KIM HIGH COUNTER
RDBYT  *   $19F3  READ BYTE FROM TAPE

HELP MONITOR LOCATION

NXTSTP *   $0304  EXECUTE NEXT STEP ENTRY

DUMP ROUTINE

PAGE ZERO LOCATIONS

00D8               ORG  $00D8

00D8 00   PARAMA =  $00   COMMAND PARAMETER
00D9 00   PARAMB =  $00   TAPE ID NUMBER
00DA 00   PARAMC =  $00   SAL OR CNTLO
00DB 00   PARAMD =  $00   SAH OR CNTHI
00DC 00   PARAME =  $00   MEMORY ADDRESS LOW
00DD 00   PARAMF =  $00   MEMORY ADDRESS HIGH
00DE 00   PARAMG =  $00   CNTLO
00DF 00   PARAMH =  $00   CNTHI

00E4               ORG  $00E4

00E4 00   STEPNO =  $00   NEXT STEP NUMBER

00E8               ORG  $00E8

00E8 02   NPUL   =  $02   NO. PULSES FOR TAPE DUMP
00E9 C3   TIMG   =  $C3   LOW FREQUENCY LENGTH
00EA 03          =  $03   NO. PULSES AT HIGH FREQ.
00EB 7E          =  $7E   HIGH FREQUENCY
00EC 00   COUNT  =  $00   COUNTER
00ED 00   GANG   =  $00   TEMP STORAGE
```

A BLOCK HEX DUMP AND CHARACTER MAP UTILITY PROGRAM FOR THE KIM-1

J. C. Williams
35 Greenbrook Drive
Cranbury, NJ 08512

Here's a useful, fully relocatable utility program which will dump a specified block of memory from a KIM to a terminal. At the user's option, a hex dump with an ASCII character map is produced.

The hex dump will allow the programmer to rapidly check memory contents against a "master" listing when loading or debugging programs. With a printing terminal, the hex dump produces documentation of machine code to complement an assembly listing of a program.

A character map is useful if the block being dumped is an ASCII file. An example would be source code being prepared with an editor for later assembly. The map shows what the file is and where it is in case a minor correction is needed using the KIM monitor.

To use this utility program:

1. Load the code anywhere you want it, in RAM or PROM memory.

2. Define the block to be dumped just as for a KIM-1 tape dump:

BLOCK STARTING ADDRESS	17F5	(low)
	17F6	(high)
BLOCK ENDING ADDRESS+1	17F7	(low)
	17F8	(high)

3. Select the MAP/NOMAP option:

MAP mode	00 in 17F9
NOMAP mode	FF in 17F9

monitor. The examples following the assembly listing will give you the idea.

The program as listed dumps 16 decimal bytes per line. Users with TVT's may want to initialize the line byte counter for 8 decimal bytes per line to allow the hex with MAP format to fit the display. To make this change, replace the $0F at $021E with $07.

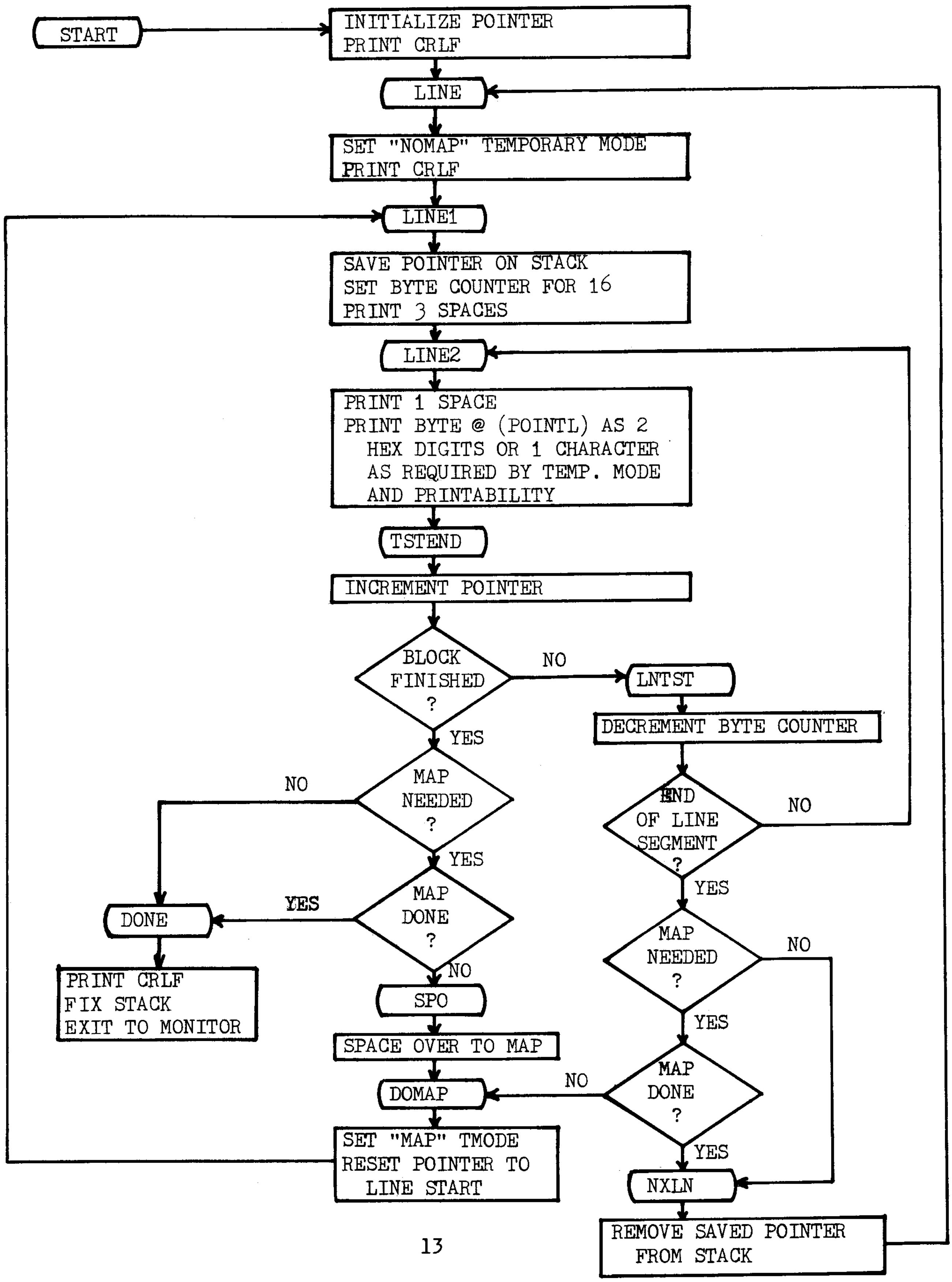
Another possible change is to have the program exit to a location other than the KIM-1 monitor. Exit to a text editor or tape dump may be convenient. Since the MAP/NOMAP option is determined by the most significant (sign) bit of what is stored at $17F9, a suitable tape ID number can be placed there for use of the KIM-1 tape dump or Hypertape. Use ID's from $01-$7F for files needing no character map and ID's from $80-$FE for ASCII files. Start the tape recorder in RECORD when the dump to the terminal is a few seconds from completion.

The flowchart will assist users wanting to make major alterations. Of necessity, ASCII non-printable characters are mapped as two hex digits. If other ASCII codes have special meaning for the user's terminal, a patch will be necessary to trap them. Single-stepping through this program can't be done because it uses the monitor's "display" locations. This is a small price to pay in order to use the monitor's subroutines. If use with a non-KIM 650X system is desired, the subroutines used must preserve the X register.

SYMBOL TABLE

CRLF	1E2F	DOMAP	026E	DONE	028A	EAH	17F8
EAL	17F7	EXT	1C4F	INCPT	1F63	INIT	0200
LINE	020D	LINEA	0217	LINEB	0228	LNTST	0279
MODE	17F9	NXLN	0285	OUTCH	1EA0	OUTSP	1E9E
POINTH	00FB	POINTL	00FA	PRTBYT	1E3B	PRTPNT	1E1E
PTBT	0243	SAH	17F6	SAL	17F5	SPO	0262
TMODE	00F9	TSTEND	0247				

BLOCK HEX DUMP WITH CHARACTER MAP



BLOCK HEX DUMP AND CHARACTER MAP
UTILITY PROGRAM FOR KIM-1

J. C. WILLIAMS - 1978

```
0200                   ORG    $0200

               MEMORY LOCATIONS

0200           TMODE  *      $00F9  TEMPORARY MODE FLAG
0200           POINTL *      $00FA  POINTER
0200           POINTH *      $00FB
0200           SAL    *      $17F5  BLOCK STARTING ADDRESS
0200           SAH    *      $17F6
0200           EAL    *      $17F7  BLOCK ENDING ADDRESS + 1
0200           EAH    *      $17F8
0200           MODE   *      $17F9  00 FOR NO MAP, FF FOR HEX AND MAP
0200           EXT    *      $1C4F  EXIT TO KIM MONITOR

               SUBROUTINES IN KIM MONITOR

0200           OUTCH  *      $1EA0  PRINTS BYTE IN A AS ONE ASCII CHARACTER
0200           CRLF   *      $1E2F  CARRIAGE RETURN AND LINE FEED
0200           OUTSP  *      $1E9E  PRINTS ONE SPACE
0200           PRTBYT *      $1E3B  PRINTS BYTE IN A AS TWO HEX DIGITS
0200           PRTPNT *      $1E1E  PRINTS POINTER
0200           INCPT  *      $1F63  INCREMENTS POINTER

0200 AD F5 17  INIT   LDA    SAL    INITIALIZE POINTER
0203 85 FA            STA    POINTL
0205 AD F6 17         LDA    SAH
0208 85 FB            STA    POINTH
020A 20 2F 1E         JSR    CRLF

020D A9 00     LINE   LDAIM  $00    START A LINE
020F 85 F9            STA    TMODE  INTI TMODE
0211 20 2F 1E         JSR    CRLF
0214 20 1E 1E         JSR    PRTPNT PRINT POINTER
0217 A5 FA     LINEA  LDA    POINTL START A LINE SEGMENT
0219 48               PHA
021A A5 FB            LDA    POINTH
021C 48               PHA
021D A2 0F            LDXIM  $0F    INIT BYTE COUNTER
021F 20 9E 1E         JSR    OUTSP  OUTPUT SOME SPACES
0222 20 9E 1E         JSR    OUTSP
0225 20 9E 1E         JSR    OUTSP
0228 20 9E 1E  LINEB  JSR    OUTSP
022B A0 00            LDYIM  $00    GET THE BYTE
022D B1 FA            LDAIY  POINTL AND SAME ON STACK
022F 48               PHA
0230 24 F9            BIT    TMODE  IN MAP MODE?
0232 10 0F            BPL    PTBT   NO
0234 29 7F            ANDIM  $7F    YES.  TEST FOR PRINTABLE
0236 C9 20            CMPIM  $20    CHARACTER
0238 30 09            BMI    PTBT   PRINT AS TWO HEX DIGITS
023A 68               PLA
```

```
023B 20 A0 1E         JSR    OUTCH  PRINT AS ONE ASCII CHARACTER
023E 20 9E 1E         JSR    OUTSP  AND A SPACE
0241 10 04            BPL    TSTEND UNCONDITIONAL BRANCH
0243 68        PTBT   PLA           RECOVER BYTE AND
0244 20 3B 1E         JSR    PRTBYT PRINT AS TWO HEX DIGITS

0247 20 63 1F  TSTEND JSR    INCPT  INCREMENT POINTER
024A A5 FA            LDA    POINTL AND TEST AGAINST ENDING
024C CD F7 17         CMP    EAL    ADDRESS + 1
024F A5 FB            LDA    POINTH
0251 ED F8 17         SBC    EAH
0254 90 23            BCC    LNTST  NOT BLOCK END.  TEST FOR LINE END
0256 2C F9 17         BIT    MODE   END OF BLOCK REACHED.  IS MAP
0259 10 2F            BPL    DONE   NEEDED.  DONE IF NOT.
025B 24 F9            BIT    TMODE  HAS MAP BEEN DONE?
025D 30 2B            BMI    DONE   YES, EXIT
025F CA               DEX
0260 30 0C            BMI    DOMAP  NO SPACES NEEDED
0262 20 9E 1E  SPO    JSR    OUTSP  SPACE OVER TO CHARACTER MAP
0265 20 9E 1E         JSR    OUTSP
0268 20 9E 1E         JSR    OUTSP
026B CA               DEX
026C 10 F4            BPL    SPO

026E C6 F9     DOMAP  DEC    TMODE  DO THE MAP.  FIRST SET THE
0270 68               PLA           MAP FLAG AND RESET POINTER TO
0271 85 FB            STA    POINTH START OF LINE
0273 68               PLA
0274 85 FA            STA    POINTL
0276 38               SEC
0277 B0 9E            BCS    LINEA  AND PRINT THE MAP SEGMENT

0279 CA        LNTST  DEX           TEST FOR END OF LINE
027A 10 AC            BPL    LINEB  NOT AT END.  DO THE NEXT BYTE
027C 2C F9 17         BIT    MODE   END OF LINE SEGMENT REACHED.  IS MAP NEEDED?
027F 10 04            BPL    NXLN   NO, DO THE NEXT LINE
0281 24 F9            BIT    TMODE  HAS THE MAP SEGMENT BEEN DONE?
0283 10 E9            BPL    DOMAP  NO, DO IT NOW

0285 68        NXLN   PLA           DO THE NEXT LINE
0286 68               PLA           FIRST FIXT THE STACK
0287 38               SEC
0288 B0 83            BCS    LINE   DO THE NEXT LINE

028A 20 2F 1E  DONE   JSR    CRLF   DONE
028D 68               PLA           REMOVE SAVED POINTER FORM STACK
028E 68               PLA
028F 4C 4F 1C         JMP    EXT    EXIT TO KIM MONITOR
```

```
KIM
2880 52 17F5
17F5 00 00.   BLOCK STARTING ADDRESS = 2800
17F6 28 28.
17F7 80 80.   BLOCK ENDING ADDRESS + 1 = 2880
17F8 28 28.
17F9 00 FF.   SELECT MAP OPTION
17FA FF 021E
021E 0F 07.   SELECT 8 LOCATIONS PER LINE
021F 20 0200
0200 AD G     START PROGRAM AT 0200

2800     0D 00 10 20 20 20 42 4C    0D 00 10           B  L
2808     4F 43 4B 20 48 45 58 20    O  C  K     H  E  X
2810     44 55 4D 50 20 41 4E 44    D  U  M  P     A  N  D
2818     20 43 48 41 52 41 43 54       C  H  A  R  A  C  T
2820     45 52 20 4D 41 50 0D 00    E  R     M  A  P  0D 00
2828     20 20 20 20 55 54 49 4C                U  T  I  L
2830     49 54 59 20 50 52 4F 47    I  T  Y     P  R  O  G
2838     52 41 4D 20 46 4F 52 20    R  A  M     F  O  R
2840     4B 49 4D 2D 31 0D 00 30    K  I  M  -  1  0D 00 0
2848     0D 00 40 20 20 20 4A 2E    0D 00 @           J  .
2850     20 43 2E 20 57 49 4C 4C       C  .     W  I  L  L
2858     49 41 4D 53 20 2D 20 31    I  A  M  S     -     1
2860     39 37 38 0D 00 50 0D 00    9  7  8  0D 00 P  0D 00
2868     60 20 4F 52 47 20 24 30    `     O  R  G     $  0
2870     32 30 30 0D 00 70 0D 00    2  0  0  0D 00 p  0D 00
2878     80 20 20 20 4D 45 4D 4F    80          M  E  M  O
```

```
KIM
17F5
17F5 00 00.   BLOCK STARTING ADDRESS = 2800
17F6 28 28.
17F7 80 80.   BLOCK ENDING ADDRESS + 1 = 2880
17F8 28 28.
17F9 FF 00.   SELECT NOMAP OPTION
17FA FF 021E
021E 07 0F.   SELECT 16 LOCATIONS PER LINE
021F 20 0200
0200 AD G     START PROGRAM AT 0200

2800     0D 00 10 20 20 20 42 4C 4F 43 4B 20 48 45 58 20
2810     44 55 4D 50 20 41 4E 44 20 43 48 41 52 41 43 54
2820     45 52 20 4D 41 50 0D 00 20 20 20 20 55 54 49 4C
2830     49 54 59 20 50 52 4F 47 52 41 4D 20 46 4F 52 20
2840     4B 49 4D 2D 31 0D 00 30 0D 00 40 20 20 20 4A 2E
2850     20 43 2E 20 57 49 4C 4C 49 41 4D 53 20 2D 20 31
2860     39 37 38 0D 00 50 0D 00 60 20 4F 52 47 20 24 30
2870     32 30 30 0D 00 70 0D 00 80 20 20 20 4D 45 4D 4F
```

IMPORTANT ADDRESSES OF KIM-1 AND MONITOR

William R. Dial
438 Roslyn Avenue
Akron, OH 44320

Address	Label	Function
00EF	PCL	Program Counter - Lo Byte
00F0	PCH	Program Counter - Hi Byte
00F1	P (PREG)	Status Register of Processor Set "00" for Binary
00F2	SP (SPUSER)	Stack Pointer
00F3	A (ACC)	Accumulator
00F4	Y	Y-Register
00F5	X	X-Register
00F6	CHKHI	Checksum on Tape, Hi
00F7	CHKSUM	Checksum on Tape, Lo
00F8	INL	Input Buffer, Lo - Display Buffer
00F9	INH	Input Buffer, Hi - Display Buffer
00FA	POINTL	Pointer, Lo - Display
00FB	POINTH	Pointer, Hi - Display
00FC	TEMP	Temporary Storage Byte
00FD	TMPX	Temporary Storage Byte
00FE	CHAR	Current Character for TTY
00FF	MODE	Byte Indicating KYBD or TTY Mode on KIM

Detail of Processor Status Register P (00F1)

Bit No.		Flags (1 = set)
0	C	Carry
1	Z	Zero Result
2	I	Interrupt Disable
3	D	Decimal Mode
4	B	Break Command
5		Reserved for Expansion
6	V	Overflow
7	N	Negative Result

01FF
01FE .... } STACK Needed to Process Interrupts, save Addresses, etc.
01F8 etc.

I/O Ports, Interval Timers, and 6530 RAM Usage

Address		Label	Function
1700		PAD	Port A Data (user 1/0)
1701		PADD	Port A Data Direction (1 = Output)
1702		PBD	Port B Data (User 1/0)
1703		PBDD	Port B Data Direction (0 = Input)
1704 /	1744	CLKIT	INTERVAL TIMER
1705	1745	CLK8T	1704 et seq User
1706	1746	CLK64T	1744 et seq KIM MONITOR
1707	1747	CLK1024T	
1707	1747	CLKRDI	Read Time Out Bit
1706	1746	CLKRDT	Read Time
170C	174C	1T	TIMER USED when IRQ Interrupt at PB7 needed
170D	174D	8T	
170E	174E	64T	
170F	174F	1024T	
1740		SAD	Port A Data (KIM MONITOR)
1741		PADD (SADD)	Port A Data Direction
1742		SBD	Port B Data (KIM MONITOR)
1743		PBDD (SBDD)	Port B Data Direction
1780			Available Memory Block (Program PLEASE, etc.)
17E7		CHKL	Checksum for Tape Monitor
17E8		CHKH	
17E9		SAVX	Storage Location
17EA			" "
17EB			" "
17EC		VEB	Volatile Execution Block
17F2		CNTL 30	TTY Delay
17F3		CNTH 30	TTY Delay
17F4		TIMH	
17F5		SAL	Starting Address - Lo (Audio and Paper Tape)
17F6		SAH	- Hi
17F7		EAL	Ending Address - Lo
17F8		EAH	- Hi
17F9		ID	ID Number (Program No. on Tape)

17FA/FFFA	NMIV	(NMIL)	Stop Vector (Stop = ICOO)	Load 00
FB/FFFB		(NMIH)		1C
FC/FFFC	RSTV	(RSTL)	RST Vector	00
FD/FFFD		(RSTH)		1C
FE/FFFE	1RQV	(IRQL)	IRQ Vector (BRK = ICOO)	00
FF/FFFF		(IRQH)		1C

SUB-ROUTINES - 6530-003

Address	Label	Function
1800	DUMPT	Dump Memory to Tape
1873	LOADT	Load Memory from Tape
1932	INTVEB	Initiate Volatile Execution Block
194C	CHKT	Compute CHKSUM for Tape Load
195E	OUTBTC	Output One Byte
196F	HEXOUT	Convert LSD of A to ASCII and Output to Tape
197A	OUTCHT	Output to Tape One ASCII CHAR (Use Subs ONE & ZRO)
199E	ONE	Output to Tape = 1 (9 pulses 138 μ sec each)
19C4	ZRO	Output 0 to Tape (6 pulses 207 μ sec each)
19EA	INCVEB	Sub to INC VEB + 1, 2
19F3	RDBYT	Sub to read Byte from Tape
1A00	PACKT	Pack A = ASCII into SAVX as Hex Data
1A24	RDCHT	Get 1 Character from Tape and Return with Character in A (Use SAVX + 1 to ASM Char)
1A41	RDBIT	Gets one bit from Tape and returns it in sign of A
1A6B	PLLCAL	Diagnostics: PLL calibrate Output, 166 μ sec pulse string

SUB-ROUTINES - 6530-002

Address	Label	Function
1C00	SAVE	KIM Entry vis STOP (NMI) or BRK (IRQ) Also SST
1C22	RST	KIM Entry via RST (Reset)
1C2A	DETCPS	Count Start Bit
1C4F	START	Make TTY/KB Selection
1CDC	PCCMD	Display Program Counter by Moving PC to POINT
1C64	CLEAR	Clear Input Buffer INL, INH
1C6A	READ	Get Character
1C77	TTYKB	Main Routine for Keyboard and Display

Address	Label	Function
1CE7	LOAD	Load Paper Tape from TTY
1D42	DUMP	Dump to TTY from Open Cell Address to LIMHL, LIMHH Limit High, H and L
1E1E	PRTPNT	Sub to Print POINTL, POINTH
1E2F	CRLF	Print String of ASCII Characters from TOP + X to TOP
1E3B	PRTBYT	Print 1 Hex Byte as Two ASCII Characters
1E5A	GETCH	Get 1 Character from TTY, Return from Sub with Char in A. X is preserved and Y returned = FF.
1E88	INITS	Initialization for SIGMA
1E9E	OUTSP	Print One Character CHAR = A. X is preserved, Y returned = FF.
1EA0	OUTCH	OUTSP Prints One Space.
1ED4	DELAY	This loop simulates DETCPS Section and will delay 1 Bit Time.
1EEB	DEHALF	Delay half Bit Time - Double right shift of Delay Constant for a Div by 2.
1EFE	AK	Sub to Determine if Key is depressed or Condition of SSW (Key not dep or TTY Mode A = 0) (Key dep or KB Mode A = not zero)
1F19	SCAND	Output to 7 Segment Display
1F1F	SCANDS (DISPLA)	Lights 7 Segment Display
1F48	CONVD	Convert and Display Hex - Used by SCAND only
1F63	INCPT	Sub to Increment POINT
1F6A	GETKEY	Get Key from Keyboard, Return with A = Key value. If A GT. than 15 then illegal or no Key.
1F91	CHK	Sub to Compute Check Sum
1F9D	GETBYT	Get 2 Hex Characters and Pack into INL, INH. X preserved, Y returned = 0.
1FAC	PACK	Shift Character in A into INL, INH
1FD5	TOP	Table
1FE7	TABLE	Table Hex to 7 Segment

A DEBUGGING AID FOR THE KIM-1

Albert Gaspar
305 Wall Street
Hebron, CT 06248

DEBUG is a program designed to assist the user in debugging and manipulating programs. It resides in memory locations 1780 - 17E6 and provides a means for inserting breakpoints in a user program, moving blocks of bytes throughout memory, filling memory with repetitious data, and calculating branch values. It uses selected KIM monitor subroutines.

Operating Modes

DEBUG has three operating modes:

1. Keyboard Mode: DEBUG remains in a wait loop anticipating keyboard entry which will be recognized as either data or command characters. This mode is initiated either by using the KIM monitor to start at location 178E, or by the execution of a previously inserted breakpoint in a user program.

2. Execute Mode: DEBUG executes logic to service a user command. This mode is completed in microseconds and will not be noticeable by the user.

3. Non-Control Mode: DEBUG relinquishes control when the user keys in "RS", or "ST" during Keyboard Mode, or uses the CONTINUE Command.

To start, the user must first load "B5" into 17FE and "17" into 17FF using the KIM. Then the user begins DEBUG by starting at location 178E. This puts DEBUG into Keyboard Mode. The user then keys in combinations of the 16 data characters available on the keyboard. Input data is displayed in a manner similar to that of the KIM - from right to left - except that only the left-most five display positions are utilized (exceptions are noted below).

The user must continue to key in characters until he is satisified that the required data is input. Then one of the several Command code characters available (B, C, D, E, or F) is keyed in. At this point, or at any time previous to this, if the input is not correct and the user wishes to change the display, he merely continues to enter data until the display string is correct. When the display concatenation is satisfactory (either 2 or 4 data characters and 1 Command character) he keys in "AD". Now DEBUG will go into Execute Mode (without echoing the entry of "AD") and immediately examines the last previous character input. If this character is not a legitimate Command character (B, C, D, E, or F), DEBUG becomes confused and will transfer to unpredictable memory locations. Thus the user is held wholly responsible for the validity of his input. He should always check that either his keyed-in data is correct before hitting "AD", or that his Command was indeed executed. Note: if a key other than "AD", the 16 data characters, "RS", or "ST" is depressed, its high order 4 bits are stripped and the remaining low order 4 bits are displayed and evaluated as whatever the combination happens to represent.

Assuming that the character input immediately prior to "AD" is a legitimate Command character, DEBUG - still in Execute Mode - will process the data which was input prior to the Command code (either 2 or 4 characters). Note that the Command values (B, C, D, E, of F) if found in the data field are processed as standard hex values.

BREAK This command allows the user to insert a breakpoint anywhere desired in his program. When this point is subsequently reached during execution of his program, control will be passed to Keyboard Mode of DEBUG and further execution of the user program will effectively be temporarily discontinued. Also at this time the user area will be restored to the original configuration existing at the time of the breakpoint insertion.

Input Sequence:

Press Keys	See on Display
4 Data Characters B "AD"	4 char B1

The 4 Data Characters define the Breakpoint location desired. The BREAK Command saves the user byte at the Breakpoint and deposits a BRK instruction in place of it. Thus, that user area should not be altered by the user while DEBUG is in Non-Control Mode and a Breakpoint is eminent, or the Breakpoint return will not work. More than one Breakpoint can be eminent at one time; however since DEBUG will store only one byte at a time, multiple simultaneous Breakpoints should be applied only at user locations containing the same instruction. This way it is immaterial which BRK triggers a return to DEBUG - the user area will be properly replaced.

This Command includes 1 of 2 instances where the sixth display position is used. If the sixth position contains a 1, the Command has been correctly processed. If the position contains any other value, it indicates that depression of the "AD" key has caused multiple bounces and the byte stored by DEBUG within itself is now "00" - not the original user byte. Thus DEBUG will still function correctly but will not correctly restore the user position when a Breakpoint return is initiated. The user must restore the location manually (using KIM) after the return has been performed - otherwise "00" will be left in the location.

CONTINUE This Command causes DEBUG to pass execution to a user specified location. It is similar to the passing of control through KIM and either method may be used to execute user code.

Input Sequence:

Press Keys	See on Display
4 Data Characters C "AD"	4 char C0

The 4 Data Characters define the address to which control is to be passed. The above display is only momentary since control is immediately passed to a user area (Non-Control Mode) The purpose of the Continue Command will usually be to execute to a previously inserted Breakpoint. When this occurs, as previously stated, control returns to Keyboard Mode, of DEBUG. **At this point, the leftmost 4 display digits will contain the address at which the Breakpoint was located.** See Overall Notes #1 for a continuation warning.

MOVE This Command will move a block of up to 256 bytes to another memory area. It is non-destructive (unless, of course, a shift is performed).

Input Sequence:

Press Keys			See on Display	
4 Data Characters	F	"AD"	4 char	F0
	(F for From)			
4 Data Characters	D	"AD"	4 char	D0
	(D for Destination)			
2 Data Characters	E	"AD"	XX 2 char	E0
	(E for Execute)			

The 4 Data Characters above represent the locations one less than the locations, respectively, from which and to which the data is to moved. The 2 Data Characters above represent the hex value of the number of bytes to be moved. If the user desires to move 256 (dec.) bytes, he must input "00" in the "E" Command. "F" and "D" execution may be input in either order - "F" then "D" or "D" then "F".

MOVE will correctly move blocks of bytes from one area of memory to another. However it will correctly shift bytes only in an upward direction. Attempting downward shifts will result in the repeating of as many of the last bytes in the original block as there is a difference in the block positions. For example - shifting a block of say (n) bytes starting at 0200 to a new area starting at 0202 will correctly shift the (n) bytes upward 2 locations. Attempting to shift a block of (n) bytes starting in 0202 to a new area starting in 0200 will result in the last 2 bytes of the original block to be repeated downward from their original locations continuing to 0200. This may not be completely undesireable since - 1) normally the user will be interested in expanding an area, not in compressing it (for example, to add instructions); and, 2) this serves as a useful tool to provide filler bytes in memory when desired.

BRANCH This Command assists in calculating Branch values.

Input Sequence:

1. Enter the necessary 12 bytes of Branch Overlay, either through KIM or by tape overlay. (These will, of course, have to be restored to the original configuration when through with BRANCH).

1. Put DEBUG into Keyboard Mode.

Press Keys			See on Display
2 char/2 Char.	E	"AD"	2 char/2 char/D-VALUE

The first 2 characters are the 2 least significant values of the Branch Address. The next 2 characters are the 2 least significant values of the Branch to Address. The "E" stands for Evaluate. The correct Displacement VALUE will appear in the 5th and 6th display positions. The displacement is calculated assuming that the two addresses are in the same page. For page overlap, entry will have to be done twice. We believe that different users will have different preferential methods for doing this, so our own method, which is somewhat involved, is not described. If both entries are on the same page but are separated by a distance greater than the standard branch range, the value calculated will be incorrect. It is the user's responsibility to check for out-of-range values.

Overall Notes

1. When a Breakpoint has been executed, DEBUG does not store and then restore accumulator, register, and status values. Thus, the user must take care in continuing from a Breakpoint if any of these parameters have a subsequent bearing in further user program execution. (Though this and other omissions are glaring defects, no apology is made - there was just insufficient memory available for inclusion of any refinements.)

2. When returning from a "BRK" instruction, DEBUG pulls the status register information from the stack and ignores it. If this DEBUG version is used in conjunction with an interrupt system, locations 17FE - 17FF must contain the address of the user interrupt handler. The beginning of the handler must be similar to that shown on page 144 of the KIM Programming Manual. The logic listed in example 9.7 must be utilized as shown. "BNE BRKP" will point to the DEBUG location defined below. If the user handler determines that the interrupt was caused by "BRK", then the handler must jump to location 17B5. DEBUG will then obtain the "BRK" address and perform subsequent logic to return the user byte to its original configuration and continue on into Keyboard Mode.

3. This version of DEBUG uses page zero locations 0000, 0001, 0002, 0003, and 0004, but only as scratch areas during Keyboard and Execute Modes. The user can use these areas as temporary scratch areas when DEBUG is not being executed.

4. Due to limited instruction space, DEBUG is particularily susceptible to key bounce. The user should remain watchful of such occurrences, especially during BREAK execution as previously described.

5. My goal here was to fit as much DEBUG power into locations 1780 - 17E6 as possible - not to write a great breakpoint/move/branch calculate routine. (That has already been done by others) Thus DEBUG had to be written in relatively concise and tight code, using data as instructions, instructions as data, overlapping instructions, using the same code to do different things, instruction modification, position instructions in prescribed relative locations, use of "write-only-memory", etc. I do not approve of this type of programming - in fact I strongly recommend against it. However, in this case I hope the goal I had justifies the mess that DEBUG has turned out to be. In any event I would like to point out that as tight as the code is, it is still possible to add other functions here and there. For example the version I usually use displays the value of the accumulator in display locations 5 and 6 when returning back from a Breakpoint. At times I also use another version which doesn't require the "BRK" instruction at all. This is convenient when debugging interrupt programs since no additional interrupt is needed for DEBUG. However, both versions penalize me in other areas, which makes it all a trade-off decision.

[Editor's Note: Gaspar seems to be suggesting a collection of specialized DEBUG programs, each customized to provide a particular set of capabilities while residing in minimal memory. Using his code as a starting point, a "programwise" reader should be able to construct his own set of DEBUG aids.]

```
        ZERO   *      $0000  LOCATION 0000
        ONE    *      $0001
        TWO    *      $0002
        THREE  *      $0003
        FOUR   *      $0004

        INH    *      $00F9  KIM DISPLAY POINTERS
        POINTL *      $00FA
        POINTH *      $00FB

        RETURN *      $17B5  INTERNAL ADDRESS
        TBLOFF *      $17D4  TABLE OFFSET
        JUMPER *      $17DD  INTERNAL ADDRESS

        INITI  *      $1E8C  KIM INITIALIZE ROUTINE
        SCANDS *      $1F1F  KIM SCAN DISPLAY ROUTINE
        GETKEY *      $1F6A  KIM GET KEYBOARD CHARACTER
1780 B1 02     EXEC   LDAIY TWO    GET CHAR TO BE MOVED
1782 91 00            STAIY ZERO   MOVE IT
1784 88               DEY
1785 D0 F9            BNE   EXEC   CONTINUE UNTIL DONE
1787 98        DANDF  TYA          GET TO OR FROM ADDRESS
1788 95 F3            STAZX $00F3  STORE IT IS SCRATCH
178A A5 FB            LDAZ  POINTH
178C 95 F4            STAZX $00F4
178E 20 8C 1E  START  JSR   INITI  SET FLAGS AND INIT.
1791 20 1F 1F         JSR   SCANDS DISPLAY BUFFER
1794 D0 F8            BNE   START
1796 20 1F 1F  KEY    JSR   SCANDS NEW CHARACTER INPUT?
1799 F0 FB            BEQ   KEY    NO, CONTINUE TO DISPLAY
179B 20 6A 1F         JSR   GETKEY YES, GET THE CHARACTER
179E A6 04            LDXZ  FOUR   PICK UP LAST CHAR. INPUT
17A0 C9 10            CMPIM $10    IS THE NEW CHAR. "AD"?
17A2 F0 30            BEQ   PROCES YES.  PROCESS CURRENT COMMAND
17A4 85 04            STAZ  FOUR   NO. STORE IT
17A6 A2 04            LDXIM $04    AND SHIFT IT INTO THE DISPLAY
17A8 0A        SHIFT  ASLA
17A9 26 F9            ROL   INH    SHIFT THE DISPLAY LEFT
17AB 26 FA            ROL   POINTL
17AD 26 FB            ROL   POINTH
17AF CA               DEX
17B0 D0 F6            BNE   SHIFT  DONE SHIFTING
17B2 85 F9            STA   INH    YES.  ADD NEW CHAR TO DISPLAY
17B4 F0 D8            BEQ   START  UNCONDITION RETURN
17B6 38               SEC
17B7 68               PLA          IGNORE STATUS
17B8 68               PLA          GET "FROM" ADDRESS
17B9 E9 02            SBCIM $02    SUBTRACT 2
17BB 85 FA            STAZ  POINTL DISPLAY LOW ORDER
17BD 68               PLA
17BE E9 00            SBCIM $00    SUBTRACT CARRY, IF ANY
17C0 85 FB            STAZ  POINTH DISPLAY HI ORDER
17C2 A2 0C            LDXIM $0C    CHEAT ON RX
17C4 E6 F9     B      INC   INH    COUNT KEY BOUNCES
17C6 A0 00            LDYIM $00
17C8 B1 FA            LDAIY POINTL GET USER BYTE
17CA 9D DC 17         STAX  $17DC  STORE IT
17CD BD DB 17         LDAX  $17DB  GET "BRK"
17D0 91 FA            STAIY POINTL STORE IN USER AREA
17D2 A2 0D            LDXIM $0D    CHEAT ON RX
17D4 A4 FA     PROCES LDYZ  POINTL
17D6 BD D4 17         LDAX  TBLOFF PREPARE TO GO TO COMMAND LOGIC
17D9 8D DD 17         STA   $17DD  ALTER INSTRUCTION
17DC D0 FF            BNE   JUMPER JMP TO COMMAND LOGIC
17DE EA               NOP          FUTURE EXPANSION
17DF E6        TABLE  =     $E6    BRANCH TO "B"
17E0 06               =     $06    BRANCH TO "C"
17E1 A9               =     $A9    BRANCH TO "D"
17E2 A2               =     $A2    BRANCH TO "E"
17E3 A9               =     $A9    BRANCH TO "F"
17E4 6C FA 00  C      JMI   POINTL 00 OR ADDRESS USED AS "BRK"
```

BRANCH CALCULATION OVERLAY

```
                        ORG    $1780

                 INH    *      $00F9
                 POINTL *      $00FA
                 POINTH *      $00FB

1780 38          EXEC   SEC           INITIALIZE SUBTRACT
1781 A5 FA              LDAZ   POINTL
1783 69 FD              ADCIM  $FD    CORRECTION CONSTANT
1785 E5 FB              SBCZ   POINTH
1787 85 F9              STAZ   INH    STORE RESULT IN DISPLAY
1789 4C 8E 17           JMP    $178E  JUMP TO START
```

Examples

1. Load DEBUG. Load "B5" into 17FE and "17" into 17FF.

2. Start execution at location 178E.

3. Depressing any of the 16 keyboard characters will cause the 5 leftmost display digits to shift left and the new character to be inserted into the fifth position.

4. Assume that there is a program in 0200-0250. Now, to execute from 0200-0240:

0 2 4 0 B AD	Display is	0240 B1
0 2 0 0 C AD		0200 C0
		0240 XX

When the user program executes to location 0240, it will return to DEBUG which then will replace the original byte at 0240 and will return to Keyboard Mode.

5. User wishes to add a 3 byte instruction in 0241-0243. Thus he must shift his program from 0241-0250 to 0244-0253.

0 2 4 0 B AD	Display is	0240 B1
0 2 4 0 F AD		0240 F0

(Remember that MOVE requires addresses 1 less than the actual values.)

X X 1 0 E AD	Display is	XX10 E0

(10 = 0250 - 0241 + 1)

This shifts bytes in 0241-0250 to 0244-0253. User can now insert his 3 new instructions into locations 0241, 0242, and 0243.

6. User wishes to load NOP into locations 0300-03FF. Load "EA" into 03FF using KIM. Return to DEBUG.

0 3 0 0 F AD	Display is	0300 F0
0 2 F F D AD		02FF D0
0 0 E AD		XX00 E0

(Move 256 decimal bytes.)

7. User wishes to calculate the value required for a HERE BCC START where HERE = 0204 and START = 0250.

First, load overlay (12 bytes) and return to DEBUG.

0 4 5 0 E AD	Display is	0450 4A

Thus the branch value is 4A and the branch instruction will be BCC 4A.

Remember that if further DEBUG usage is planned, the original 12 bytes starting at 1780 have to be replaced.

Program Notes

1. The instruction listings at 17B4 and 17E4 are NOT errors and must be placed in memory exactly as shown.

2. Locations 17E7 and 17E8 are used by the KIM monitor for tape checksum. However, their usage in DEBUG will not interfere with KIM since the two programs do not, of course, use them at the same time.

EMPLOYING THE KIM-1 MICROCOMPUTER AS A TIMER AND DATA LOGGING MODULE

Marvin L. De Jong
Dept. of Mathematics-Physics
The School of the Ozarks
Point Lookout, MO 65726

The interval timers on the 6530 on the KIM-1 microcomputer provide a convenient way to measure the time between two or more events. Such events might include the start and end of a race, the exit of a bullet from a gun and its arrival at a measured distance along its trajectory, the interruption of light to a series of phototransistors placed along the path of a falling object, an animal arriving at this feeding station, the arrival of telephone calls, etc. Some of these measurements will be described in more detail below. Each event must produce a negative pulse which the microcomputer detects and records the time at which the event occurred. The time is stored in memory and later displayed on the 6-digit KIM display.

Description of the Programs

The data logging, timer, and display programs are listed in Tables 1, 2, and 3, respectively. The programs must be used together for the applications described in this paper, but each might be used with other applications, for example pulse generators, frequency counters, temperature logging, light flashing, etc. The events to be timed must produce either a one-shot pulse (positive-zero-positive) whose duration is at least 50 microseconds or a zero to positive transition which must be reset to zero before the next event. These signals are applied to pin PA0 on the KIM applications connector. The programs could easily be modified to detect positive pulses.

The first pulse starts the timer which continues to operate on an interrupt basis. The first pulse is not recorded by the data logging program since it corresponds to t = 0. Successive pulses cause the data logging program to store the six digit time counter in memory. The number of events (not counting the first event) N, to be timed must be stored in location 0003. Remember to convert the number of events, N, to base 16 before entering it in memory. As the program is written, N must be less than 75. = 4B hex.

The function of the timer program is to load the interval timer, increment the six digit time counter, and return to the data logging program. At the end of each timing period the timer causes an interrupt to occur (pin PB7 on the application connector must be connected to pin 4 on the expansion connector), the computer jumps to the timer program, does its thing, and returns to the main data logging program to wait for events.

Table 4 lists several timing intervals which are possible and the numbers which must be loaded into the various timers to produce the given interval. For example, if one wishes to measure time in units of 100 microseconds, then 49 hex must be stored in the divide-by-one counter whose address is 170C. In this case, the numbers which appear on the display during the display portion of the program represent the number of 100 microsecond intervals between the first event and the event whose time is being displayed. To put it another way, multiply the number on the display by 0.0001 to get the time in seconds. The other possibilities listed in the table are treated in the same way.

When all N events have been logged, the program automatically jumps to the display program. When one is ready to record the data, key #1 on the keyboard is depressed. The time of each event, excepting the first which occurred at t = 0 is displayed on the six digit readout for several seconds before the display moves to the time of the next event. This gives the experimenter time to record the data on paper. If more time is required, increase the value of the number stored in location 0289.

Table 4 also lists the measured time interval and gives the percent error between the stated interval (say 100 microseconds) and the actual measured interval (99.98 microseconds). The measurements were made by connecting a frequency counter (PASCO SCIENTIFIC Model 8015) to pin PB7 while the program was running and after the first event had started the timer. If greater accuracy is required for the 10 millisecond and 100 millisecond intervals, then experiment with putting NOP instructions between the PHA instruction and the LDA TIME instruction in the timer program.

Experiments and Applications

The simplest application for the program is a simple stopwatch with memory. Any suitably debounced switch can be used. See pages 213 and 280 in CMOS COOKBOOK by Don Lancaster, published by Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, Indiana 46268 for some suitable switching circuits.

Being a physics teacher, I originally designed the program to collect data for an "acceleration of gravity" experiment in the introductory physics lab. The technique may be applicable to other problems so it is described herein. Nine phototransistors (Fairchild FPT 100 available from Radio Shack) were mounted on a meter stick at 10 cm intervals. An incandescent (do not try fluorescent lighting) 150 watt flood lamp provided the illumination. The interface circuit is shown in Figure 1.

The 555 timer serves as a Schmitt trigger and buffer which produces a negative pulse when an object passes between the light and the phototransistor. The 500 kilo ohm potentiometer is adjusted so that an interruption of the light to any of the phototransistors increases the voltage at pin 2 of the 555 from about 1.5 volts to at least 3.5 volts; a very simple adjustment which should be made with a VTVM or other high impedance meter.

In the case of a simple pendulum, the relationship between the period and the amplitude can be investigated by allowing the pendulum to "run down" while logging the times when the bob interrupts the light to a single phototransistor. With only one phototransistor the timer-data logging program can also be used as a tachometer if a rotating system of some kind is involved.

Lancaster, in the CMOS COOKBOOK, describes a tracking photocell pickoff which could be used in conjunction with the program for outdoor races and other sporting events. See page 346 in the "COOKBOOK". A simple light beam-phototransistor system could be placed in a cage and the apparatus would record the times at which an animal interrupted the beam, giving a measurement of animal activity.

If you want to measure the muzzle velocity of your rifle or handgun, you will have to be more devious. First, I would modify the program so that one pin, say PA0, is used to start the timing while another pin, say PB0, is used to stop the timing. This can be accomplished by changing instructions 0226 and 022D in Table 1 from AD 00 17 to AD 02 17. Then I would use a fine wire foil to hold the clock input of a 7474 flip-flop low until the wire foil was broken by the exit of the bullet from the gun. The Q output going high would start the timing, so it would be connected to PA0. To end the timing one could use a microphone to detect a bullet hitting the backstop. Of course, the microphone signal would have to be amplified and used to trigger say the other flip-flop of the 7474 to signal the second event. So as not to take all your fun away, that is the last hint except that the distance between start and stop should be at least 10 feet. Please be careful.

I would like to acknowledge the education and inspiration I received at an NSF Chautauqua Type Short Course and a KIM workshop, both conducted by Dr. Robert Tinker.

[Editor's Note: For a related KIM-1 application, see "A Simple Frequency Counter Using the KIM-1", by Charles R. Husbands, on page 26 of this issue.]

Table 1

Data logging program

```
                DLOG   ORG    $0200

                LOW    *      $0000
                MID    *      $0001
                HIGH   *      $0002
                N      *      $0003
                LO     *      $0003
                MI     *      $0053
                HI     *      $00A3
                INH    *      $00F9
                POINTL *      $00FA
                POINTH *      $00FB
                KEY    *      $0271
                PAD    *      $1700
                GETKEY *      $1F6A
                SCANDS *      $1F1F

0200 78         INIT   SEI          DISABLE INTERRUPT
0201 F8                SED          SET DECIMAL MODE
0202 A2 00             LDXIM $00    SET X = 0
0204 A9 50             LDAIM $50    SET INTERRUPT VECTOR = 0250
0206 8D FE 17          STA   $17FE
0209 A9 02             LDAIM $02
020B 8D FF 17          STA   $17FF
020E A9 99             LDAIM $99    INIT COUNTER BY STORING 255 (FF)
0210 85 00             STAZ  LOW    INT THE THREE, TWO DIGIT
0212 85 01             STAZ  MID    MEMORY LOCATIONS OF THE
0214 85 02             STAZ  HIGH   COUNTER
0216 AD 00 17   START  LDA   PAD    READ INPUT PIN PA0
0219 29 01             ANDIM $01    LOGICAL AND WITH PAC
021B D0 F9             BNE   START  LOOP IF PIN IS 1
021D AD 00 17   FLIP   LDA   PAD    IF PIN IS NOT 1, READ AGAIN
0220 29 01             ANDIM $01    LOGICAL AND WITH PA0
0222 F0 F9             BEQ   FLIP   LOOP IF PIN IS 0
0224 58                CLI          ELSE, ENABLE INTERRUPT AND JUMP TO
0225 00                BRK          TIMER PROGRAM THEN RETURN
0226 EA                NOP          PADDING FOR BRK COMMAND
0227 AD 00 17   CHEK1  LDA   PAD    THESE INSTRUCTIONS ARE THE SAM
022A 29 01             ANDIM $01    AS THE START AND FLIP SEQUENCE
022C D0 F9             BNE   CHEK1
022E AD 00 17   CHEK2  LDA   PAD
0231 29 01             ANDIM $01
0233 F0 F9             BEQ   CHEK2
0235 E8                INX          INCREMENT X FOR EACH DATA POINT
0236 A5 00             LDAZ  LOW    COUNTER CONTENTS ARE STORED IN A
0238 95 03             STAZX LO     SEQUENCE OF LOCATIONS INDEXED
023A A5 01             LDAZ  MID    BY X
023C 95 53             STAZX MI
023E A5 02             LDAZ  HIGH
0240 95 A3             STAZX HI
0242 E4 03             CPXZ  N      COMPARE X TO N.  RETURN TO CHEK1
0244 D0 E1             BNE   CHEK1  IF X IS LESS THAN N
0246 78         DISPLA SEI          ELSE GO TO DISPLAY AFTER
0247 4C 71 02          JMP   KEY    DISABLING INTERRUPTS
```

Table 2

Timer program

```
                     TIMER  ORG   $0250

                     TIME   *     $0049
                     TIMEX  *     $170C
                     LOW    *     $0000
                     MID    *     $0001
                     HIGH   *     $0002

0250 48              INTRPT PHA          PUSH ACCUMULATOR ON STACK
0251 A9 49                  LDAIM TIME   START TIMER FOR 49(16) CYCLES
0253 8D 0C 17               STA   TIMEX
0256 A9 01                  LDAIM $01    INCREMENT COUNTER BY ADDINT 1
0258 65 00                  ADCZ  LOW    TO THE TWO LOW DIGITS
025A 85 00                  STAZ  LOW    AND STOR RESULT
025C A9 00                  LDAIM $00    ADD CARRY FROM PREVIOUS
025E 65 01                  ADCZ  MID    ADDITION TO MID DIGITS.  IF
0260 85 01                  STAZ  MID    CARRY OCCURS FROM THE TWO MID
0262 A9 00                  LDAIM $00    FROM THE TWO MID DIGITS, THEN
0264 65 02                  ADCZ  HIGH   ADD THIS TO THE TO HIGH DIGITS
0266 85 02                  STAZ  HIGH
0268 68                     PLA          PULL ACCUMULATOR FROM STACK
0269 40                     RTI          RETURN TO DATA LOGGER
```

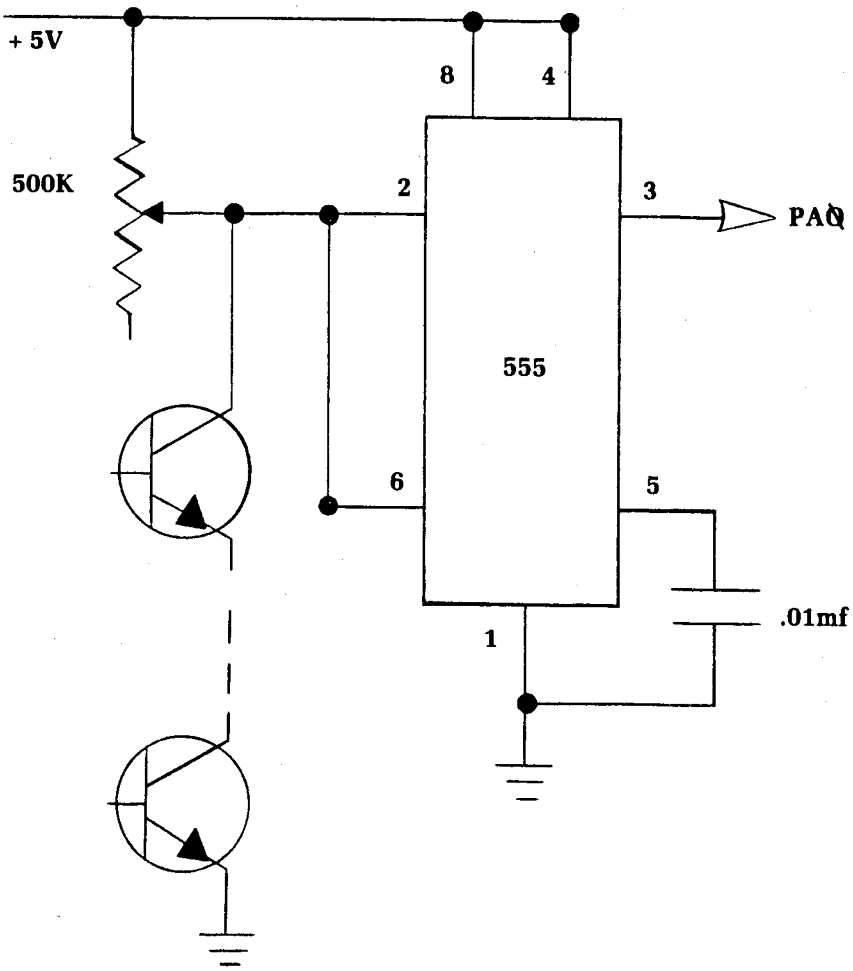


Figure 1

Interface circuit using up to 10 phototransistors. The dashed line represents other phototransistors. The time at which the light to any of the phototransistors is interrupted is recorded by the timer-data logging program.

Table 3

Display program

```
                DISPLA ORG   $0271

                N      *     $0003
                LO     *     $0003
                MI     *     $0053
                HI     *     $00A3
                INH    *     $00F9
                POINTL *     $00FA
                POINTH *     $00FB
                INIT   *     $0200
                TIME   *     $1707
                GETKEY *     $1F6A
                SCANDS *     $1F1F

0271 20 6A 1F   KEY    JSR   GETKEY JUMP TO KIM KEYBOARD MONITOR
0274 C9 01             CMPIM $01    TEST VALID INPUT
0276 D0 F9             BNE   KEY    IF NOT, WAIT FOR INPUT
0278 A2 01             LDXIM $01    INIT X REGISTER TO INDEX
027A B5 03      NXPNT  LDAZX LO     DATA POINTS
027C 85 F9             STAZ  INH    PUN IN KIM DISPLAY REGISTERS
027E B5 53             LDAZX MI
0280 85 FA             STAZ  POINTL
0282 B5 A3             LDAZX HI
0284 85 FB             STAZ  POINTH
0286 8A                TXA          SAVE X WHILE IN SUBROUTINE BY
0287 48                PHA          PUSHING IT ON THE STACK
0288 A0 10             LDYIM $10    TIME TO DISPLAY EACH POINT
028A 98         AGN    TYA          SAVE Y WHILE IN SUBROUTINE BY
028B 48                PHA          PUSHING IT ON THE STACK
028C A9 FF             LDAIM $FF
028E 8D 07 17          STA   TIME
0291 20 1F 1F   REPEAT JSR   SCANDS SCANDS IS KIM ROUTINE WHICH
0294 AD 07 17          LDA   TIME   DISPLAYS DATA IN 00F9, 00FA
0297 30 03             BMI   OVER   AND 00FB.  REPEATED JUMPS TO
0299 4C 91 02          JMP   REPEAT SCANDS PRODUCES A CONSTANT DISPLAY
029C 68         OVER   PLA          RESTORE Y REGISTER
029D A8                TAY
029E 88                DEY          DECREMENT Y BY 1 AND REPEAT
029F F0 03             BEQ   HOP    DISPLAY UNTIL Y = 0
02A1 4C 8A 02          JMP   AGN
02A4 68         HOP    PLA          RESTORE X REGISTER
02A5 AA                TAX
02A6 E4 03             CPXIM N      COMPARE X WITH N.  IF X IS LESS
02A8 F0 04             BEQ   BEGIN  THAN N INCREMENT X AND DISPLAY
02AA E8                INX          NEXT POINT.  ELSE, RETURN TO
02AB 4C 7A 02          JMP   NXPNT  THE BEGINNING
02AE 4C 00 02   BEGIN  JMP   INIT
```

Table 4

Timing intervals for the program.

Time Interval	Value	Address	Measured Interval	% Error
100 microsec	49	170C	99.98 microsec	0.02%
1 millisec	7A	170D	0.9998 millisec	0.02%
10 millisec	9C	170E	10.007 millisec	0.07%
100 millisec	62	170F	100.5 millisec	0.5%

A SIMPLE FREQUENCY COUNTER USING THE KIM-1

Charles R. Husbands
24 Blackhorse Drive
Acton, MA 01720

A piece of test equipment that is occassionally very useful in the computer laboratory is a frequency counter. This article explains how to use the capabilities of the KIM-1, with a minimum of additional hardware, to provide the functions of such an instrument. The frequency counter described operates over the audio range from 500 Hz to above 15 KHz. To reduce the amount of external hardware needed, the design assumes TTL level input signals. However, the addition of a small amount of analog hardware to the design presented would allow the counter to be used with analog signal sources.

Basic Counter Mechanization

In order to develop a frequency counter from the KIM-1 microcomputer it is necessary to count and display the number of input pulses detected over a specific time interval. The basic time interval chosen was 100 milliseconds. This time interval is established by using one of the two interval timers available on the KIM-1. Transitions in the applied waveform are sensed by the external logic and force non-maskable interrupts to the KIM. As each interrupt is detected a memory location is incremented. Because of the availability of the decimal mode in the 6502 instruction set, the count can be maintained in decimal rather than binary or hexadecimal form. At the conclusion of the 100 millisecond interval the accumulated count is loaded into the display registers and the process is repeated. Figure 1 is a flow chart of the frequency counter program.

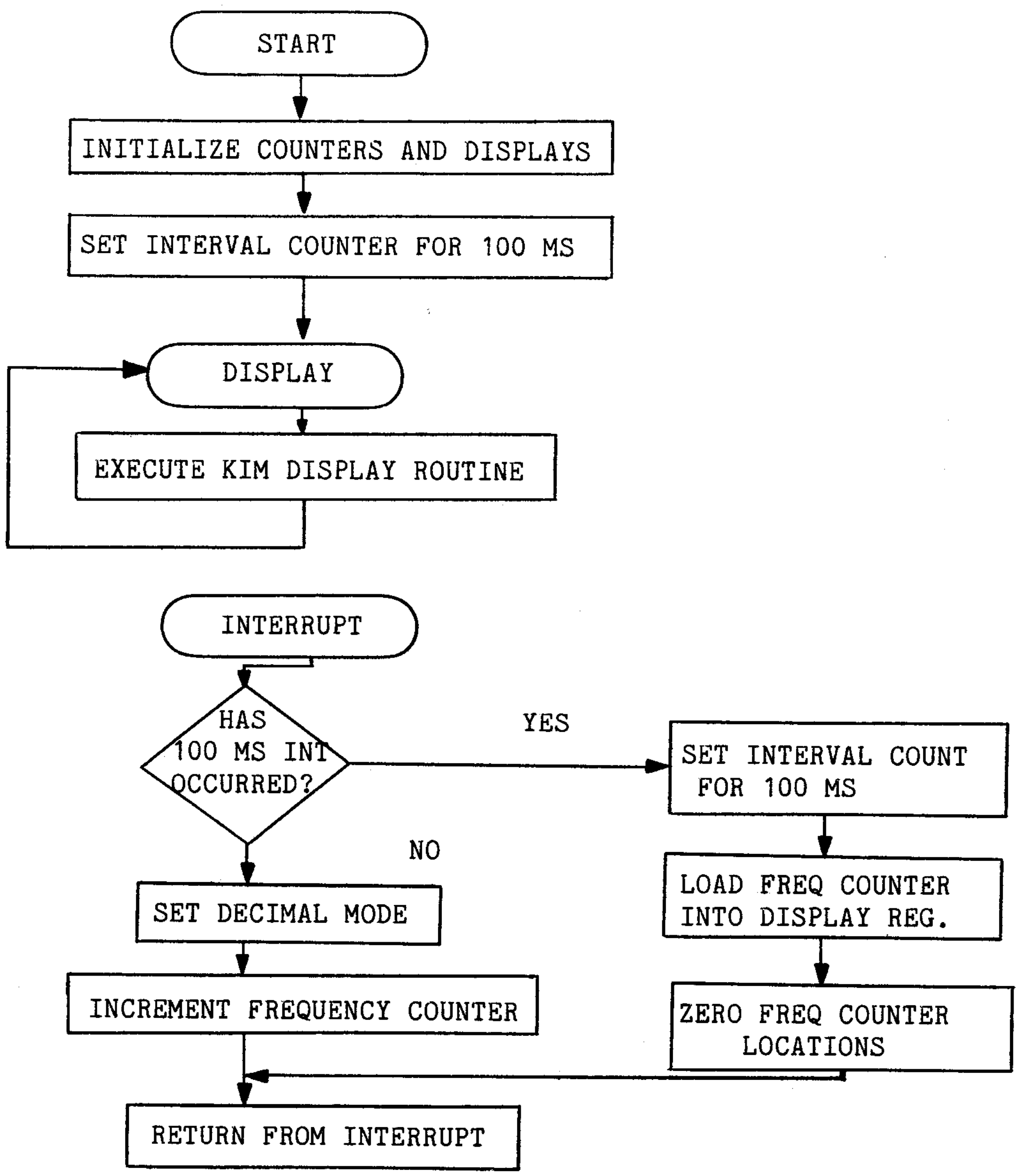
Detailed Software Description

As shown in the flow chart (Figure 1) and in the program listing (Figure 2) the program is started at location 0005 and the frequency counter memory location and display locations are initialized to zero. A Value of 99. is loaded into the interval counter at location 1747. A value stored at this location is decremented every 1024 microseconds. Under these conditions a zero register value will then be realized 101.376 milliseconds after the register is loaded.

After the initialization process the program goes into an idle loop called DISPLAY and waits for an interrupt to occur. The DISPLAY program consists of repeated calls to the KIM display routine which presents the contents of the display registers 00FA and 00F9 on the seven segment display LEDs.

When an IRQ interrupt is sensed, the KIM logic forces program control to the address stored in memory locations 17FE and 17FF. In this mechanization, the value stored in these locations will direct program control to be transferred to the start of the interrupt routine (location 0021). The interrupt program first stores away the values of A and X from the interrupted program. The contents of the interval timer register, location 1746, is then read to establish if the 100 millisecond interval has been completed. A non zero number indicates that the counter is still counting and an input pulse transition has been detected. The logic sets the processor into the decimal mode and adds one to the contents of the frequency counter location. As we wish to detect values above 1 KHz, a second frequency counter register must be employed to count the overflow from the least significant two decimal digits. Having completed the incrementation process, the program restores the the values of A and X and returns to the interrupted program by executing the RTI instruction.

If a zero value is observed when the interval timer register is read, then the 100 millisecond timing interval has been completed. The program reloads the 100 millisecond value into the interval counter, loads the accummulated count in the frequency counter memory locations into the appropriate display



FLOW DIAGRAM FOR FREQUENCY COUNTER PROGRAM

Figure 1

```
                         ORG    $0005

                    INTGER *    $00FA            Figure 2
                    FRACT  *    $00F9
                    PBDD   *    $1703
                    CLOCKX *    $1746
                    CLOCK  *    $1747
                    SCANDS *    $1F1F

0005 A9 00          START  LDAIM $00    INIT COUNTERS AND DISPLAY
0007 85 51                 STAZ  CNTONE
0009 85 52                 STAZ  CNTTWO
000B 85 FA                 STAZ  INTGER
000D 85 F9                 STAZ  FRACT
000F 8D 03 17              STA   PBDD
0012 A9 62                 LDAIM $62    SET UP 100 MILLISECOND TIMER
0014 8D 47 17              STA   CLOCK

0017 20 1F 1F       DISPLA JSR   SCANDS DISPLAY DATA
001A 4C 17 00              JMP   DISPLA CONTINUOUSLY

0021                       ORG   $0021

0021 48             INTRPT PHA          SAVE A REGISTER
0022 8A                    TXA          SAVE X REGISTER
0023 48                    PHA
0024 AD 46 17              LDA   CLOCKX TEST CLOCK TIMED OUT
0027 30 11                 BMI   MILLI  TEST OF 100 MILLISECONDS

0029 F8             COUNT  SED          SET DECIMAL MODE
002A 18                    CLC          CLEAR CARRY BIT
002B A5 51                 LDAZ  CNTONE GET FRACTIONAL PART
002D 69 01                 ADCIM $01    INCREMENT
002F 85 51                 STAZ  CNTONE
0031 A5 52                 LDAZ  CNTTWO ADD CARRY BIT IF SET
0033 69 00                 ADCIM $00
0035 85 52                 STAZ  CNTTWO
0037 4C 4D 00              JMP   EXIT

003A A9 62          MILLI  LDAIM $62    RESET CLOCK
003C 8D 47 17              STA   CLOCK
003F A5 51                 LDAZ  CNTONE MOVE DATA TO DISPLAY
0041 85 F9                 STAZ  FRACT
0043 A5 52                 LDAZ  CNTTWO
0045 85 FA                 STAZ  INTGER
0047 A9 00                 LDAIM $00    RESET COUNTERS
0049 85 51                 STAZ  CNTONE
004B 85 52                 STAZ  CNTTWO

004D 68             EXIT   PLA          RESTORE X REGISTER
004E AA                    TAX
004F 68                    PLA          RESTORE A REGISTER
0050 40                    RTI          RETURN FROM INTERRUPT

0051 00             CNTONE =     $00    FRACTIONAL COUNTER
0052 00             CNTTWO =     $00    INTEGER COUNTER
```

registers, and then zeros the contents of the frequency counter locations. The interrupt program is exited by restoring the values of A and X and returning via the RTI instruction.

The Hardware Configuration

Figure 3 illustrates the additional logic required to use the KIM as a frequency counter and shows how that logic is connected to the KIM Expansion connector. The purpose of the 74121 monostable multivibrator is to produce a negative going pulse of short duration onto the IRQ interrupt lines whenever the input to that chip experiences a high-to-low transition. It should be noted that the IRQ is a level rather than an edge sensitive interrupt and that the interrupt line must be held low only long enough to allow the processor to sense the interrupt. Therefore, with the addition of this flip-flop the KIM will experience an IRQ interrupt each time the input source exhibits a high-to-low transition. If a periodic pulse train is being applied to the input, then an IRQ interrupt will be experienced on each cycle.

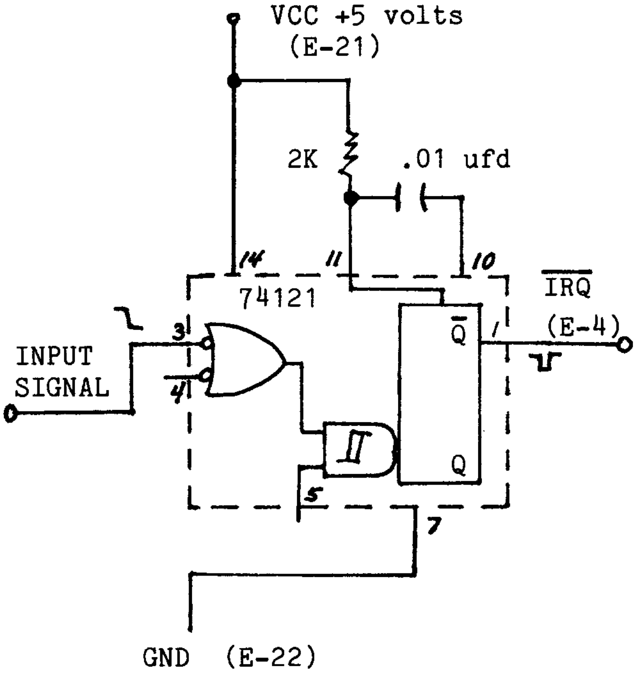


Figure 3

The accuracy of this hardware/software on a KIM-1 for measuring frequencies is shown in the table (Figure 4). A very accurate frequency meter was used to obtain the meter measurements. Since there are probably slight variations in the speed of different KIM-1s, you should calibrate your own unit before using it for any "real" measurements.

Frequency Calibration

Meter	KIM
14.960	15.00
13.961	14.00
12.960	13.00
11.968	12.00
10.966	11.00
9.965	10.00
8.970	9.00
7.977	8.00
6.984	7.00
5.983	6.00
4.985	5.00
3.992	4.00
2.991	3.00
2.003	2.00
1.003	1.00
.902	0.90
.801	0.80
.705	0.70
.608	0.60
.507	0.50

Figure 4

Additional Comments

In addition to entering the values shown in the accompanying listing, the values 0010 should be stored in locations 17FA and 17FB, and 2100 should be stored in locations 17FE and 17FF. The latter value directs program control to the beginning of the interrupt routine when an IRQ is sensed.

The results displayed on the seven segment indicators will be in the form XX.XX KHz. This format was chosen for convenience and the range can be shifted for higher accuracy by software modifications. Additional improvements are left to the reader to create. The author would appreciate being informed of any interesting improvments you come up with.

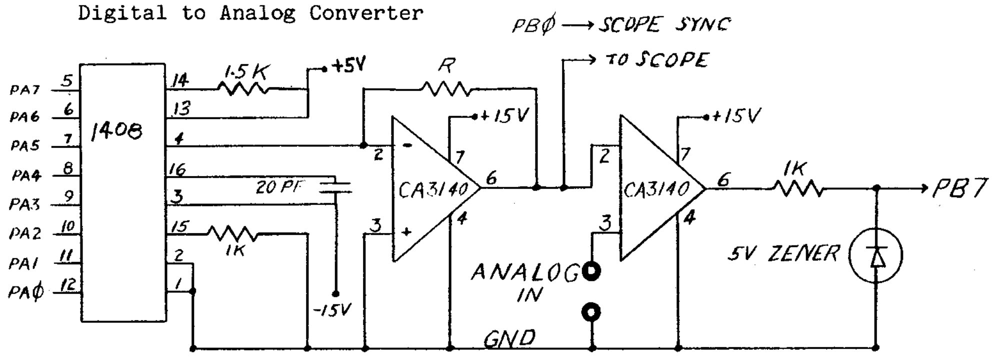
DIGITAL-ANALOG AND ANALOG-DIGITAL CONVERSION

USING THE KIM-1

Marvin L. De Jong
Department of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

A Motorola 1408 8-bit digital to analog converter is connected as shown in the circuit diagram. (The 1408 is available from James Electronics, 1021 Howard Ave., San Carlos, CA 94070, as are the op amps used in these experiments.) The PAD port of the KIM is used to provide the digital input to the 1408. The analog output of the 1408 is a current sink at pin 4, which we converted to a voltage by means of the RCA CA-3140 operational amplifier. The feedback resistor R is adjusted to give the desired voltage output. For example, an R of about 500 ohms gives a voltage range from 0 volts when PAD is 00000000 to 1 volt when PAD is 11111111.

Circuit Diagram for
Digital to Analog Converter



1. Generation of a Ramp Voltage Waveform

For the first experiment do not connect the second op amp, simply connect the output of the first op amp to an oscilloscope as shown. Load the following program.

Program to Generate a Ramp Voltage Waveform

ADDRESS	OPCODE	LABEL	INSTRUCTION		COMMENTS
0300	A9 FF	START	LDAIM	FF	255 in Accumulator
0302	8D 01 17		STA	PADD	Port A is Output Port
0305	EE 00 17	BACK	INC	PAD	Increment number in PAD
0308	4C 05 03		JMP	BACK	Increment in a Loop

Running this program should cause a ramp waveform to be observed on the oscilloscope screen. A close examination of the ramp will show that it consists of $2^8 = 256$ steps rather than a straight line.

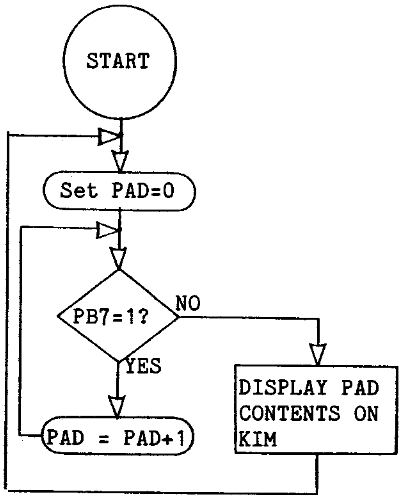
2. A DAC as an Analog to Digital Converter

The second op amp acts as a comparator. It compares the voltage from the output of the first op amp (which we shall call the digital signal) with a voltage from some source to be applied to pin 3 (which we shall call the analog signal). The output is connected to PB7 on the KIM. If PB7 = 1, the analog signal is greater than the digital signal. If PB7 = 0, the analog signal is less than the digital signal. The digital signal is, of course, produced by the contents of PAD.

A flow chart showing what we intend to do is shown below. Output port PAD is set to zero. If the analog signal is positive the PB7 = 1. PAD is now incremented until the comparator indicates that the analog signal is less than the digital signal, i.e., PB7 = 0. At that instant the digital and analog signals are the same to within one bit, the least significant bit, on PAD. The digital value of PAD is then displayed and the cycle continues.

If the feedback resistor is adjusted so that a value of PAD = 255_{10} = FF_{16} produces a voltage of 2.55 volts, then we have constructed a simple digital voltmeter with a full scale reading (in hex) of 2.55 volts. The extremely high impedance of the 3140 op amp makes this a rather good voltmeter. A simple program to convert from hex to base ten would make the meter easier to read.

Flow Chart for
Analog to Digital Converter



Program for Analog to Digital Converter
(Ramp Approximation)

ADDRESS	OPCODE	LABEL	INSTRUCTION		COMMENTS
0300	A9 FF	START	LDAIM	FF	255 in Accumulator
0302	8D 01 17		STA	PADD	Make Port A Output Port
0305	A2 00	AGN	LDXIM	00	Start PAD at zero
0307	8E 00 17	RAMP	STX	PAD	Output Value of X register
030A	AD 02 17		LDA	PBD	Read Port B
030D	10 04		BPL	DISP	Branch if bit 7 = 0
030F	E8		INX		Increment X register
0310	4C 07 03		JMP	RAMP	Continue loop
0313	86 F9	DISP	STX	INH	Put X into Display register
0315	20 1F 1F		JSR	SCANDS	Use KIM Display Subroutine
0318	4C 05 03		JMP	AGN	and start again at zero

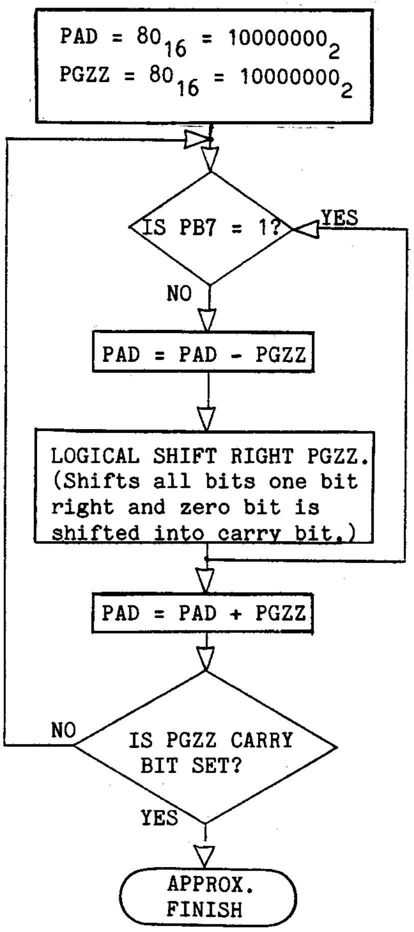
3. Successive Approximation Analog to Digital Used as a Storage Scope.

The ramp approximation is quite slow and there is a faster technique known as "successive approximation." It works as follows: the most significant bit to the DAC is set to one and all the others are set to zero. If the comparator indicates that the analog signal is greater than the digital signal, the next lower bit is set to 1 and the test is repeated. If the comparaactor indicates that the analog signal is less than the digital signal, the highest bit is made zero, and the next lower bit is set to 1 and the test is repeated. This iterative process is repeated until all eight bits have been tested, starting with the MSB and ending with the LSB. The flow chart indicates how this will be accomplished.

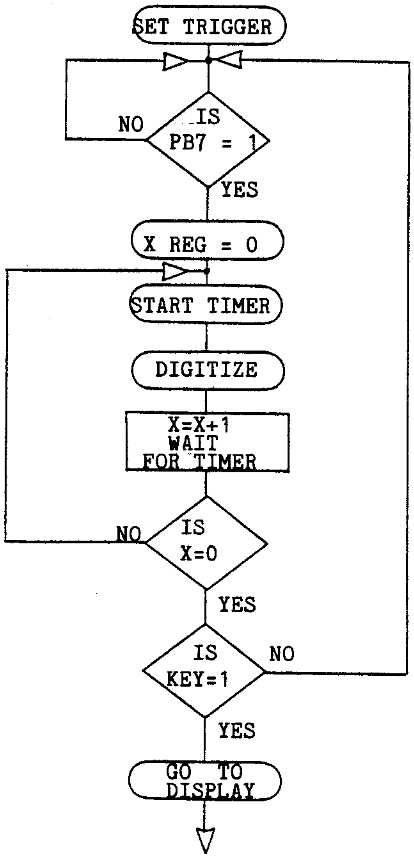
This analog to digital conversion scheme will be used in a program which digitizes 256 points on a waveform and then stores the results, to be displayed at a convenient time and with as many repetitions as desired on an oscilloscope. It is useful for examining slow waveforms with an oscilloscope with a low persistance screen, for example ECG waveforms, and it is useful for examining non-periodic waveforms such as a one-shot impulse from an accelerometer. The program has triggering built in, and the output scan portion synchronizes the oscilloscope with a sync signal, turning an inexpensive scope into something more useful. A flow chart for the program is given below.

A short description of the behavior of the circuit and program follows. The experimenter chooses the desired trigger level and loads this into location 0306. When the analog signal is greater than this, the comparator makes PB7 go high and the scan begins. The sampling rate and the scan time is determined by the number loaded into the timer and the timer used; locations 0314 and 0316, respectively. It takes at least 200 microseconds to digitize so there is no point in choosing time intervals smaller than this. X is used as an index to identify each of the 256 points on the scan. After the timer is started the analog signal is digitized and the timer is watched until it is finished. X is then incremented and a new point is digitized

Flow Chart for
Successive Approximation
Analog to Digital Conversion



Flow Chart for Storage Scope



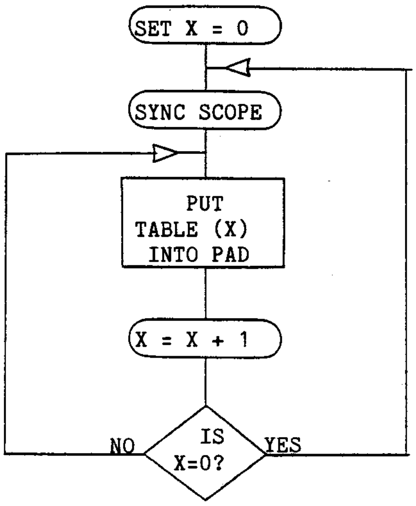
until all 256 points are finished and stored in TABLE,X.

X is then zero again. This entire process will repeat unless the 1 key is depressed, in which case the program displays the data on the oscilloscope, connected as before to the output of the first op amp. The display will repeat, complete with SYNC signal output from PB0, until the program is halted. In our case we loaded the vector 17FA and 17FB with the starting address of the program (0300) so a depression of the ST key caused the entire program to start over.

A listing of the program is shown on the following page. Notice that the data is stored in TABLE,X located in page 2 of memory, PGZZ is at location 0000, the trigger level is in 0306 and the scan time variable is in 0314 and 0316. The scan time should not be shorter than 200 microseconds. As far as display is concerned, we found that a sweep rate of 200 to 500 microseconds per cm gave good results.

A few other comments may be in order. First, most of the ideas for this project were obtained in a KIM workshop offered by Dr. Robert Tinker. The software implementation is the author's work. There are some obvious improvements, such as a sample and hold device between the analog source and the comparator or a faster approximation routine. These improvements are left for the reader to implement. The author would be glad to be informed if such improvements are made.

Flow Chart for Display



Program for Storage Scope

ADDRESS	OPCODE	LABEL	INSTRUCTION		COMMENTS
0300	A9 FF	BEGIN	LDAIM	FF	Initialize Port A to Output
0302	8D 01 17		STA	PADD	
0305	A9 10	START	LDAIM	TSET	Trigger Voltage Set
0307	8D 00 17		STA	PAD	
030A	A2 00		LDXIM	00	Initialize X register
030C	EA		NOP		
030D	EA		NOP		
030E	AD 02 17	TRIG	LDA	PBD	Tinput and test PB7
0311	10 FB		BPL	TRIG	Wait if PB7 = 0
0313	A9 C0	STIME	LDAIM	C0	Set Scan Time here
0315	8D 05 17		STA	TIMER	Select Interval Timer
0318	A9 80		LDAIM	80	Start Digitize Sequence
031A	85 00		STAZ	PGZZ	Store Initial Value
031C	8D 00 17	TEST	STA	PAD	Output Value
031F	AC 02 17		LDY	PBD	Test PB7
0322	30 03		BMI	FWRD	Branch if PB7 = 1
0324	38		SEC		Clear Borrow Flag
0325	E5 00		SBCZ	PGZZ	Subtract bit 7
0327	46 00	FWRD	LSRZ	PGZZ	Set PGZZ for Next Lower Bit
0329	B0 08		BCS	OUT	Out of Digitize Loop if Finished
032B	65 00		ADC	PGZZ	Set Next Lower Bit = 1
032D	4C 1C 03		JMP	TEST	Return to Test all Lower Bits
0330	8D 00 17	OUT	STA	PAD	Final Approximation in PAD
0333	9D 00 02		STAX	TABLE	and in TABLE(X) in Page 2
0336	E8		INX		Bump Table Index
0337	F0 08		BEQ	DISPLY	Go to Display if Table Complete
0339	AD 07 17	CHEK	LDA	TCHEK	Test if Timer is Finished
033C	10 FB		BPL	CHEK	If not, Wait in Loop
033E	4C 13 03		JMP	STIME	Digitize another Point
0341	20 6A 1F	DISPLY	JSR	GETKEY	Is Key 1 Depressed?
0344	C9 01		CMPIM	01	
0346	F0 03		BEQ	SYNC	Yes. Display the Data
0348	4C 05 03		JMP	START	No. Return to Start
034B	A9 01	SYNC	LDAIM	01	Set up PB0 as Sync
034D	8D 03 17		STA	PBDD	Output Pin
0350	A2 00		LDXIM	00	Init X to Display Table
0352	AD 02 17	RPT	LDA	PBD	Toggle PB0 for Sync
0355	49 01		EORIM	01	Signal to Scope
0357	8D 02 17		STA	PBD	
035A	BD 00 02	SCAN	LDAX	TABLE	Output Table(X) for
035D	8D 00 17		STA	PAD	Display on Scope
0360	E8		INX		Increment X register
0361	D0 F7		BNE	SCAN	Continue until all Points Done
0363	4C 52 03		JMP	RPT	Then Repeat

NOTE: This material was submitted by the author to the KIM-1 User Notes and has also been distributed by MOS Technology as "KIM Application Note #11701." It is printed here with the permission of the KIM-1 User Notes and MOS Technology.

MAKING MUSIC WITH THE KIM-1

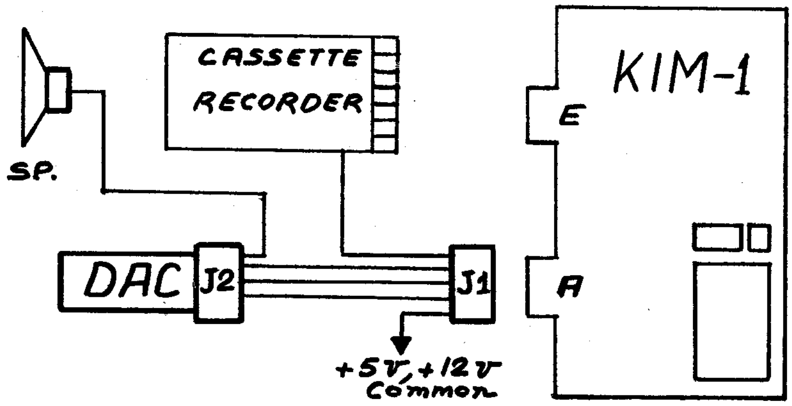
Armand L. Camus
P.O. Box 294
Westford, MA 01886

What kind of music can you make with the help of a microcomputer, namely the KIM-1 with its 1.1K bytes of memory? Well, it certainly will not sound like the Boston Symphony Orchestra, live or on records, but with the right type of music it will give an acceptable rendition of a chosen piece of music. Many elements of good music will be missing, especially the timbre of the different instruments of the orchestra, but on the positive side the notes will be on tune, you will be able to compose in four-part harmony, the tempo will be adjustable, and the whole process will permit some of the artistic creativity which may hide in each of us to emerge to the surface. Last, but not least, it will be a lot of fun.

This elementary article explains the "HOW-TO" rather than the "WHY" in making music with a microcomputer. Many of the hobbyists who may find it too simple may refer to the excellent article by Hal Chamberlin which dwells in detail on the subject.

An easy way for the beginner to start his musical career is to acquire a minimum of equipment besides the KIM-1 and cassette recorder it is assumed are already in his possession.

The DAC unit is a printed circuit board containing a complete audio output system for the KIM-1. This board also comes with a cassette tape, an instruction sheet listing the songs which can be loaded in the KIM, and reprint of the reference article including the interconnections to be made between the two connectors.



J1, J2 connectors: Vector R644, Winchester HKD2250, or equivalent. J2 will be too long, but will work just the same.
Speaker, 2 1/2", 8 ohm, 0.3W, from Radio Shack, or equivalent.

Now that we have described the hardware we will concentrate on what to do in order to get some music out of the system. The simplest way at this time is to load File 1 and File 2 of the tape and to see if the Star Spangled Banner comes out clear and patriotic. The procedure is simple:

Start the KIM-1 and press the appropriate keys to get:

```
AD  00F1 DA  00
AD  17F9 DA  01
AD  1873 Press GO
```

Start the cassette until you get 0000 in the address display, which indicates that the loading was done properly. After stopping the cassette, press the keys to get:

```
AD  17F9  DA  02
AD  1873  Press GO
```

Start the cassette again until you get 0000. Stop the cassette. Now you are ready. Press AD 0100, press GO and the song will be played. As it stops, the program resets the address AD to 0100, so by pressing GO again, the song will repeat itself.

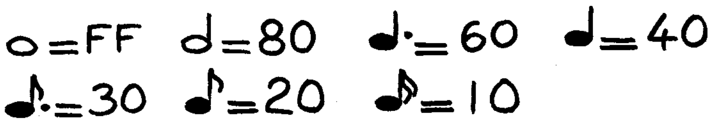
In the same manner you could load Files 3 and 4 to get a rendition of Exodus. The sound quality may be changed by loading File 5 or File 6. Personally, I prefer File 6 which has a much more mellow timbre.

Transcribing a Song

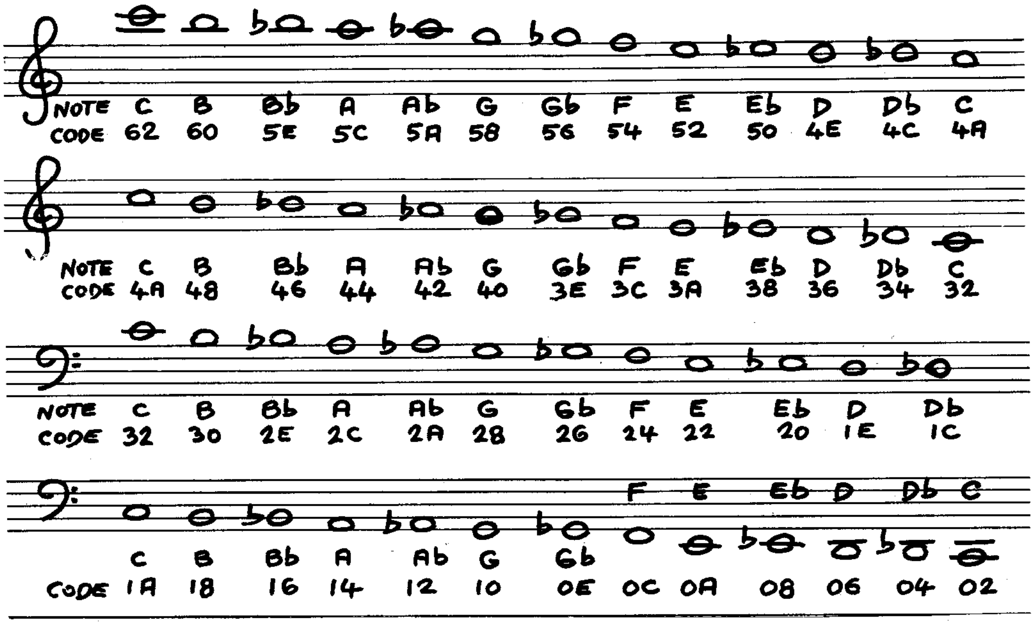
Now that we have gone through the above steps, we will learn to code a song. For our purpose, a particular note of music will have two characteristic elements:

its pitch, represented by its position on the staff;
its duration, relative to other notes.

1. Duration Code:
We will assign a two-digit code to the duration of a note:



2. Pitch Code:



What we mean is that a half note lasts twice as long as a quarter note, a quarter note lasts twice as long as an eighth note, etc...We are not talking about tempo yet, this will come later.

With the help of this lookup table we can find quickly the code for any note within the limits of C6 and C2, the high and low C's. However, the very low notes may not be reproduced too well with a small speaker and it may not be advisable to go below C3 (Code 1A).

Coding a Song

The program given at the end of this article is a coding of the well-known carol "Deck the Halls", which we thought would be appropriate for the Christmas Issue. If you look at this coding, you will observe that it is done line by line. Each line is composed of six elements. For example, the first line is:

0200 60 4A 44 32 24

- the 0200 is the memory address of the element 60. The next element, 4A, would then have an address 0201, and so on.
- the 60 is the duration of the group of four notes which follow. A 60 means a dotted quarter note.
- the 4A is the note C, for the first voice.
- the 44 is the note A, for the second voice.
- the 32 is the note C, for the third voice.
- the 24 is the note F, for the fourth voice.

This is an F major chord which could be represented as in (1), and it corresponds to the word "DECK" of the song.

Now we will code the first bar of the song. Remember that each line will have the same format:

address (4 digits), duration (2 digits), 1st voice (2 digits), 2nd voice (2 digits), 3rd voice (2 digits), and 4th voice (2 digits) for a total of fourteen (14) digits. If a voice is quiet, use 00 at the appropriate location.

The first vertical group of notes (C,A,C,F) corresponding to the word "DECK" has already been explained above.

The second vertical group of notes corresponding to the word "THE" is made of B flat, G, C, and E. Looking up the pitch code table, we find the following codes:

Bb = 46, G = 40, C = 32, and E = 22. Each note is an eighth note so the duration code is 20. The address of the duration code is 0205 so our second line will be:

0205 20 46 40 32 22

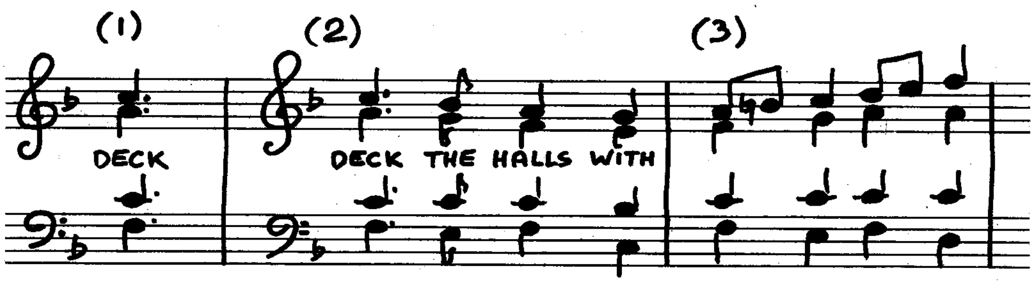
In the same fashion the two other vertical groups are made of quarter notes (code 40) and we get for the first bar:

```
0200 60 4A 44 32 24 (DECK)
0205 20 46 40 32 22 (THE)
020A 40 44 3C 32 24 (HALLS)
020F 40 40 3A 2E 1A (WITH)
```

Remember that there is a Key Signature in this carol and that all the B's, wherever located on the staff, are flat, unless otherwise indicated, which explains the 46 of the second line and the 2E of the fourth line.

Another part of that song is shown in the example (3). The first voice plays two notes (A and B natural), while the other voices play only one. We solve this problem by writing two lines, one for the A and one for the B natural, repeating the other notes to extend their duration to a quarter note. We get:

```
02D2 20 44 3C 32 24
02D7 20 48 3C 32 24
```



Both A note (code 44) and B natural note (code 48) have only the duration of one eighth note each (code 20), and we have to write two sepprate lines for them, but the three other notes will be repeated so that their total duration is a quarter note. Fortunately, the lower notes, even when repeated, will blend together and and sound more like a quarter note than two consecutive eighth notes.

Now we should be able to code a song, but as a preliminary exercise, you may want to load "Deck the Halls" and see how it works out. Here is the procedure:

Load Files 01 and 02 of the DAC tape, as explained at the beginning of this article. You may also want to load File 06 to give a more mellow timbre. Then go to address 0200 and start inputting the data. The addresses in the left side give you a check on your progress and catch possible omissions of data. What we are doing here is using the main program and writing over the song already in memory. At any time it is possible to go back to AD 0100, push GO and listen to what is already in memory. Somewhere at about 2/3 of the song, we run out of memory (0200 to 02F9), but we have enough left to tell our microcomputer that it is the end of that particular segment (02FA 01), and that we wish to continue at address 0083 (02FB and 02FC). At the very end, check address 00DD 00. The data 00 indicates the end of the piece and this will reset the KIM-1 to address 0100, ready to "GO", so to speak.

After you have loaded the code and pushed the GO key, the carol should start, sounding good if no mistake was made, but perhaps a little bit on the slow side. To change the tempo. either way, go to address 001D and the data will probably show 60. Change the data to 40, go back to address 0100, push GO and the tempo will be much faster. Experiment with the data at AD 001D and find the tempo you prefer.

I have found out that while I am coding I like to listen to what is already in memory, because a simple mistake at the beginning, especially forgetting one voice or the duration code, will throw the rest out of whack. Starting the song at the beginning, when it is already correct is a waste of time, but it is possible to start the song at some other point. However, it must always be at one of the duration addresses shown at the end of this article. If not, the KIM-1 would interpret the duration code as a musical note and vice-versa! The starting address is contained in locations 0017 and 0018. To start, for example, at address 0237, go to address 0017 read 00, 0018 read 02. This means that the song normally starts at 0200. All we have to do is change the data to read:

```
AD 0017 DA 37
AD 0018 DA 02
```

Then setting address 0100 and pushing GO will cause the song to start at location 0237 every time.

Available Memory

The memory available to the user is divided in two groups, each group not necessarily in consecutive order. First group is associated with the music program, frequency table or the notes, KIM, etc...Second group is associated with the song. The actual layout of the memory is as follows:

```
0000 to 001E Program variables
001F to 0082 Note frequency table
0083 to 00EE Song, second part
00EF to 00FF KIM variables
0100 to 01AA Music program
01AB to 01F3 Song, third part
01F4 to 01FF 6502 Stack
0200 to 02FF Song, first part
0300 to 03FF Waveform (voice) table
1780 to 17E4 Sone, fourth part
```

If your music score extends beyond the first part locations, you have to provide room for continuation. Assuming a score uses all of the available memory space for coding a song, the following locations are important:

Use of Location	Part 1	Part 2	Part 3	Part 4
Beginning of Part (Song)	0200	0083	01AB	1780
Beginning of Last Line	02F5	00E7	01EC	17DF
Last note of Last Line	02F9	00EB	01F0	17E3
End of Sequence (Song)	02FA (01)	00EC (01)	01F1 (01)	17E4 (00)
Low Address Next Segment	02FB (83)	00ED (AB)	01F2 (80)	
High Address Next Segment	02FC (00)	00EE (01)	01F3 (17)	

Reference: Chamberlin, Hal, "A sampling of Techniques for Computer Performance of Music", BYTE Magazine, Sept. 1977, pp. 62-83.

Score for "Deck the Halls"

```
0200: 60 4A 44 32 24
0205: 20 46 40 32 22
020A: 40 44 3C 32 24
020F: 40 40 3A 2E 1A
0214: 40 3C 32 2C 1E
0219: 40 40 3A 32 1A
021E: 40 44 3C 32 24
0223: 40 3C 32 2C 24
0228: 20 40 3A 32 1A
022D: 20 44 3C 32 1A
0232: 20 46 40 32 1A
0237: 20 40 3A 32 1A
023C: 60 44 3C 32 24
0241: 20 40 36 2E 16
0246: 40 3C 32 2C 1A
024B: 40 3A 32 28 1A
0250: 80 3C 32 2C 24
0255: 60 62 5C 32 24
025A: 20 5E 58 32 22
025F: 40 5C 54 32 24
0264: 40 58 52 2E 1A
0269: 40 54 4E 2C 1E
026E: 40 58 52 32 1A
0273: 40 5C 54 32 24
0278: 40 54 4A 2C 24
027D: 20 58 52 32 1A
0282: 20 5C 54 32 1A
0287: 20 5E 58 32 1A
028C: 20 58 52 32 1A
0291: 60 5C 54 32 24
0296: 20 58 4E 2E 16
029B: 40 54 4A 2C 1A
02A0: 40 52 4A 28 1A
02A5: 80 54 4A 2C 24
02AA: 60 40 3A 32 1A
02AF: 20 44 3C 32 1A
02B4: 40 46 40 32 1A
02B9: 40 40 3A 32 1A
02BE: 60 44 3C 32 24
02C3: 20 46 3C 28 24
02C8: 40 4A 3C 2C 24
02CD: 40 40 40 32 22
02D2: 20 44 3C 32 24
02D7: 20 48 3C 32 24
02DC: 40 4A 40 32 22
02E1: 20 4E 44 32 24
02E6: 20 52 44 32 24
02EB: 40 54 44 32 1E
02F0: 40 52 40 32 28
02F5: 40 4E 3C 30 28
02FA: 01
02FB: 83
02FC: 00

0083: 80 4A 3A 28 1A
0088: 60 4A 44 32 24
008D: 20 46 40 32 24
0092: 40 44 3C 32 24
0097: 40 40 3A 2E 1A
009C: 40 3C 32 2C 1E
00A1: 40 40 3A 2E 1A
00A6: 40 44 3C 32 24
00AB: 40 3C 32 2C 24
00B0: 20 4E 3C 2E 16
00B5: 20 4E 3C 2E 16
00BA: 20 4E 3C 2E 16
00BF: 20 4E 3C 2E 16
00C4: 60 4A 3C 2C 24
00C9: 25 46 3C 36 28
00CE: 50 44 3C 32 1A
00D3: 60 40 3A 2E 1A
00D8: 95 3C 32 2C 24
00DD: 00
```

A COMPLETE MORSE CODE SEND/RECEIVE PROGRAM FOR THE KIM-1

Marvin L. De Jong, K0EI
Dept. of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

I. INTRODUCTION

The program described below will convert ASCII from a keyboard to a Morse code digital signal which can be used to key a transmitter. It will also convert a Morse code digital signal to ASCII for display on the user's video system. Suitable references for circuits to convert the audio signal from a communications receiver to a digital Morse signal are also given. [1,2]

The entire program resides in the memory on the KIM-1, and has the following features:

1. The precise code speed in words per minute can be entered at any time from the keyboard. Key in CONTROL S followed by any two-digit decimal number from 05 to 99 words per minute.

2. The operator can type as many as 256 characters ahead of the character currently being sent. One page of memory is devoted to a FIFO buffer.

3. When there are less than 16 characters left in the buffer, the KIM-1 display indicates how many characters are left (F to 0 hex).

4. Backspace capability is provided. CONTROL B erases the last character entered into the buffer, and the operator then enters the correct character.

5. The buffer can be pre-loaded with as many characters (up to 256) as desired while the program is in the receive mode. Pressing CONTROL G starts the program sending code as soon as the operator is ready.

6. CONTROL R sends the program from the send mode to the receive mode.

7. While in the receive mode the display on the KIM-1 informs the operator to either increase the code speed (F, for faster, on the display) or decrease (S, for slower) the speed for proper reception. The receive program actually tolerates a large range in code speeds with no adjustment.

8. The feature just mentioned can be used to measure the "other guy's" code speed.

9. If the receive mode is not used, any CONTROL key not mentioned above will put the program in an idle loop so the buffer can be loaded. CONTROL G starts the message.

10. The carriage return key restarts the send program, or it can be returned from the receive mode to the send mode with CONTROL G.

The KIM-1 was first programmed to send code by Pollock [3], and some of the features of his program are found here. Pollock [4] has also described a microprocessor controlled keyboard using the 6504. It has more features than his original program written for the KIM-1, but the program described here has some additional features which are very attractive, especially the receive program.

II. BACKGROUND

A. Sending Morse Code (ASCII to Morse)

A negative going 10 microsecond strobe pulse from the keyboard is connected to the NMI pin on the KIM-1. Whenever a key is pressed an NMI interrupt occurs and the ASCII code from the keyboard is read at the lowest 7 pins of port A (PAD). The eighth bit is held high, so the number read is actually the ASCII code plus 80 hex. This number is stored in the FIFO buffer which is page 2 of memory on the KIM-1. The send routine uses the numbers in the FIFO memory to index a location in page zero which contains the information to construct the Morse character.

An illustration will make this clear. The ASCII hex representation of the letter C is 43. The strobe pulse causes port A to be read, which results in the number C3 (C3 = 43 + 80) being stored in the FIFO. When the send routine gets to the location in the FIFO where C3 is stored, it uses it to locate the contents of address 00C3. In location C3 in zero page is found 1A which is 00011010 in binary. The most significant 1 is simply a bit which indicates that all lesser significant bits contain the code information, namely 1 = dash and 0 = dot. Thus, C is dash-dot-dash-dot (1010).

The program causes the 00011010 to be rotated left (ROL) until a 1 appears in the carry position. The carry flag set causes the program to analyze the remaining bits for their code content. It does this by successively rotating them (ROL) into the carry position. If a 1 appears in the carry position, PB0 is held at logical 1 for the appropriate time followed by a space while PB0 is at logical 0. If a 0 appears in the carry position a dot is sent, followed by a space. When a total of 8 ROL commands have been completed, counting those needed to find the leading 1, then PB0 is held at logical 0 for an additional time to give a character space. The space bar produces still more time at logical 0 to produce a word space.

CONTROL S changes the NMI interrupt vectors so that the next two characters (hopefully decimal digits) from the keyboard are read, converted from base ten to hex [5], and converted to the basic time unit (see below). The interrupt vectors are then restored so that further characters from the keyboard are read as usual. Control characters are obtained by pressing the control key followed by the appropriate control character.

B. Timing Considerations.

Before going much further, the timing calculations will be described. Morse code is a variable length code. That is, the number of bits is variable as contrasted to a fixed bit-length code such as ASCII. Its structure is based on the time duration of the various components as follows:

Mark Elements:

Dot = 1t
Dash = 3t

Space Elements

Element space = 1t
  (time between dots and dashes)
Character space = 3t
  (time between letters)
Word space = 7t
  (time between words)

The time t depends on the code speed. According to **The Radio Amateur's Handbook** a code speed of 24 words per minute (wpm) corresponds to 10 dots per second. Since there are 10 element spaces included in the 10 dots per second, there are a total of 20 t in one second: that is, t = 1/20 second at 24 wpm. At any other speed then

$$\begin{aligned} t &= (1/20)(24/S) \\ &= (50 \text{ ms})(24/S) \\ &= (1200/S) \text{ in milliseconds (ms)} \end{aligned}$$

where S is the code speed in wpm. If the divide-by-1024 timer on the KIM is used, 1 count corresponds to 1.024 ms. The number T (called TIME in the program) to be loaded into the timer is then

$$\begin{aligned} T &= (1172/S) \text{ base ten or} \\ &= (494/S) \text{ hex.} \end{aligned}$$

The speed S in wpm is entered in decimal from the keyboard, converted to base 16 (hex), sent to a divide routine to find T, and T is stored at 0000 in memory. 99 wpm gives 0C hex in TIME while 05 wpm gives EB hex. Care was taken in developing the above calculations because of a discrepancy between it and the results given by Pollock[4].

The system timing was tested by comparing it with code sent by W1AW. The speeds are the same to better than one word per minute from 5 wpm to 35 wpm.

In the receiving program a word space is detected when a space counter exceeds 5T. At moderate code speeds 5T is greater than 255 resulting in an overflow. Consequently, in the receive program 1/2T is used as the basic time unit. In this case, speeds as low as 12 wpm can be received. At slower speeds the system still works, but word spaces occur between each letter.

C. Receiving Morse Code (Morse to ASCII)

To receive Morse code and convert it to ASCII, the inverse of the above process is carried out. It is assumed that a suitable audio detection circuit [1,2[produces a logical 1 for a space element and a logical 0 for a mark element. This digital Morse signal is applied to PB7 and the IRQ pin on the KIM-1. A character register begins with a 1 in the zero bit position. Each time a dot is received the character register is shifted left and a zero is loaded into the character register. Each time a dash is received the character register is shifted left and a one is loaded into the zero bit position. Thus, when a character space is detected, and a C (for example) has been received, the character register will contain 1A, just as in sending a C. However, the 1A is used to index a zero page location which contains the ASCII code for C, namely 43. The various components are identified by timing their duration.

III. THE PROGRAMS

A detailed listing of the programs is given below. The detailed comments should allow the reader to understand, modify, and trouble-shoot the program.

A. The Send Program

Some important variables, their meanings, and their locations in zero page are given:

Name Location Use

TIME 0000 TIME is the quantity T mentioned in the section on timing considerations. It is the time, in units of 1.024 ms, of the dot or element space components.

SPEED 0013 SPEED is the hex equivalent of the number entered for the speed by the operator.

PNTR 0015 PNTR is a number which points to the location in the FIFO memory which contains the character currently being sent. The program idles as long as Y = PNTR, but begins to send when Y exceeds PNTR.

Name Location Use

LO 001E Scratchpad location for division of 494 by SPEED to give TIME.

HI 001F Same use as LO.

CNTR 0022 CNTR keeps track of how many characters are left in the FIFO memory. A character entered decrements CNTR; a character sent increments CNTR.

CHEK 0024 Scratchpad location to count the number of numbers which have been entered after the control S has been entered.

YREG 00F4 The Y register is used to point to the location in the FIFO memory where the last character entered from the keyboard is, namely 0200,Y.

B. The Receive Program

Some important variables, their meanings, and their locations are given:

Name Location Use

XREG 00F5 The X register is the character register. It begins with a 1 in the 0-bit. It is shifted left for each mark element received and loaded with a 1 for a dash and a zero for a dot. Later it is used to index a table in zero page which has the ASCII code for the character.

MCNTZ 0054 If a mark element (dot or dash) is being received (PB7 and IRQ at logical 0) the mark counter is incremented at a rate of 1 count every 2.048 ms.

SCNTZ 00EE Same as mark counter except the incrementing occurs when a space is being detected (PB7 high and IRQ high). Rate is also 1 count every 2.048 ms.

HALFT 0051 If the SPEED is set correctly, the number of counts during a dot should be exactly 1/2 TIME. This is the "dot length". If MCNTZ exceeds 1/2 the dot length the program decides that a valid mark character has been received. HALFT is 1/2 the dot length. A valid space element occurs when SCNTZ exceeds HALFT.

Name Location Use

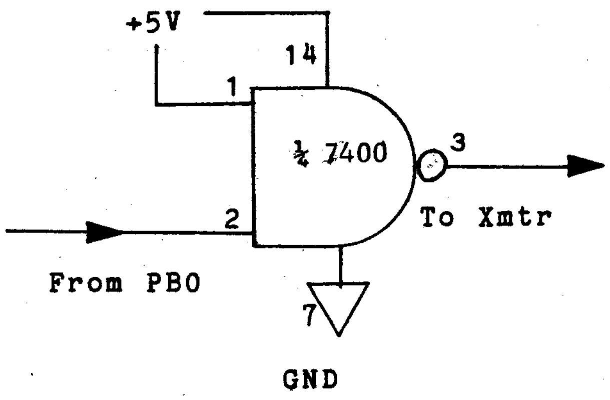
TWOT 0052 TWOT is twice the dot length and is used to decide if a dot or a dash has been received. If MCNTZ exceeds TWOT the element is a dash, otherwise it is a dot.

FIVET 0053 FIVET is five times the dot length and is used to decide when a word space has been received.

IV. INTERFACE

The keyboard strobe is connected to the NMI pin on the expansion connector on the KIM-1, and the 7 bit ASCII code from the keyboard goes to pins PA0-PA6, the low order bit to PA0 and the high order bit to PA6. PA7 should be pulled up with a 10K resistor.

The author's transmitter is a solid-state Triton IV and can be keyed with TTL IC's. The circuit diagram below indicates how it was connected to the KIM-1. Transmitters using grid-block keying or cathode keying cannot use this circuit. A relay driven by a Darlington pair connected to pin PB0 should work. The KIM-1 manuals give the appropriate details.



The audio from the receiver must produce a logical 0 at pin PB7 and the IRQ pin when a tone is detected, and a logical 1 at the same pins when a space is detected. The reader is urged to try either of the circuits found in references 1 and 2. I used a half-baked scheme in which the audio from the receiver was fed to a half-wave rectifier (diode), filtered slightly, and connected to the inverting input of a CA3140 op amp. The voltage at the non-inverting input was adjustable. The op amp was operated as an open-loop comparator with the output connected to pin PB7 and IRQ. An oscilloscope was necessary to monitor the output and make the necessary adjustments for various signal levels. I am not recommending this circuit for general use.

I have also tried using the tape-input PLL system on the KIM-1 to convert the receiver audio to a digital signal. To lower the free-running frequency of the VCO a shunt capacitor must be added. The digital signal appears at address 1742, bit 7. I had only marginal success, the problem being that the digital signal tends to drop out for very short periods of time, which clears the mark counter (instructions 039F-03A2). Substituting NOP's for these instructions seems to improve the performance, but receiver tuning and volume control adjustments are sensitive. Some users may wish to experiment with deleting the aforementioned instructions in whatever interface circuit they may use.

V. MISCELLANEOUS REMARKS

To get the entire Send/Receive program in the KIM-1 memory extensive use was made of page 1. This is also used as the stack. Care was taken to leave enough room for the stack operations, and for insurance, there are several points in the program where the stack pointer is initialized to FF. No problems should be encountered once the program is up and running. If you have any debugging to do I suggest using the single-step mode (be sure to set the NMI vectors) to check the jumps and branches. My experience has been that errors in branches generally result in about half the program being wiped out, especially if it is in page 1 of memory.

Wouldn't it be nice if some outfit like The COMPUTERIST would offer an interface board which would provide an audio to digital Morse circuit, a relay driver and relay (reed type) for transmit, a DIP socket for a ribbon cable from the keyboard, and a DIP socket for the ASCII out (see appendix), all on a single board which would mate with the KIM-1 application socket.

The first time I operated the system, I answered a CQ on 40 meters from WB2GMN, Hank, who has Army Signal Corps experience. Even though he rated his speed at 55 wpm he copied me at 60 wpm. Hank reported that the code sounded like perfect code (which it should be) and that it was very crisp at 60 wpm. It was a real coincidence to contact someone who had the capability to appreciate the keyboard system and to give an evaluation of its performance.

I hope that you enjoy working these programs. If you do not want the receive program, simply put in a JMP 0300 instruction (4C 00 03) starting at 0300. If you have any questions, feel free to write, enclosing a SASE for a response. I will try to answer any questions about interfacing the system to your station.

References:

[1] Steber, G. R., and Reyer, S. E., "The Morse-A-Letter", Popular Electronics, January, 1977.

[2] Riley, T. P., "A Morse Code to Alphanumeric Converter and Display", in three parts, QST for October, November and December, 1975.

[3] Pollock, James W., "1000 WPM Morse Code Typer", 73 Magazine, January, 1977.

[4] Pollock, James, W., "A Microprocessor Controlled CW Keyboard", Ham Radio, January, 1978.

[5] Ward, Jack, "Manipulating ASCII Data", Kilobaud, February, 1978.

ACSII to MORSE and MORSE to ASCII
Lookup Tables in Page Zero

00	XX	20	45	54	49	41	4E	4D	53	55	52	57	44	4B	47	4F
10	48	56	46	XX	4C	XX	50	4A	42	58	43	59	5A	51	XX	XX
20	35	34	XX	33	XX	XX	XX	32	XX	XX	XX	XX	XX	XX	XX	31
30	36	3D	2F	XX	XX	XX	XX	XX	37	XX	XX	XX	38	XX	39	30
40	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	3F	XX	XX	XX
50	XX	XX	XX	XX	XX	2E	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX
A0	80	XX	XX	2A	45	XX	XX	XX	XX	XX	XX	XX	73	XX	55	32
B0	3F	2F	27	23	21	20	30	38	3C	3E	XX	XX	XX	31	XX	4C
C0	XX	05	18	1A	0C	02	12	0E	10	04	17	0D	14	07	06	0F
D0	16	1D	0A	08	03	09	11	0B	19	1B	1C	XX	XX	XX	XX	XX

Special Morse Characters	Keyboard Character
$\overline{\text{BT}}$	=
$\overline{\text{SK}}$	$
$\overline{\text{AR}}$	#
Space (Word)	Space Bar

```
                    TIME   *     $0000         MORSE CODE SEND PROGRAM
                    ZTB    *     $0000
                    SPEED  *     $0013
                    PNTR   *     $0015
                    LO     *     $001E
                    HI     *     $001F
                    CNTR   *     $0022
                    CHEK   *     $0024
                    HALFT  *     $0051  1/2 DOT TIME
                    TWOT   *     $0052  TWICE DOT TIME
                    FIVET  *     $0053  FIVE TIME DOT TIME
                    MCNTZ  *     $0054
                    SCNTZ  *     $00EE
                    FIFO   *     $0200
                    CULO   *     $13F9  AUTHORS DISPLAY DEVICE
                    CUHI   *     $13FA  REGISTERS
                    DATA   *     $13FB
                    NMIL   *     $17FA  NON-MASKABLE INTERRUPT LOW
                    NMIH   *     $17FB  NON-MASKABLE INTERRUPT HIGH
                    IRLO   *     $17FE  INTERRUPT REQUEST LOW
                    IRHI   *     $17FF  INTERRUPT REQUEST HIGH
                    PAD    *     $1700  PORT A DATA
                    PADD   *     $1701  PORT A DATA DIRECTION
                    PBD    *     $1702  PORT B DATA REGISTER
                    PBDD   *     $1703  PORT B DATA DIRECTION REGISTER
                    SAD    *     $1740  KIM DISPLAY
                    SADD   *     $1741  KIM DISPLAY DIRECTION
                    SBD    *     $1742
                    SBDD   *     $1743
                    TIM    *     $1706  DIVIDE BY 64 TIMER
                    TMER   *     $1707  DIVIDE BY 1024 TIMER
                    TAB    *     $1FE7  KIM ROM CHARACTER TABLE

0056                       ORG   $0056

0056 D8             INIT   CLD          INIT SEQUENCE.  CLEAR DECIMAL
0057 A9 40                 LDAIM $40
0059 85 00                 STAZ  TIME   INITIAL CODE SPEED OF 18 WPM
005B 78             RTN    SEI          PREVENT INTERRUPTS
005C A2 FF                 LDXIM $FF    FROM RECEIVER
005E 9A                    TXS          SET STACK POINT TO TOP $01FF
005F A9 20                 LDAIM VCTL   SET NIM VECTORS FOR KEYBOARD
0061 8D FA 17              STA   NMIL
0064 A9 01                 LDAIM VCTL   /
0066 8D FB 17              STA   NMIH
0069 A9 00                 LDAIM $00
006B 8D 01 17              STA   PADD   PORT A IS INPUT PORT
006E 8D 02 17              STA   PBD    PORT B, PIN PB0, WILL BEGIN AT 0
0071 A9 01                 LDAIM $01    PORT B, PIN PB0, IS OUTPUT PIN
0073 8D 03 17              STA   PBDD
0076 A9 7F                 LDAIM $7F    SET UP DISPLAY PORTS
0078 8D 41 17              STA   SADD   PINS 0 - 6 ARE OUTPUT PINS
007B A9 1E                 LDAIM $1E
007D 8D 43 17              STA   SBDD   PINS 1 - 4 ARE OUTPUT PINS
0080 A9 08                 LDAIM $08    INIT LEFTMOST DIGIT
```

```
0082 8D 42 17          STA   SBD     ON KIM-1 DISPLAY
0085 A9 80             LDAIM $80     BLANK DISPLAY BY PUTTING 80
0087 8D 40 17          STA   SAD     IN PORT SAD
008A A0 FF             LDYIM $FF     INIT Y POINTER
008C 84 15             STYZ  PNTR    INIT SEND POINTER
008E 84 22             STYZ  CNTR    INIT BUFFER COUNTER
0090 C4 15    LOOP     CPYZ  PNTR    IS Y = PNTR?
0092 F0 FC             BEQ   LOOP    YES, IDLE UNTIL DIFFERENT
0094 E6 15             INCZ  PNTR    NO, INCR PNTR TO LOOKUP
0096 A6 15             LDXZ  PNTR    CHARACTER. PNTR = X INDEX
0098 BD 00 02          LDAX  FIFO    GET CHARACTER FROM FIFO
009B 4C 15 01          JMP   LOOPX   CONTINUE AT LOOPX

              DISPLAY SUBROUTINE

0100                   ORG   $0100

0100 A6 22    DISP     LDXZ  CNTR    TRANSFER CNTR TO X
0102 E0 10             CPXIM $10     IS CNTR LESS THAN 10 HEX
0104 90 08             BCC   OVER    YES, DISPLAY CNTR
0106 A9 80             LDAIM $80     NO, BLANK DISPLAY
0108 8D 40 17          STA   SAD
010B 4C 14 01          JMP   THER
010E BD E7 1F OVER     LDAX  TAB     FIND VALUE FROM KIM ROM
0111 8D 40 17          STA   SAD     TO DISPLAY CNTR
0114 60       THER     RTS           RETURN

0115 20 80 17 LOOPX    JSR   SEND    GO TO SEND TO OUTPUT CODE
0118 E6 22             INCZ  CNTR    INCR CNTR
011A 20 00 01          JSR   DISP    DISPLAY IF LESS THAN 10
011D 4C 90 00          JMP   LOOP    CONTINUE LOOP

              INTERRUPT ROUTINES

0120 48       VCTL     PHA           SAVE A, X AND STATUS
0121 8A                TXA           ON STACK
0122 48                PHA
0123 08                PHP
0124 AD 00 17          LDA   PAD     READ KEYBOARD
0127 48                PHA           SAVE ON STACK
0128 29 60             ANDIM $60     MASK ALL BUT TOP BITS
012A F0 0F             BEQ   CNTRL   CONTROL CHARACTER?
012C 68                PLA           NO. RECALL A AND INCR Y
012D C8                INY
012E 99 00 02          STAY  FIFO    STORE A CHAR IN FIFO
0131 20 00 01          JSR   DISP    DISPLAY CNTR IF LESS THAN 10
0134 C6 22             DECZ  CNTR    UPDATE CNTR
0136 28       BACK     PLP           RESTORE REGISTER
0137 68                PLA
0138 AA                TAX
0139 68                PLA
013A 40                RTI           RETURN FROM INTERRUPT

013B 68       CNTRL    PLA           RECALL A FROM STACK
013C 29 7F             ANDIM $7F     MAKS OFF HIGHEST BIT
013E C9 02             CMPIM $02     BACKSPACE?
```

```
0140 D0 06            BNE   CNTX   TEST OTHER CHARACTER
0142 88               DEY          YES.  DECR Y TO DELETE CHARACTER
0143 E6 22            INCZ  CNTR   FIX COUNTER
0145 4C 36 01         JMP   BACK   RETURN

0148 C9 13      CNTX  CMPIM $13    CONTROL S = SPEED
014A D0 58            BNE   ARND   NO TEST OTHERS
014C A9 58            LDAIM FIX    CHANGE INTERRUPT SO NEXT
014E 8D FA 17         STA   NMIL   INTERRUPTS GO TO FIX
0151 A9 00            LDAIM $00    INIT CHEK TO 00
0153 85 24            STAZ  CHEK
0155 4C 36 01         JMP   BACK   RETURN

0158 48         FIX   PHA          SAVE REGISTERS
0159 8A               TXA
015A 48               PHA
015B 08               PHP
015C AD 00 17         LDA   PAD    READ FIRST DIGIT
015F 29 0F            ANDIM $0F    MASK TO DIGIT
0161 AA               TAX          MOVE TO X
0162 A5 24            LDAZ  CHEK   CHEK = 0 = FIRST DIGIT
0164 C9 01            CMPIM $01    CHEK = 1 = SECOND DIGIT
0166 F0 10            BEQ   AHD    FIRST DIGIT BRANCH
0168 8A               TXA          GET DIGIT BACK
0169 0A               ASLA         TIMES 2
016A 85 13            STAZ  SPEED  SAVE
016C 0A               ASLA         TIMES 4
016D 0A               ASLA         TIMES 8
016E 18               CLC          PREPARE TO ADD SPEED
016F 65 13            ADCZ  SPEED  *8 + *2 = *10
0171 85 13            STAZ  SPEED  STORE
0173 E6 24            INCZ  CHEK   SET FOR SECOND DIGIT
0175 4C 36 01         JMP   BACK   RETURN

0178 C6 24      AHD   DECZ  CHEK   RE-INIT CHEK
017A 8A               TXA
017B 18               CLC
017C 65 13            ADCZ  SPEED  ADD ONES DIGIT TO
017E 85 13            STAZ  SPEED  TENS DIGIT ANS STORE
0180 38               SEC          DIVIDE 494(HEX)/SPEED
0181 A2 00            LDXIM $00    CLEAR X FOR QUOTIENT
0183 A9 94            LDAIM $94    LOW ORDER BYTE OF DIVIDEND
0185 85 1E            STAZ  LO
0187 A9 04            LDAIM $04    HIGH ORDER BYTE OF DIVIDEND
0189 85 1F            STAZ  HI
018B A5 1E      UP    LDAZ  LO     START SUB. FROM DIVIDEND
018D E5 13            SBCZ  SPEED  UNTIL BORROW
018F 85 1E            STAZ  LO     FROM HIG BYTE, IE CARRY IS SET
0191 A5 1F            LDAZ  HI     IF BORROW OCCURS FROM LOW ORDER
0193 E9 00            SBCIM $00    BYTE, SUB 1 FROM HIGH
0195 85 1F            STAZ  HI     ORDER BYTE
0197 E8               INX          INCR X FOR EACH SUB.
0198 B0 F1            BCS   UP     BORROW FROM HI? NO. GO BACK
019A 86 00            STXZ  TIME   AND SUB. OTHERWISE DONE
019C A9 20            LDAIM VCTL   RESET NMI VECTORS FOR VCTL
019E 8D FA 17         STA   NMIL
```

```
01A1 4C 36 01          JMP   BACK   RETURN TO MAIN PROGRAM

01A4 C9 12      ARND   CMPIM $12    REMAINDER OF VCTL
01A6 D0 03             BNE   TREE   CONTROL R?
01A8 4C 00 03          JMP   RCV    YES. GO TO RECEIVE PROGRAM
01AB C9 0D      TREE   CMPIM $0D    CARRAIGE RETURN?
01AD D0 03             BNE   BUF    BRANCH IF NOT
01AF 4C 5B 00          JMP   RTN    YES. START MAIN PROGRAM
01B2 C9 07      BUF    CMPIM $07    CONTROL G?
01B4 F0 03             BEQ   BRR    YES. RESET STACK POINTER AND GO
01B6 4C B6 01   IDLE   JMP   IDLE   TO LOOP. OR, IDLE HERE
01B9 A2 FF      BRR    LDXIM $FF    WHILE BUFFER IS LOADED
01BB 9A                TXS          RESET STACK TOP
01BC 4C 90 00          JMP   LOOP   AND CONTINUE


                MORSE CODE RECEIVE PROGRAM

                       ORG   $0300

0300 A9 90      RCV    LDAIM IRQ    SET IRQ VECTORS
0302 8D FE 17          STA   IRLO
0305 A9 03             LDAIM IRQ    / PAGE ADDRESS
0307 8D FF 17          STA   IRHI
030A A5 00      CRK    LDAZ  TIME   SET DOT LENGTH BY GETTING
030C 4A                LSRA         TIME AND DIVIDING BY 2
030D 85 51             STAZ  HALFT
030F 46 51             LSRZ  HALFT  HALFT HALFT IS 1/2 DOT LENGTH
0311 85 52             STAZ  TWOT
0313 06 52             ASLZ  TWOT   TWOT IS TWICE DOT LENGTH
0315 85 53             STAZ  FIVET
0317 0A                ASLA         MULTIPLY BY 4
0318 0A                ASLA
0319 18                CLC
031A 65 53             ADCZ  FIVET  AND ADD 1 TIMES TO GET
031C 85 53             STAZ  FIVET  5 TIMES DOT LENGTH
031E A9 00             LDAIM $00    CLEAR MARK AND SPACE
0320 85 54             STAZ  MCNTZ  COUNTERS
0322 85 EE             STAZ  SCNTZ
0324 58                CLI          ALLOW INTERRUPTS TO START
0325 A2 01             LDXIM $01    INIT CHARACTER REGISTER
0327 4C 27 03   IDL    JMP   IDL    IDLE HER UNTIL MARK OCCURS
032A 20 8A 03   AGN    JSR   TIMSET START TIMER FOR SPACE COUNT
032D E6 EE             INCZ  SCNTZ  INCR SPACE COUNTER
032F A5 EE             LDAZ  SCNTZ  DOES IT EXCEED 1/2 DOT LENGTH?
0331 C5 51             CMPZ  HALFT
0333 B0 08             BCS   CHECK  YES, JUMP TO SET CHAR REGS
0335 AD 07 17   WAIT   LDA   TMER   OTHERWISE WAIT FOR TIMER
0338 10 FB             BPL   WAIT
033A 4C 2A 03          JMP   AGN    AND COUNT SPACES

033D 8A         CHECK  TXA          SHIFT CHAR REGISTER LEFT
033E 0A                ASLA
033F AA                TAX
```

```
0340 A5 54          LDAZ  MCNTZ  IF MARK COUNTER EXCEEDS TWICE
0342 C5 52          CMPZ  TWOT   THE DOT LENGTH, PUT ONE IN
0344 90 03          BCC   SKIP   CHAR REGISTER, OTHERWISE A ZERO
0346 E8             INX
0347 B0 11          BCS   FAT    IF A DASH, SKIP DISPLAY
0349 0A      SKIP   ASLA         IF A DOT, COMPARE WITH TIME
034A C5 00          CMPZ  TIME   FOR SPEED INDICATOR
034C B0 07          BCS   CAT
034E A9 F1          LDAIM $F1    SHOW "F" IS DISPLAY
0350 8D 40 17       STA   SAD
0353 90 05          BCC   FAT
0355 A9 ED   CAT    LDAIM $ED    SHOW "S" IN DISPLAY
0357 8D 40 17       STA   SAD
035A A9 00   FAT    LDAIM $00    CLEAR MARK COUNTER
035C 85 54          STAZ  MCNTZ
035E AD 07 17 HOLD  LDA   TMER   WAIT FOR TIMER
0361 10 FB          BPL   HOLD
0363 20 8A 03       JSR   TIMSET START TIMER AGAIN
0366 E6 EE          INCZ  SCNTZ  INCR SPACE COUNTER AGAIN
0368 A5 EE          LDAZ  SCNTZ
036A C5 52          CMPZ  TWOT   DOES SPACE COUNTER EXCEED TWICE
036C 90 F0          BCC   HOLD   THE DOT LENGTH. IF NOT, HOLD
036E 20 CA 03       JSR   CHAR   IF YES, PRINT CHARACTER
0371 A2 01          LDXIM $01    RESET CHAR REGISTER
0373 AD 07 17 DOZE  LDA   TMER   WAIT FOR TIMER
0376 10 FB          BPL   DOZE
0378 20 8A 03       JSR   TIMSET START TIMER AGAIN
037B E6 EE          INCZ  SCNTZ  INCR SPACE COUNTER
037D A5 EE          LDAZ  SCNTZ
037F C5 53          CMPZ  FIVET  DOES SPACE COUNTER EXCEED FIVE TIMES
0381 90 F0          BCC   DOZE   DOT LENGTH.  IF LESS, DOZE AGAIN
0383 20 CA 03       JSR   CHAR   OTHERWISE PRINT SPACE
0386 78             SEI          PREVENT INTERRUPTS WHILE
0387 4C 0A 03       JMP   CRK    CHECKING SPEED SETTING

038A A9 20   TIMSET LDAIM $20    LOAD TIMER FOR 2.048 MS
038C 8D 06 17       STA   TIM
038F 60             RTS          RETURN TO RCV PROGRAM

0390 08      IRQ    PHP          SAVE REGISTERS
0391 48             PHA
0392 20 8A 03       JSR   TIMSET START TIMER

0395 AD 07 17 LOAF  LDA   TMER   WAIT FOR TIMER
0398 10 FB          BPL   LOAF
039A AD 02 17       LDA   PBD    IS MARK SIGNAL PRESENT
039D 10 09          BPL   OVER   YES, GO TO OVER
039F A9 00          LDAIM $00    NO, MUST HAVE BEEN NOISE
03A1 85 54          STAZ  MCNTZ  WHICH CAUSED INTERRUPT.  RETURN
03A3 E6 EE          INCZ  SCNTZ  TO COUNT SPACE AFTER RESETTING
03A5 68             PLA          MARK COUNTER TO ZERO
03A6 28             PLP
03A7 40             RTI          RETURN FROM INTERRUPT
```

```
03A8 20 8A 03  OVER   JSR    TIMSET START TIMER AGAIN
03AB E6 54            INCZ   MCNTZ  INCR MARK COUNTER
03AD A5 54            LDAZ   MCNTZ  DOES MARK COUNTER EXCEED
03AF C5 51            CMPZ   HALFT  1/2 THE DOT LENGTH?
03B1 90 E2            BCC    LOAF   NO, GO LOAF AND CHECK MARK
03B3 A9 00            LDAIM  $00    YES.  CLEAR SPACE COUNTER
03B5 85 EE            STAZ   SCNTZ
03B7 AD 07 17  KILTIM LDA    TMER   CHECK TIMER
03BA 10 FB            BPL    KILTIM KILL TIME
03BC AD 02 17         LDA    PBD    CHECK MARK SIGNAL ON PB7
03BF 10 E7            BPL    OVER   LOOP AGAIN IF STILL ON
03C1 8A               TXA           SAVE S WHILE STACK POINTER IS SET
03C2 A2 FF            LDXIM  $FF    RESET TO TOP OF STACK
03C4 9A               TXS
03C5 AA               TAX           RESTORE X
03C6 58               CLI           CLEAR INTERRUPT FLAG SET EARLIER
03C7 4C 2A 03         JMP    AGN    RETURN TO COUNT SPACE

03CA B5 00     CHAR   LDAZX  ZTB    LOOKUP ASCII SYMBOL
03CC 8D FB 13         STA    DATA   DATA IS VIDEO PORT IN AUTHORS
03CF A9 3F            LDAIM  $3F    SYSTEM.  THE REMAINDER OF THIS
03D1 2D F9 13         AND    CULO   SUBROUTINE INCREMENTS THE
03D4 C9 3F            CMPIM  $3F    POSITION OF THE CURSOR TO PREPARE
03D6 90 11            BCC    AHD    FOR THE NEXT CHARACTER
03D8 A9 1F            LDAIM  $1F
03DA 2D FA 13         AND    CUHI
03DD 18               CLC
03DE 69 01            ADCIM  $01
03E0 C9 20            CMPIM  $20
03E2 90 02            BCC    UP
03E4 A9 10            LDAIM  $10
03E6 8D FA 13  UP     STA    CUHI
03E9 EE F9 13  AHD    INC    CULO
03EC 60               RTS

               SEND SUBROUTINE

1780                  ORG    $1780

1780 AA        SEND   TAX           A CONTAINS CHAR FROM FIFO
1781 B5 00            LDAZX  ZTB    USE THIS TO LOOKUP MORSE
1783 30 3F            BMI    WDSP   SPACE BAR CHAR HAS 1 IN BIT 7
1785 18               CLC           IF NOT MINUS, CLEAR CARRY FLAG AND
1786 A2 08            LDXIM  $08    SET UP X FOR 8 ROL INSTRUCTIONS
1788 2A        RPT    ROLA          ROTATE LEFT UNTIL 1 APPEARS IN CARRY
1789 B0 06            BCS    DWN    BRANCH IF 1 IN CARRY
178B CA               DEX           ELSE, DECREMENT X
178C F0 35            BEQ    OUT    IF X = 0, THEN DONE
178E 4C 88 17         JMP    RPT    ELSE CONTINUE
1791 CA        DWN    DEX           KEEP TRACK OF BITS TESTED
1792 2A        BACK   ROLA          ROTATE A LEFT AND SAVE ON STACK
1793 48               PHA
1794 8A               TXA           SAVE X ON STACK ALSO
1795 48               PHA
```

```
1796 B0 18              BCS    DASH   DID ROTATE SET CARRY? IF YES,
1798 A2 01              LDXIM $01     SEND DASH, ELSE SEND DOT
179A EE 02 17  DAH      INC    PBD    PB0 WILL BE LOGICAL 1 FO 1 T
179D 20 C9 17  SPA      JSR    TIMER  TIME GIVES DELAY OF TIME (1.024MS)
17A0 CA                 DEX           ONE TIME UNIT IS UP
17A1 D0 FA              BNE    SPA    IS X = 0? DELAY ANOTHER UNIT
17A3 AD 02 17           LDA    PBD    YES. NOW CHECK PB0.  IF A 1
17A6 4A                 LSRA          A SHIFT WILL SET CARRY FLAG
17A7 90 0C              BCC    DONE   IF CARRY CLEAR, THEN DONE
17A9 CE 02 17           DEC    PBD    OTHERWISE, SET PB0 = 0 FOR ELEMENT
17AC E8                 INX           SPACE FOR A DELAY OF 1 UNIT BY
17AD 4C 9D 17           JMP    SPA    RESETTING X AND LOADING TIMER

17B0 A2 03     DASH     LDXIM $03     DASH TAKES 3 TIME UNITS
17B2 4C 9A 17           JMP    DAH    SEND 3 UNITS FOLLOWED BY SPACE

17B5 68        DONE     PLA           THEN ELEMENT IS DONE SO
17B6 AA                 TAX           RESTORE A AND X AND GO BACK
17B7 68                 PLA           IF X IS NOT ZERO
17B8 CA                 DEX           OTHERWISE ADD CHARACTER SPACE
17B9 D0 D7              BNE    BACK   BY RUNNING TIMER FOR
17BB A2 02              LDXIM $02     2 MORE TIME UNITS
17BD 20 C9 17  AGAIN    JSR    TIMER
17C0 CA                 DEX
17C1 D0 FA              BNE    AGAIN  IF X = 0, THEN DONE
17C3 60        OUT      RTS           OR ELSE DELAY MORE

17C4 A2 04     WDSP     LDXIM $04     WORDSPACE REQUIRES 4 MORE TIME UNITS
17C6 4C BD 17           JMP    AGAIN  SO USE TIMER FOR THIS

17C9 A5 00     TIMER    LDAZ   TIME   GET TIME FROM ZERO PAGE
17CB 8D 07 17           STA    TMER   LOAD DIVIDE BY 1024 TIMER
17CE 2C 07 17  CHK      BIT    TMER   IS TIMER FINISHED?
17D1 10 FB              BPL    CHK    NO, WAIT FOR IT
17D3 60                 RTS           YES, RETURN
```

APPENDIX:
Using the KIM-1 Ports to Output the ASCII

Most readers will not have the same addressable video system used by the author. To use the receive portion of the program, some provision must be made to output the ASCII along with a strobe pulse. Below you will find a suggested program to do this. It makes use of ports SAD and SBD addresses 1740 and 1742 respectively. These are available on the application connector. The ASCII code appears at the KB COL A-G pins, while the strobe should appear at the TTY PTR pin.

NOTE: While this program should work it has not been tested.

ALTERNATIVE ASCII OUTPUT

```
                         ORG   $03CA

                  *** THIS ROUTINE HAS NOT BEEN TESTED ***

03CA               ZTB    *     $0000
03CA               SAD    *     $1740
03CA               SADD   *     $1741
03CA               SBD    *     $1742
03CA               SBDD   *     $1743

03CA A9 20         CHAR   LDAIM $20     ENABLE OUTPUT PULSE PINS
03CC 8D 42 17             STA   SBD
03CF A9 21                LDAIM $21
03D1 8D 43 17             STA   SBDD
03D4 AD 40 17             LDA   SAD     SAVE CONTENTS OF CURRENT
03D7 48                   PHA           DISPLAY ON KIM-1
03D8 AD 41 17             LDA   SADD
03DB 48                   PHA
03DC B5 00                LDAZX ZTB     GET ASCII CODE
03DE 8D 40 17             STA   SAD     OUTPUT ASCII
03E1 A9 FF                LDAIM $FF
03E3 8D 41 17             STA   SADD    ENABLE OUTPUT PORT
03E6 EE 42 17             INC   SBD     STROBE PULSE WILL BE
03E9 EA                   NOP           LENGTHEN PULSE
03EA CE 42 17             DEC   SBD     NEGATIVE
03ED 68                   PLA           RESTORE SADD AND SAD
03EE 8D 41 17             STA   SADD
03F1 68                   PLA
03F2 8D 40 17             STA   SAD
03F5 A9 1E                LDAIM $1E     RESTORE SBDD AND SBD
03F7 8D 43 17             STA   SBDD
03FA A9 08                LDAIM $08
03FC 8D 42 17             STA   SBD
03FF 60                   RTS
```

PET

Commodore Business Machines, Inc.
901 California Avenue
Palo Alto, CA 94304
415-326-4000

* a perforated "tear-out" reference card

THE PET'S IEEE-488 BUS: BLESSING OR CURSE?

Charles Floto
267 Willow Street
New Haven, CT 06511

IEEE-488 (usually pronounced I-triple-E four-eighty-eight) is the number of a standard for information exchange adopted by the Institute of Electrical and Electronics Engineers. Given that a major complaint of microcomputer users has been that the lack of industry standards prevents the exchange of information, the reaction when it was announced that Commodore's PET 2001 would support the IEEE bus should perhaps not be surprising.

However a few people have been surprised by this 488 mania. Pickles & Trout accompanied announcement of an I/O board for the S-100 bus with with the offhand remark that they planned to produced a 488 adapter for it. When they found that enthusiasm for this incidental feature overwhelmed interest in the basic board they decided to develop an I/O card exclusively to support the IEEE-488 bus. It is expected to retail in the $200 range. Which makes the fact that Commodore is including a similar interface in the $800 PET (8KRAM version) all the more wonderful.

Just how easy will it be for a PET owner to design a system around the IEEE-488 bus? It can be compared to solving the following problem; You are to design a computer with provision for more than one CPU card. Its bus shall be limited to 16 signal lines, with several ground lines but no power lines. You are to build a separate power supply for each card in the system and, since it is to be spread all over your home or office, a separate case as well.

The difference between this problem and using the IEEE-488 bus is that in the latter case the design of the bus has been done for you and to use it you must be prepared to abide by certain specified and rather complex conventions. In short, you shouldn't even attempt to design a peripheral interface to the PET's 488 I/O bus unless you feel capable of designing internal circuit cards for other computers. Even then you may have problems if all your experience has been with a bus each of whose lines has a fixed purpose, rather than some being shared between data and either address or control functions.

If the IEEE-488 bus presents such difficulty in designing peripherals, why would Commodore want to use it? The first thing to realize is that design represents a fixed cost, the same whether you build one unit or 100,000. While design cost per unit is absurdly exorbitant for the individual making a single 488-compatible component, it becomes trivial for the mass producer.

For a second consideration suppose you were putting together your own system and Pickles & Trout offered you a circuit card to link your computer to the IEEE bus for $200. That's a lot to pay for one I/O port, but it's a bargain if it's the only one you'll ever have to buy. Thus the IEEE-488 format makes the PET less expensive than including an impressive number of serial and parallel ports.

Third, why expect PET to make things easy for individual hardware designers when that isn't the market it's aimed at?

At this point perhaps it's worth noting that the PET is only claimed to be electrically and logically compatible with the IEEE-488 bus--physical compatibiltiy is lacking as signals come out on printed circuit fingers rather than the standard connector. Standard interconnection cable consists of 16 signal lines, seven grounds, and a shield; it has male and female connections at each end. The corporate purchaser of a large system might pay as much for a single cable as the hobbyist pays for a circuit card.

We can't really judge the value of the PET's IEEE-488 bus until we see what becomes available to connect to it, and at what price. For now we may conclude that it presents a problem to those who want to design their own peripherals, but the potential for a competitive market in sophisticated mass-produced peripherals which will "plug in and go" in a wide variety of systems. And those who already own IEEE-488 products will be able to add the PET's computer power at an unprecedented price.

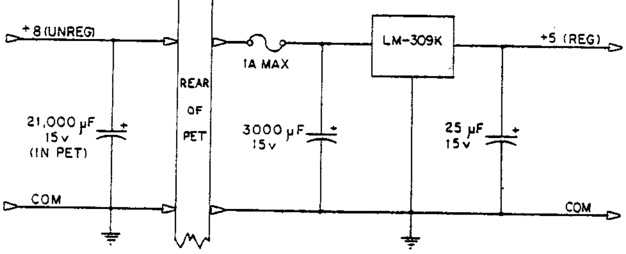
POWER FROM THE PET

Karl E. Quosig
2038 Hartnell Street
Union City, CA 94587

It is by now well known that the PET has no source of power for use outside of itself. The only source available is at the second Cassette Interface. This +5 VDC line will not source very much current; in fact, it will not even run a second cassette recorder. Also, all the +5 VDC regulators inside the PET are already running quite warm. If you want to experiment with the PET, say with the Parallel User Port (Mos Technology 6522 VIA), then where do you get the power without a complicated power supply interface? The answer is simple. I found the following inside the PET. One, the bridge rectifier is good for 3 Amperes. Two, the PET draws 1.5 Amperes worst case load. Conclusion: it should be possible to get 1 Ampere out of the PET without straining a thing.

To do this, all we need to do is run a line from the + (positive) side of the PET's filter capacitor and make it available at the rear of the PET (I put a test lead jack between the Parallel and IEEE Ports). This is +8 VDC Unregulated and by attaching a 3-point Regulator (see diagram below), say at our project board, we have plenty of power for all sorts of home projects. As an example, I brought all of the Parallel User Port pinouts down a 24" ribbon cable along with the +8 VDC line to a chassis which has the +5 VDC regulator and other circuitry, and terminated this on a homebrew mother board comprised of 22-pin edgecard connectors. I can now experiment with things such as noise makers, joysticks, etc. and have plenty of power for them.

I believe this should be of great benefit for those of you who like to mess around with the hardware. **Warning #1:** If you are going to drill a hole in the PET as I did, disconnect all connectors (very, very gently) to the PET's Main Board and remove it before going to work. Clean inside thoroughly before re-installation. **Warning #2:** In your projects, do not connect inductive loads directly to any output of the PET. Inductive loads must be fully buffered.

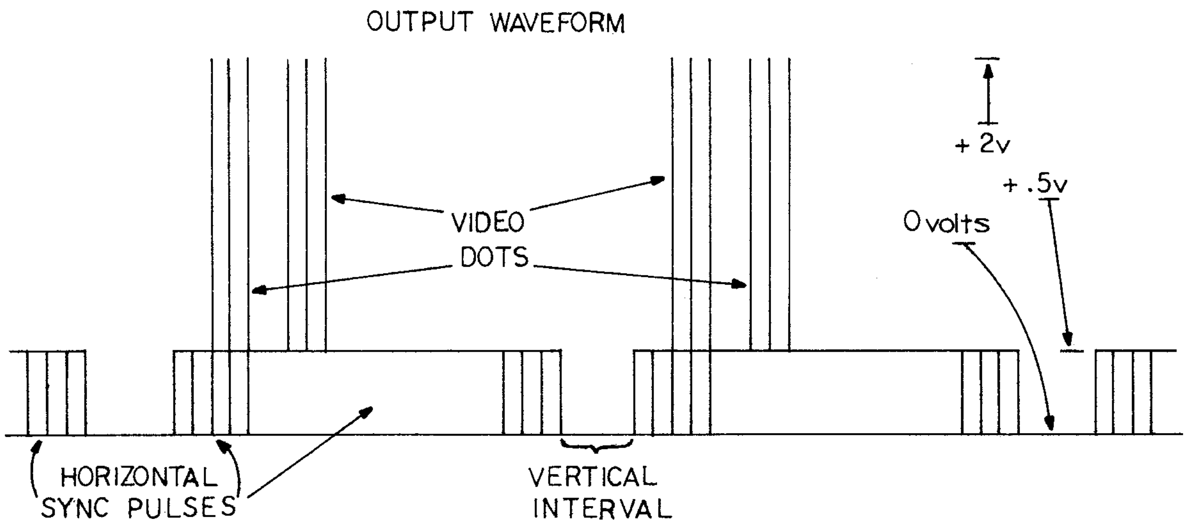


PET COMPOSITE VIDEO OUTPUT

Cal E. Merritt
R. 1, 4 Richfield Lane
Danville, IN 46122

I used one of the existing PET 5 volt sources. The easiest way to steal the video and drives is to carefully scrape clean the foils next to the monitor plug and tack solder a twisted pair to each signal and to the closest ground buss. Other variations would work equally well.

To avoid metal shavings and such falling on the main board, I removed the back cover from the monitor (Power OFF) and mounted a BNC jack two inches to the right of the brightness control



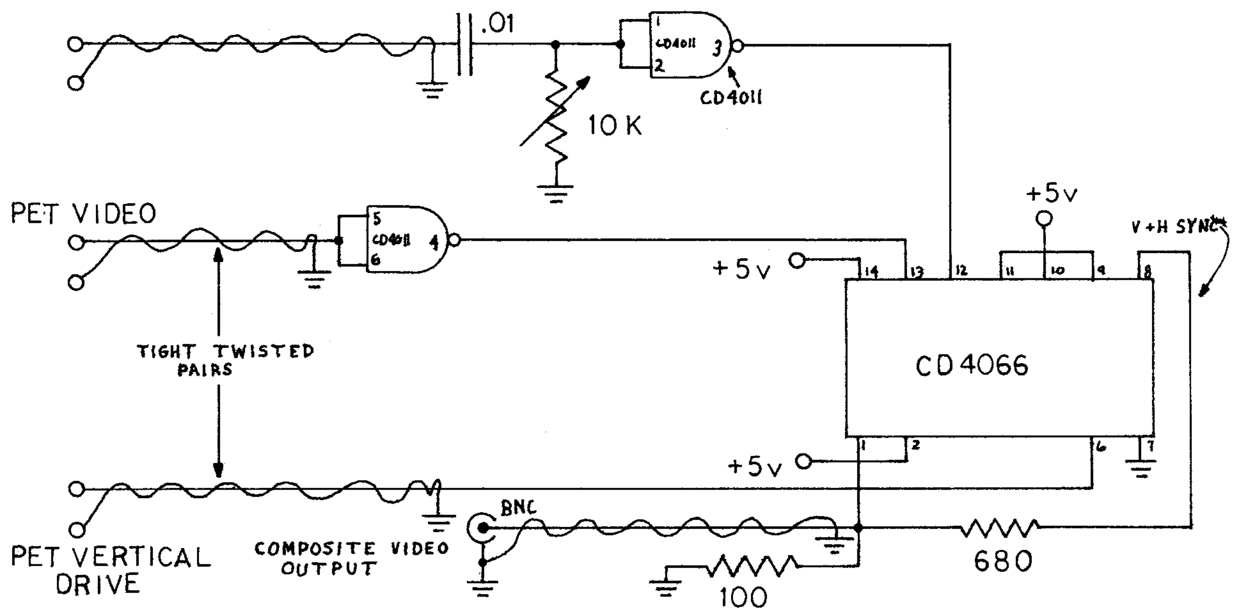
and fed it with a twisted pair. I mounted the board under one of the bolts that hold the monitor to the main chassis and attached the drive twisted pairs to the existing ones for the monitor.

This circuit provides composite video output from the PET. I have used the output to drive two different video monitors with good success.

The circuit is very simple and can be put together with a wire wrap tool in a few minutes.

Video monitors seem very tolerant and the two units I have used work fine. The only problem encountered was in attempting to do all white screen or very dense graphics which caused sync tear in one of the monitors. Normal or dense listings worked well.

All three monitors I tried worked with this video output. The appearance of the video will be a function of the quality of the monitor. Some of the scrapped out commercial units available with the 10MHz and more bandwidths look excellent with the PET video. I have had a number of people comment that my 12" commercial monitor looks better than the built-in unit. The add-on does not alter the existing PET display in any way.



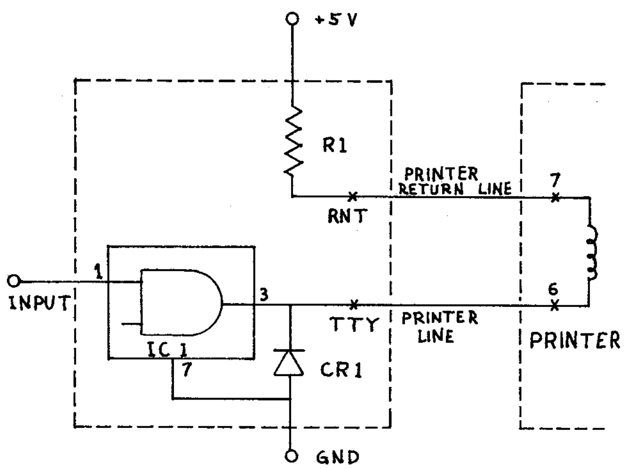
DESIGN OF A PET/TTY INTERFACE

Charles R. Husbands
24 Blackhorse Drive
Acton, MA 01720

With the recent acquisition of a PET Computer one of the facilities that was immediately needed was a method of obtaining hard copy listings of programs under development. In addition to the PET, I had an ASR 33 Teletype Unit available which had been interfaced to my KIM-1. This article describes the hardware interface and associated software necessary to use the ASR 33 TTY as a printing facility for the PET. An important design goal for the interface was to develop the software to remain resident in the computer in such a manner that the program under development could be loaded, run and listed without disturbing the listing program.

The Interface Circuit

Figure 1 shows the 20 ma current loop circuit required to interface the ASR 33 to the PET. The circuit consists of an open collector NAND gate to provide the proper buffering, a diode and a pull up resistor. The completed circuit was built on a small perforated board. The PET supplies power and ground to the interface board from the second Cassette Interface. The input signal is delivered from PA0 on the PET parallel user port. The interface board is connected to the teletype by means of the PRINTER and PRINTER RETURN lines. These lines attach to terminals 6 and 7 respectively on the ASR 33.



Parts List

IC1	7438	Quad 2 Input NAND Open Collector
CR1	1N4001	1A 50V Diode
R1	150 ohm	1/2 Watt Resistor

Figure 1.

A fairly simple circuit for buffering the control signal from the PET Computer and converting that signal to a current level capable of driving the printer mechanism on an ASR 33 TTY Unit.

Program Design

In order to allow the listing program to remain resident in the machine to list other programs under development, the program was written in machine language to be stored in Tape Buffer 2. Figure 2 shows a simple memory map of the PET random access memory allocations. Without a second tape cassette unit, a memory buffer of 198 bytes is available. When another program is loaded from tape or the NEW instruction is executed the operating system zeros out memory locations 1024 and above. However, it leaves the memory locations below 1024 undisturbed. To execute a machine language program the USR instruction must be called. The USR command uses a pair of memory location pointers stored in memory locations 1 and 2 to extablish the first location in machine language code to be processed. Locations 1 and 2 are not modified by the loading of a program from tape or the execution of the NEW instruction.

8192 .... $1200
Program Storage
1024 .... $0500
Tape Buffer 2
826 .... $033A
Tape Buffer 1
634 .... $027A
BASIC and Operating System Working Space
2 .... $0002
USR Control Pointers
0 .... $0000

Figure 2.

A Map of the PET Random Access Memory Space. The Listing Program resides in machine language in Tape Buffer 2.

A flow diagram of the Listing Algorithm is shown in Figure 3. The program after proper initiation examines the first character of the third line in the display for a value corresponding to the letter "R". It is the letter R appearing in the first display column which is used by the Listing Program to exit the listing algorithm and return control of the program to the calling routine. The R in the first column would normally correspond to the READY displayed by the computer at the end of a requested listing block or at the completion of an executed RUN. If the character in the first column is anything but an R the program executes a carriage return and then a line feed. The program examines the next displayed character and translates it from display format to ASCII format. The subroutine PRINT is then called.

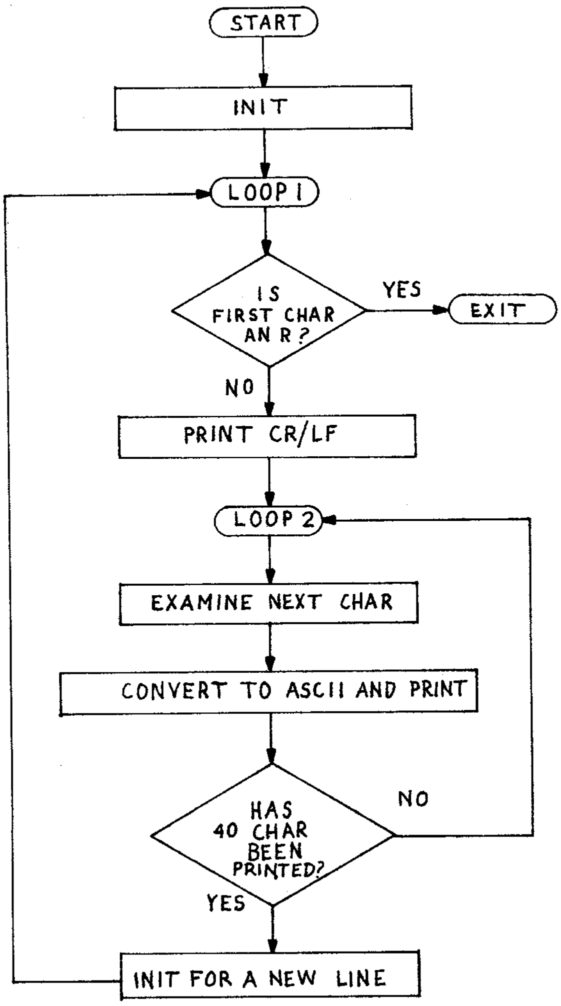


Figure 3.

A general listing algorithm for use with the TTY Listing Program. The software control of the output port is done in the PRINT subroutine.

The subroutine PRINT* is a machine language program which times out the proper serial bit pattern to the TTY to execute the printing of the designated letter. After each character is printed a counter is incremented and tested to determine if the 40 character line has been completed. If 40 characters have not been printed the next display character is examined. At the end of each line the first character of the next line is examined for an R before a carriage return and line feed is executed.

A listing of the program in BASIC format is shown in Listing 1. The program was originally hand assembled in 6502 machine language. The machine language program was then converted from hexadecimal to decimal and formatted as a series of POKE instructions. The machine language memory address pointers were also POKED into locations 1 and 2 by the BASIC program. The printout appearing in Listing 1 was produced on the authors TTY using the Listing Program.

Using the Listing Program

The program as shown in Listing 1 is loaded into the machine in the normal manner. A RUN command is then executed and the program will be POKED in machine format into Tape Buffer 2. The BASIC program to be listed is then loaded into the machine. The LIST-N instruction is then executed to allow the operator to preview the initial lines of code. When the operator is satisfied with the 15 to 18 lines of code to be printed, as displayed on the screen, the command X=USR(R) is entered and the RETURN key is depressed. The USR instruction transfers control to the machine language code located at the address specified by memory locations 1 and 2.

The teletype printer will then print the display on the PET CRT from the beginning of display line 3 to the word READY. The operator then uses the LIST M-X command to preview the next series of lines to be printed. It should be noted that the PET listing format leaves a blank line between the last line number selected and the READY response if the last line requested is not the last line in the program. The preview function allows the operator to block out the lines to be printed regardless of the line numbering technique employed when the program was composed. If the program being listed has an R in column 1 due to a line length in excess of 40 characters, the operator must take some action to remove this condition before executing the listing of that portion of the program.

Conclusions and Recommendations

The hardware and software illustrated in this article can be used to permit the listing of programs and recording the results of program runs on a conventional TTY unit. In using the program to print the results of computer runs it should be noted that the results should be formatted to begin on the third line of the display. An improved version of this program could be designed to look ahead when an R was discovered to extablish if an RE or REA string was present. As only 3 bytes were not used in Tape Buffer 2 in writing this program, that feature could not be included. Additional space could be freed if the program was redesigned to use the parallel to serial conversion facility available with the 6522 VIA output port. Using this facility the 90 bytes required to do the conversion from parallel to serial and timing out this information could be greatly reduced.

Listing 1.

A listing of the PET Listing Program as printed on the author's TTY unit. The program was hand assembled in 6502 language then converted to decimal format and entered as a series of BASIC "POKE" instructions. When executed the program will reside in Tape Buffer 2 in machine code format.

* The PRINT subroutine is a modified version of the "PRINT 1 CHAR" program developed by MOS Technology for the KIM-1.

```
1 REM***TELETYPE LISTING ROUTINE******
2 REM    CHARLES R. HUSBANDS
3 REM
4 REM THIS PROGRAM LISTS THE DATA
5 REM APPEARING ON THE SCREEN IN
6 REM SERIAL TELETYPE FORMAT. THE
7 REM PROGRAM IS STORED IN MACHINE
8 REM CODE IN TAPE BUFFER #2. THE
9 REM PROGRAM IS EXECUTED USING "USR".
10 POKE(01),58
20 POKE(02),03
29 REM..INIT...INITALIZE VARIABLES
30 POKE(826),169
40 POKE(827),00
50 POKE(828),141
60 POKE(829),251
70 POKE(830),03
80 POKE(831),170

88 REM..LOOP1..TEST FIRST CHAR ON EACH
89 REM  LINE FOR AN "R".
90 POKE(832),189
100 POKE(833),80
110 POKE(834),128
150 POKE(835),201
160 POKE(836),18
170 POKE(837),240
180 POKE(838),83
189 REM..LOOP3..PRINT CR/LF
190 POKE(839),169
200 POKE(840),13
210 POKE(841),141
220 POKE(842),255
230 POKE(843),03
240 POKE(844),32
250 POKE(845),166
260 POKE(846),03

270 POKE(847),169
280 POKE(848),10
290 POKE(849),141
300 POKE(850),255
310 POKE(851),03
320 POKE(852),32
330 POKE(853),166
340 POKE(854),03
348 REM..LOOP2..EXAMINE AND PRINT THE
349 REM  OTHER CHARACTERS ON THE LINE.
350 POKE(855),189
360 POKE(856),80
370 POKE(857),128
380 POKE(858),141
390 POKE(859),252
400 POKE(860),03
410 POKE(861),56
420 POKE(862),233
430 POKE(863),32
440 POKE(864),48
450 POKE(865),12
460 POKE(866),173
470 POKE(867),252
480 POKE(868),03
490 POKE(869),141
500 POKE(870),255
510 POKE(871),03
520 POKE(872),32
530 POKE(873),166
540 POKE(874),03
550 POKE(875),76
560 POKE(876),122
570 POKE(877),03
579 REM..ALPHA..PRINT ALPHABETIC CHAR
580 POKE(878),173
580 POKE(878),173
590 POKE(879),252
600 POKE(880),03
610 POKE(881),24
620 POKE(882),105
630 POKE(883),64
640 POKE(884),141
650 POKE(885),255
660 POKE(886),03
670 POKE(887),32
680 POKE(888),166

690 POKE(889),03
698 REM..CLNUP..COUNT CHARACTERS AND
699 REM  TEST FOR END OF LINE.
700 POKE(890),238
710 POKE(891),251
720 POKE(892),03
730 POKE(893),173
740 POKE(894),251
750 POKE(895),03
760 POKE(896),201
770 POKE(897),40
780 POKE(898),240
790 POKE(899),13
800 POKE(900),232
810 POKE(901),138
820 POKE(902),208
830 POKE(903),06
840 POKE(904),238
850 POKE(905),89
860 POKE(906),03
861 POKE(907),238
862 POKE(908),66
863 POKE(909),03
870 POKE(910),76
880 POKE(911),87
890 POKE(912),03
899 REM..NEWL..INITALIZES NEW LINE.
900 POKE(913),169
910 POKE(914),00
911 POKE(915),141
912 POKE(916),251
913 POKE(917),03
914 POKE(918),232

917 POKE(919),76
918 POKE(920),64
919 POKE(921),03
920 REM..FINDR..PROGRAM COMES HERE IF
921 REM  AN "R" IS FOUND IN 1ST COLM.
921 POKE(922),169
922 POKE(922),169
923 POKE(923),128
924 POKE(924),141
925 POKE(925),66
926 POKE(926),03
927 POKE(927),141
928 POKE(928),89
929 POKE(929),03
930 POKE(930),96
```

```
949 REM..PRINT..THIS SUBROUTINE PRINTS
950 REM  THE CHARACTER IN TTY FORMAT.
960 POKE(934),169
961 POKE(935),255
962 POKE(936),141
963 POKE(937),67
964 POKE(938),232
965 POKE(939),173
966 POKE(940),255
970 POKE(941),03
980 POKE(942),141
990 POKE(943),252
1000 POKE(944),03
1010 POKE(945),142
1020 POKE(946),253
1030 POKE(947),03
1040 POKE(948),32
1050 POKE(949),230
1060 POKE(950),03
1070 POKE(951),169
1080 POKE(952),79
1090 POKE(953),232
1100 POKE(954),41
1110 POKE(955),254
1120 POKE(956),141
1130 POKE(957),79
1140 POKE(958),232
1150 POKE(959),32
1160 POKE(960),230
1170 POKE(961),03
1180 POKE(962),162
1190 POKE(963),08
1199 REM..OUT1
1200 POKE(964),173

1210 POKE(965),79
1220 POKE(966),232
1230 POKE(967),41
1240 POKE(968),254
1250 POKE(969),78
1260 POKE(970),252
1270 POKE(971),03
1280 POKE(972),105
1290 POKE(973),00
1300 POKE(974),141
1310 POKE(975),79
1320 POKE(976),232
1330 POKE(977),32
1340 POKE(978),230
1350 POKE(979),03

1360 POKE(980),202
1370 POKE(981),208
1380 POKE(982),237
1390 POKE(983),173
1400 POKE(984),79
1410 POKE(985),232
1420 POKE(986),09
1430 POKE(987),01
1440 POKE(988),141
1450 POKE(989),79
1460 POKE(990),232
1470 POKE(991),32
1480 POKE(992),230
1490 POKE(993),03
1500 POKE(994),174
1510 POKE(995),253
1520 POKE(996),03
1530 POKE(997),96
1539 REM..DELAY
1540 POKE(998),169
1550 POKE(999),02
1560 POKE(1000),141
1570 POKE(1001),254
1580 POKE(1002),03
1590 POKE(1003),169
1600 POKE(1004),82
1609 REM..DE2
1610 POKE(1005),56
1619 REM..DE4
1620 POKE(1006),233

1630 POKE(1007),01
1640 POKE(1008),176
1650 POKE(1009),03
1660 POKE(1010),206
1670 POKE(1011),254
1680 POKE(1012),03
1689 REM..DE3
1690 POKE(1013),172
1700 POKE(1014),254
1710 POKE(1015),03
1720 POKE(1016),16
1730 POKE(1017),243
1740 POKE(1018),96
1750 REM..COUNT(1019)
1760 REM..CHAR (1020)
1770 REM..TMPX (1021)
1780 REM..TIMH (1022)
1790 REM..PCHAR(1023)
1800 END
```

LABEL	OP	FIELD	LOC	OP	F1	F2
INIT	LDA	#0	826	169	00	
	STA	COUNT	828	141	251	03
	TAX		831	170		
LOOP1	LDA	32848,X	832	189	80	128
	CMP	#18	835	201	18	
	BEQ	FINDR	837	240	83	
LOOP3	LDA	#0D	839	169	13	
	STA	PCHAR	841	141	255	03
	JSR	PRINT	844	32	166	03
	LDA	#0A	847	169	10	
	STA	PCHAR	849	141	255	03
	JSR	PRINT	852	32	166	03
LOOP2	LDA	32848,X	855	189	80	128
	STA	CHAR	858	141	252	03
	SEC		861	56		
	SBC	#20	862	233	32	
	BMI	ALPHA	864	48	12	
	LDA	CHAR	866	173	252	03
	STA	PCHAR	869	141	255	03
	JSR	PRINT	872	32	166	03
	JMP	CLNUP	875	76	122	03

```
ALPHA   LDA  CHAR     878    173  252   03
        CLC           881     24
        ADC   #40     882    105   64
        STA  PCHAR    884    141  255   03
        JSR  PRINT    887     32  166   03
CLNUP   INC  COUNT    890    238  251   03
        LDA  COUNT    893    171  251   03
        CMP   #28     896    201   40
        BEQ  NEWL     898    240   13
        INX           900    232
        TAX           901    138
        BNE  NEXTC    902    208   06
        INC   869     904    238   89   03
        INC   834     907    238   66   03
NEXTC   JMP  LOOP2    910     76   87   03
NEWL    LDA   #0      913    169   00
        STA  COUNT    915    141  251   03
        INX           918    232
        JMP  LOOP1    919     76   64   03
FINDR   LDA   #80     922    169  128
        STA   834     924    141   66   03
        STA   860     927    141   89   03
        RTS           930     96
PRINT   LDA   #FF     934    169  255
        STA  PADD     936    141   67  232
        LDA  PCHAR    939    173  255   03
        STA  CHAR     942    141  252   03
        STX   TMPX    945    142  253   03
        JSR   DELAY   948     32  230   03
        LDA   SAD     951    169   79  232
        AND    #FE    954     41  254
        STA   SAD     956    141   79  232
        JSR   DELAY   959     32  230   03
        LDX    #08    962    162   08
OUT1    LDA   SAD     964    173   79  232
        AND   #FE     967     41  254
        LSR   CHAR    969     78  252   03
        ADC   #00     972    105   00
        STA   SAD     974    141   79  232
        JSR   DELAY   977     32  230   03
        DEX           980    202
        BNE   OUT1    981    208  237
        LDA   SAD     983    173   79  232
        ORA   #01     986     09   01
        STA   SAD     988    141   79  232
        JSR   DELAY   991     32  230   03
        LDX   TMPX    994    174  253   03
        RTS           997     96
DELAY   LDA   #02     998    169   02
        STA   TIMH    1000   141  254   03
        LDA   #52     1003   169   82
DE2     SEC           1005    56
DE4     SBC   #01     1006   233   01
        BCS   DE3     1008   176   03
        DEC   TIMH    1010   206  254   03
DE3     LDY   TIMH    1013   172  254   03
        BPL   DE2     1016    16  243
        RTS           1018

COUNT     (1019)
CHAR      (1020)
TMPX      (1021)
TIMH      (1022)
PCHAR     (1023)
```

THE PET VET EXAMINES SOME BASIC IDIOSYNCRASIES

Charles Floto
325 Pennsylvania Ave., S.E.
Washington, DC 20003

Richard Rosner has supplied a program listing produced using his RS-232 printer interface for the PET. As it's well commented I'll only point out examples of some of the unusual features of PET BASIC.

Line 1 is an example of the OPEN statement. The first number specifies that it applies to logical file number 5. This is the name by means of which other statements in the program will use this data file. The second number specifies that physical device number 5 is being used. Which device is number 5 is determined by the wiring of the system.

The PET, as sold, is wired for device 0 the keyboard; 1, the built-in tape drive; 2, the auxiliary drive connector on the back; and 3, the screen. Referring to a physical device that hasn't been electrically connected will result in a DEVICE NOT PRESENT ERROR. Richard's system does contain a physical device 5: his RS-232 output port.

If the third number in the OPEN statement is 0, reading the file is enabled. Writing is prepared for by 1, while a 2 here enables file writing with an end-of-tape character to be added when the file is CLOSEd.

Line 2 illustrates the use of CMD. It allows program commands to be applied to a device specified by the logical file connected with it (not by the physical device number). Note that RUN will merely cause a listing to be produced. RUN 5 calls the rest of the program into action.

Line 2000 demonstrates use of the OPEN statement with a variable. Lines 2000-2300 print data either on the tape drive or on the screen depending on which device number is the current value of variable D. In each case logical file 8 is used.

Another idiosyncrasy comes up here: while PRINT may be entered as ?, PRINT# cannot be entered as ?# - it must be spelled out. Otherwise a SYNTAX ERROR will result when the program is run, even though the listing will look alright.

But you can still save a good deal of typing entering these lines. Once 2110 is in simply move the cursor up to change the line number to 2111 and NA to AD. Then hit RETURN and you'll have both 2110 and 2111 in memory.

I suggest you make a few changes in Richard's program. Add 105 DIM ST$(CO) Consider storing the zip code as a string rather than as an integer. Repeat lines 2000-2300 as 5000-5300 (by changing the first digit in each line number) and change line 4500 accordingly. Then you can alter the display format without messing up the tape format. And remember that you can slow screen printing by holding the RVS key down.

A final note: I understand Commodore is now using a different tape drive and recording system. This may create compatibility problems in exchanging programs between the early PETs and the later ones.

```
1 OPEN 5,5,1,"Mailing List Program (Incomplete)"
2 CMD5:PRINT"":LIST:END
5 REM THE ABOVE LINES LIST THE PROGRAM ON THE HARD COPY UNIT
10 REM
11 REM WRITTEN BY RICHARD ROSNER
12 REM              BROOKFIELD, CONN.
13 REM FOR THE COMMODORE PET.
14 REM PRINTED ON A GE PRINTER
15 REM USING A PET ADA AVAILABLE FROM THE AUTHOR.
49 REM D=DEVICE CODE
```

```
50 D=1:REM TAPE DRIVE #1
90 CO=50
91 REM CO=MAX NO. OF RECORDS IN LIST
100 DIM NA$(CO),AD$(CO),CI$(CO)
101 REM NA$=NAME,AD$=ADDRESS,CI$=CITY
102 REM ST$=STATE,Z=ZIP CODE
103 REM KC=KEY CODE. UP TO 10 FOR EACH ADDRESS
110 DIM Z(CO),KC%(10,CO)
997 REM ENTER RECORDS FOR MAILING LIST
998 REM EXIT ON '!' FOR NAME
1000 FOR N=0 TO CO
1010 INPUT"NAME";NA$(N)
1020 IF NA$(N)="!" GOTO 2000
1025 LN=N
1030 INPUT"ADDRESS";AD$(N)
1040 INPUT"CITY,STATE";CI$(N),ST$(N)
1050 INPUT"ZIP CODE"; Z(N)
1060 FOR N1=0 TO 10
1070 PRINT "KEY#";N1;:INPUT KC%(N1,N)
1080 IF KC%(N1,N)=0 GOTO 1180
1100 NEXTN1
1180 NEXT N
1998 PRINT ON TAPE DRIVE(D=1) OR SCREEN (D=3)
2000 OPEN 8,D,1,"ADDRESS FILE"
2009 REM LN=NUMBER OF RECORDS
2010 PRINT#8,LN
2100 FOR N=0 TO LN
2110 PRINT#8,NA$(N)
2111 PRINT#8,AD$(N)
2112 PRINT#8,CI$(N)
2113 PRINT#8,ST$(N)
2115 PRINT#8,Z(N)
2120 FOR N1=0 TO 10
2130 PRINT#8,KC%(N1,N)
2150 NEXT N1
2200 NEXT N
2300 CLOSE 8
3000 END
3997 REM ENTER AT 4000 TO READ IN FROM TAPE
3998 REM DRIVE N0. 1 AND THEN PRINT ON SCREEN
4000 OPEN 8,1,0,"ADDRESS FILE"
4010 INPUT#8,LN
4011 PRINTLN:REM PRINT RECORD COUNT
4100 FOR N=0 TO LN
4110 INPUT#8,NA$(N)
4120 REM IF ST1 AND 64 GOTO 4300
4130 INPUT#8,AD$(N)
4131 INPUT#8,CI$(N)
4132 INPUT#8,ST$(N)
4135 INPUT#8,Z(N)
4140 FOR N1=0 TO 10
4150 INPUT#8,KC%(N1,N)
4160 NEXTN1
4190 PRINTN:REM PRINT RECORD NO. AS READ
4200 NEXT N
4300 CLOSE 8
4500 D=3:GOTO 2000
READY.
```

THE PET VET TACKLES DATA FILES

Charles Floto
325 Pennsylvania Ave., S.E.
Washington, DC 20003

Several people have contacted the PET Vet about their difficulties in recording data files on tape and reading the information back in. Preliminary information on PET BASIC lists the commands to be used, but doesn't tell how to put them together. This makes for a frustrating situation, especially as file handling should be one of the PET's strong points.

The following program is offered as a starting point for development according to your specific application. Reading and writing have been combined in one program for two reasons. First, modifications to one process may call for corresponding changes in the other. Second, this minimizes the need to juggle two cassettes while saving programs on one and data on the other. I recommend that a separate cassette be used for data storage. If you use this program please save it on tape before you try to run it. I have found that while I'm experimenting with data files, the PET is especially liable to go out of control, forcing me to turn off the power. The same memory location that controls the tape drive apparently controls a function essential to BASIC.

To write a data file to load this program, have a blank cassette in the tape drive and type RUN. Line 50 clears the screen. Lines 60-300 build a string consisting of: a file name or record number followed by two asterisks; data to be saved that may be broken into data fields by delimiters of your choice; and three consecutive backslashes that mark the end of the record. Lines 90 and 100 cause the keyboard to be read until a key is struck. Then 105 echoes it to the screen and 110 adds it to the string. Use of GET rather than INPUT allows the data file to contain commas and carriage returns. Line 190 warns when C$ is approaching the maximum size; you may wish to have a later or less frequent warning. At the end of the record type three backslashes. These will be detected in line 300, causing 320 to be executed rather than going back to 90 for another character.

Lines 320-400 write C$ onto the tape. You will be instructed (on the screen) to press play and record on the tape drive if you have not already done so. In line 320 the first two numbers indicate that device #1 is tape drive 1. The third 1 indicates a write operation. Compare this to line 1000 where the 0 indicates a read command.

Line 450 provides for creation of the next record in the file. To create the last record simply input the record number and type three backslashes. Then, after it has been written, BREAK IN 500 will appear on the screen.

At this point you're ready to rewind the tape and type RUN 900. Lines 910 to 990 initialize 256 empty strings. Lines 1000-1090 read the tape and build up C$ until three consecutive backslashes are found. Line 2000 prints what has been read while 2850 displays available memory. Then in 3000-3020 C$ is broken down into its individual elements. These can be manipulated further by adding your own lines between 3050 and 9000. Line 9000 will head back to read the next record unless 3050 has detected the last record in a file.

To record numeric data generated in a program rather than entered from the keyboard it must be converted to a string with the STR$ function. Then when it's read back the VAL function can be used on data fields representing numbers. For example, N=VAL(B$(8)+B$(9)+B$(10)) might be used if you knew the eighth, ninth and tenth elements of C$ represented a three-digit number. Of course, it usually won't be nearly so simple as that.

If you have any problems with specific applications of your PET, drop me a note, preferably giving a phone number where you can be reached evenings and weekends. I'd also be interested to see any information you've been able to pry out of Commodore or discover on your own.

```
50 PRINT CHR$(147)
60 PRINT "ENTER FILE NAME OR RECORD #"
70 INPUT C$
80 C$=C$+"**"
90 GET A$
100 IF A$=""THEN 90
105 PRINT A$;
110 C$=C$+A$
190 IF LEN(C$)>200 THEN PRINT 255-LEN(C$); "BYTES AVAILABLE"
300 IF RIGHT$(C$,3)<>"\\\" THEN 90
320 OPEN 1,1,1,"NAILFILE"
350 PRINT#1,C$
400 CLOSE 1
450 IF RIGHT$(C$,5)<>"**\\\" THEN 50
500 STOP
900 DIM B$(255)
910 FOR J=1 TO 255
920 B$(J)=""
930 NEXT J
990 C$=""
1000 OPEN 1,1,0,"NAILFILE"
1010 GET#1,A$
1020 C$=C$+A$
1030 IF A$<>"\" THEN SL=0:GOTO 1010
1040 SL=SL+1
1050 IF SL<3 THEN 1010
1090 CLOSE 1
2000 PRINT C$
2850 PRINT FRE(0); "BYTES FREE"
3000 FOR J=1 TO LEN(C$)
3010 B$(J)=MID$(C$,J,1)
3020 NEXT J
3030 PRINT FRE(0); "BYTES FREE"
3050 IF RIGHT$(C$,5)="**\\\" THEN END
9000 GOTO 910
```

A PARTIAL LIST OF PET SCRATCH PAD MEMORY

Gary A. Creighton
625 Orange Street, No. 43
New Haven, CT 06510

A function and a symbol defined:

```
DEF FN IND(LOC) = PEEK(LOC+!)*256+PEEK(LOC)
       Which specifies an indirect address in the form:LOC+1=(Page)
                                                       LOC  =(Item)
M(LOC)          specifies contents of a memory location.
```

M(0)	JMP instruction
FN IND(1)	USR jump location
M(3)	Present I/O Device Number (suppress printout)
M(5)	POS function store
FN IND(8)	Arguments of commands with range 0 to 65535 (PEEK,POKE,WAIT,SYS,GOTO,GOSUB,Line Number,RAM check)
M(10-89)	Input Buffer
M(90-98)	Flags for MISMATCH, Distinguishing between similar subroutines, etc.
M(91)	Ignore Code Value and do direct (between quotes, etc.)
M(98)	(0 INPUT, 64 GET/GET#, 152 READ) Flag
FN IND(113)	Transfer Number pointer
FN IND(115)	Number pointer
FN IND(122)	Begin Basic Code pointer
FN IND(124)	Begin Variables pointer
FN IND(126)	Variable List pointer
FN IND(128)	End Variables pointer
FN IND(130)	Lowest String Variables pointer
FN IND(132)	Highest String Variables pointer
FN IND(134)	First Free After Strings pointer
FN IND(136)	Present Line Number (if M(137)=255, no line number)
FN IND(138)	Line Number at BREAK
FN IND(140)	Continue Run pointer (if M(141)=0, can't continue)
FN IND(142)	Line Number of Present DATA line
FN IND(144)	Next DATA pointer (for READ)
FN IND(146)	Next Data/Input After Last Comma pointer
M(148)	Coded 1st Character of Last Variable
M(149)	Coded 2nd Character of Last Variable
FN ND(150)	Variable pointer (all variables)
FN IND(152)	Variable pointer
M(156)	Comparison Symbol Accumulator (<=>)
FN IND(157)	Pointer to FN pointer
M(157-161)	Number Store/Work area (SQR)
M(163-165)	JMP (FN IND(164))
FN IND(164)	Function Jump address
M(166-170)	Number Store/Work area (Transcendentals (not EXP) & SQR)
M(171-175)	Number Store/Work area (Transcendentals & SQR)
M(176-181)	Main Number Store/Work area
M(181)	Number Sign
M(184-189)	Secondary Number Store/Work area
M(192)	Length of things in Input Buffer M(10-89) or Length of things in Output Number M(256-)...other
M(194-217)	Subroutine: Point through code one at a time, RTS with code value in accumulator and Carry Flag Clear if ASC(0-9) 0 if end of line. Ignore Spaces.
FN IND(201)	Code Pointer
M(218-222)	Number Store/Work area (RND)
FN IND(224)	Screen Memory Row location
M(226)	Screen Column position

FN IND(227)	Move Memory (from or to) pointer
M(234)	Quote flag (0 end quote)(1 begin quote)
M(238)	Length of File name after SAVE,VERIFY etc.
M(239)	File #
M(240)	I/O Option (0 read, 1 write, 2 write/EOT)
M(241)	Device # (0 keyboard, 1 tape#1, 2 tpae#2, 3 screen)
M(242)	Wraparound flag (39 single line, 79 2nd of double line)
FN IND(243)	Tape #1 or #2 Buffer pointer
M(245)	Screen Row (0 - 24)
FN IND(247)	Load into/ Verify from? Save into pointer
M(251)	Insert Counter (INST)
M(256)	Minus sign or Space for Output Number
M(256-)	Output Number ASC Digits til a Null (0) or Tape Read Working Storage
M(311?-511)	Stack area
M(512-514)	TI clock
M(515)	Only One Value per Keypush flag
M(516)	SHIFT flag (0 no shift, 1 shift)
M(517-518)	TI Update Interrupt Counter
M(521) or	Bit Cancel Keys
M(59410)	Turns bits off under the following rules:

BIT	KET	DECIMAL #	
0	RVS	254	
1		253	
2	space	251	More than one key
3		247	
4	stop	239	may be pushed at once.
5	(none)		
6		191	Decimal # is Binary
7		127	equivalent.

M(523)	VERIFY/LOAD flag (0 LOAD, 1 VERIFY)
M(524)	ST Status
M(525)	Key Pushed Counter (MOD 10)
M(526)	RVS flag (0 RVS off, 1 RVS on) or any key pushed)
M(527-536)	Input Run Buffer (keys stored during a RUN
FN IND(537)	Interrupt Vector (normally at: Store Keypush
FN IND(539)	BRK instruction Vector (User loaded) in Input Run Buffer)
M(547)	Keyboard Input Code (Stays equal to Input code til finger off key, Matches up one to one with M(59228-59307) which is Keyboard Input Code to ASC Code Table)
M(548)	Blink Cursor flag (if 0 (no key pushed))
M(549)	Cursor Blink Duration counter (20 interrupts)
M(550)	Screen Value of Input Char. when Cursor moves on
M(551)	Insure no Cursor Breadcrumbs left behind
M(553-577)	Screen Page Array / single or double Line flags
M(578-587)	File # of one of 10 files
M(588-597)	Device # of one of 10 files
M(598-607	I/O option one of 10 files
M(608)	Input from screen/Input from keyboard flag
M(610)	Number of Open Files
M(611)	Device Number of Input Device (0 keyboard normally)
M(612)	Device Number of Output Device (3 screen normally)
M(616)	Tape Buffer Item Counter
M(634-825)	Tape #1 Buffer area
M(826-1023)	Tape #2 Buffer area

LIFE FOR YOUR PET

Dr. Frank H. Covitz
Deer Hill Road
Lebanon, NJ 08833

Since this is the first time I have attempted to set down a machine language program for the public eye, I will attempt to be as complete as practical without overdoing it.

The programs I will document here are concerned with the game of "LIFE", and are written in 6502 machine language specifically for the PET 2001 (8K version). The principles apply to any 6502 system with graphic display capability, and can be debugged (as I did) on non-graphic systems such as the KIM-1.

The first I heard of LIFE was in Martin Gardner's "Recreational Mathematics" section in Scientific American, Oct-Nov 1970; Feb. 1971. As I understand it, the game was invented by John H. Conway, an English mathematician. In brief, LIFE is a "cellular automation" scheme, where the arena is a rectangular grid (ideally of infinite size). Each square in the grid is either occupied or unoccupied with "seeds", the fate of which are governed by relatively simple rules, i.e. the "facts of LIFE". The rules are: 1. A seed survives to the next generation if and only if it has two or three neighbors (right, left, up, down, and the four diagonally adjacent cells) otherwise it dies of loneliness or overcrowding, as the case may be. 2. A seed is born in a vacant cell on the next generation if it has exactly 3 neighbors.

With these simple rules, a surprisingly rich game results. The original Scientific American article, and several subsequent articles reveal many curious and surprising initial patterns and results. I understand that there even has been formed a LIFE group, complete with newsletter, although I have not personally seen it.

The game can of course be played manually on a piece of graph paper, but it is slow and prone to mistakes, which have usually disasterous effects on the final results. It would seem to be the ideal thing to put to a microprocessor with bare-bones graphics, since the rules are so simple and there are essentially no arithmetic operations involved, except for keeping track of addresses and locating neighbors.

As you know, the PET-2001 has an excellent BASIC interpreter, but as yet very little documentation on machine language operation. My first stab was to write a BASIC program, using the entire PET display as the arena (more about boundaries later), and the filled circle graphic display character as the seed. This worked just fine, except for one thing - it took about 2-1/2 minutes for the interpreter to go through one generation! I suppose I shouldn't have been surprised since the program has to check eight neighboring cells to determine the fate of a particular cell, and do this 1000 times to complete the entire generation (40x25 characters for the PET display).

The program following is a 6502 version of LIFE written for the PET. It needs to be POKE'd into the PET memory, since I have yet to see or discover a machine language monitor for the PET. I did it with a simple BASIC program and many DATA statements (taking up much more of the program memory space than the actual machine language program!). A routine for assembling, and saving on tape machine language programs on the PET is sorely needed.

The program is accessed by the SYS command, and takes advantage of the display monitor (cursor control) for inserting seeds, and clearing the arena. Without a serious attempt at maximizing for speed, the program takes about 1/2 second to go through an entire generation, about 300 times faster than the BASIC equivalent! Enough said about the efficiency of machine language programming versus BASIC interpreters?

BASIC is great for number crunching, where you can quickly compose your program and have plenty of time to await the results.

The program may be broken down into manageable chunks by subroutining. There follows a brief description of the salient features of each section:

MAIN (hex 1900)

In a fit of overcaution (since this was the first time I attempted to write a PET machine language program) you will notice the series of pushes at the beginning and pulls at the end. I decided to save all the internal registers on the stack in page 1, and also included the CLD (clear decimal mode) just in case. Then follows a series of subroutine calls to do the LIFE generation and display transfers. The zero page location, TIMES, is a counter to permit several loops through LIFE before returning. As set up, TIMES is initialized to zero (hex location 1953) so that it will loop 256 times before jumping back. This of course can be changed either initially or while in BASIC via the POKE command. The return via the JMP BASIC (4C 8B C3) may not be strictly orthodox, but it seems to work all right.

INIT (hex 1930) and DATA (hex 193B)

This shorty reads in the constants needed, and stores them in page zero. SCR refers to the PET screen, TEMP is a temporary working area to hold the new generation as it is evolved, and RCS is essentially a copy of the PET screen data, which I found to be necessary to avoid "snow" on the screen during read/write operations directly on the screen locations. Up, down, etc. are the offsets to be added or subtracted from an address to get all the neighbor addresses. The observant reader will note the gap in the addresses between some of the routines.

TMPSCR (hex 1970)

This subroutine quickly transfers the contents of Temp and dumps it to the screen, using a dot (81 dec) symbol for a live cell (a 1 in TEMP) and a space (32 dec) for the absence of a live cell (a 0 in TEMP).

SCRTMP (hex 198A)

This is the inverse of TMPSCR, quickly transferring (and encoding) data from the screen into TEMP.

RSTORE (hex 19A6)

This subroutine fetches the initial addresses (high and low) for the SCR, TEMP, and RCS memory spaces.

NXTADR (hex 19BD)

Since we are dealing with 1000 bytes of data, we need a routine to increment to the next location, check for page crossing (adding 1 to the high address when it occurs), and checking for the end. The end is signaled by returning a 01 in the accumulator, otherwise a 00 is returned via the accumulator.

TMPRCS (hex 19E6)

The RCS address space is a copy of the screen, used as mentioned before to avoid constant "snow" on the screen if the screen were being continually accessed. This subroutine dumps data from TEMP, where the new generation has been computed, to RCS.

GENER (hex 1A00)

We finally arrive at a subroutine where LIFE is actually generated. After finding out the number of neighbors of the current RCS data byte from NBRS, GENER checks for births (CMPIM $03 at hex addr. 1A0E) if the cell was previously unoccupied. If a birth does not occur, there is an immediate branch to GENADR (the data byte remains 00). If the cell was occupied (CMPIM 81 dec at hex 1A08), OCC checks for survival (CMPIM $03 at hex 1A1A and CMPIM $02 at hex 1A1E), branching to GENADR when these two conditions are met, otherwise the cell dies (LDAIM $00 at hex 1A22). The results are stored in TEMP for the 1000 cells.

NBRS (hex 1A2F)

NBRS is the subroutine that really does most of the work and where most of the speed could be gained by more efficient programming. Its job, to find the total number of occupied neighbors of a given RCS data location, is complicated by page crossing and edge boundaries. In the present version, page crossing is taken care of, but edge boundaries (left, right, top, and bottom of the screen) are somewhat "strange". Above the top line and below the bottom line are considered as sort of forbidden regions where there should practically always be no "life" (data in those regions are not defined by the program, but I have found that there has never been a case where 81's have been present (all other data is considered as "unoccupied" characters). The right and left edges are different, however, and lead to a special type of "geometry". A cell at either edge is not considered as special by NBRS, and so to the right of a right-edge location is the next sequential address. On the screen this is really the left edge location, and one line lower. The inverse is true, of course for left addresses of left-edge locations. Topologically, this is equivalent to a "helix". No special effects of this are seen during a simple LIFE evolution since it just gives the impression of disappearing off one edge while appearing on the other edge. For an object like the "spaceship" (see Scientific American articles), then, the path eventually would cover the whole LIFE arena. The fun comes in when a configuration spreads out so much that it spills over both edges, and interacts with itself. This, of course cannot happen in an infinite universe, so that some of the more complex patterns will not have the same fate in the present version of LIFE. Most of the "blinkers", including the "glider gun" come out OK.

This 40x25 version of LIFE can undoubtedly be made more efficient, and other edge algorithms could be found, but I chose to leave it in its original form as a benchmark for my first successfully executed program in writing machine language on the PET. One confession, however - I used the KIM-1 to debug most of the subroutines. Almost all of them did not run on the first shot! Without a good understanding of PET memory allocation particularly in page zero, I was bound to crash many times over, with no recovery other than pulling the plug. The actual BASIC program consisted of a POKING loop with many DATA statements (always save on tape before running!).

Although the LIFE program was designed for use on the PET (8K version), no references are made to PET ROM locations or subroutines, and except for MAIN and SUBROUTINE address, are fully relocatable. The PET screen addresses (8000 - 83E8 hex) are treated as RAM. For anyone (with a 6502-based system) trying to convert the PET program, the following points need to be watched:

1. The BLANK symbol = 20 hex
2. The DOT symbol = 51 hex
3. The OFFSETs in DATA must be set for the user's display.

A Brief Introduction to the Game of Life

by Mike Rowe

One of the interesting properties of the game of LIFE is that such simple rules can lead to such complex activity. The simplicity comes from the fact that the rules apply to each individual cell. The complexity comes from the interactions between the individual cells. Each individual cell is affected by its eight adjacent neighbors, and nothing else.

The rules are:

1. A cell survives if it has two or three neighbors.

2. A cell dies from overcrowding if it has four or more neighbors. It dies from isolation if it has one or zero neighbors.

3. A cell is born when an empty space has exactly three neighbors.

With these few rules, many different types of activity can occur. Some patterns are STABLE, that is they do not change at all. Some are REPEATERS, patterns which undergo one or more changes and return to the original pattern. A REPEATER may repeat as fast as every other generation, or may have a longer period. A GLIDER is a pattern which moves as it repeats.

```
                                      REPEATERS

       STABLE                   * *     *                         GLIDERS
                                * *       * *      * * *
           *                * *       * *        * * *                         *
  * *    *   *    *   * *   * *           *                   *                  *
  * *    *   *    * *   *                                       *    *           *
           *                                                * * *      * * * * * *
```

```
1900                LIFE   ORG   $1900

1900                BASIC  *     $C38B  RETURN TO BASIC ADDRESS
1900                OFFSET *     $002A  PAGE ZERO DATA AREA POINTER
1900                DOT    *     $0051  DOT SYMBOL = 81 DECIMAL
1900                BLANK  *     $0020  BLANK SYMBOL = 32 DECIMAL

1900                SCRL   *     $0020  PAGE ZERO LOCATIONS
1900                SCRH   *     $0021
1900                CHL    *     $0022
1900                CHH    *     $0023
1900                SCRLO  *     $0024
1900                SCRHO  *     $0025
1900                TEMPL  *     $0026
1900                TEMPH  *     $0027
1900                TEMPLO *     $0028
1900                TEMPHO *     $0029
1900                UP     *     $002A
1900                DOWN   *     $002B
1900                RIGHT  *     $002C
1900                LEFT   *     $002D
1900                UR     *     $002E
1900                UL     *     $002F
1900                LR     *     $0030
1900                LL     *     $0031
1900                N      *     $0032
1900                SCRLL  *     $0033
1900                SCRLH  *     $0034
1900                RCSLO  *     $0035
1900                RCSHO  *     $0036
1900                TMP    *     $0037
1900                TIMES  *     $0038
1900                RCSL   *     $0039
1900                RCSH   *     $003A

1900 08             MAIN   PHP          SAVE EVERYTHING
1901 48                    PHA          ON STACK
1902 8A                    TXA
1903 48                    PHA
1904 98                    TYA
1905 48                    PHA
1906 BA                    TSX
1907 8A                    TXA
1908 48                    PHA
1909 D8                    CLD          CLEAR DECIMAL MODE
190A 20 30 19              JSR   INIT
190D 20 8A 19              JSR   SCRTMP
1910 20 E6 19       GEN    JSR   TMPRCS
1913 20 00 1A              JSR   GENER
1916 20 70 19              JSR   TMPSCR
1919 E6 38                 INCZ  TIMES  REPEAT 255 TIMES
191B D0 F3                 BNE   GEN    BEFORE QUITTING
191D 68                    PLA          RESTORE EVERYTHING
191E AA                    TAX
191F 9A                    TXS
1920 68                    PLA
```

```
1921 A8             TAY
1922 68             PLA
1923 AA             TAX
1924 68             PLA
1925 28             PLP
1926 4C 8B C3       JMP   BASIC  RETURN TO BASIC

1930                ORG   $1930

             MOVE VALUES INTO PAGE ZERO

1930 A2 19   INIT   LDXIM $19    MOVE 25. VALUES
1932 BD 3A 19 LOAD  LDAX  DATA   -01
1935 95 1F          STAZX $1F    STORE IN PAGE ZERO
1937 CA             DEX
1938 D0 F8          BNE   LOAD
193A 60             RTS

193B 00      DATA   =     $00    SCRL
193C 80             =     $80    SCRH
193D 00             =     $00    CHL
193E 15             =     $15    CHH
193F 00             =     $00    SCRLO
1940 80             =     $80    SCRHO
1941 00             =     $00    TEMPL
1942 1B             =     $1B    TEMPH
1943 00             =     $00    TEMPLO
1944 1B             =     $1B    TEMPHO
1945 D7             =     $D7    UP
1946 28             =     $28    DOWN
1947 01             =     $01    RIGHT
1948 FE             =     $FE    LEFT
1949 D8             =     $D8    UR
194A D6             =     $D6    UL
194B 29             =     $29    LR
194C 27             =     $27    LL
194D 00             =     $00    N
194E E8             =     $E8    SCRLL
194F 83             =     $83    SCRLH
1950 00             =     $00    RCSLO
1951 15             =     $15    RCSHO
1952 00             =     $00    TMP
1953 00             =     $00    TIMES

1970                ORG   $1970

1970 20 A6 19 TMPSCR JSR  RSTORE GET INIT ADDRESSES
1973 B1 26   TSLOAD LDAIY TEMPL  FETCH BYTE FROM TEMP
1975 D0 06          BNE   TSONE  BRANCH IF NOT ZERO
1977 A9 20          LDAIM BLANK  BLANK SYMBOL
1979 91 20          STAIY SCRL   DUMP IT TO SCREEN
197B D0 04          BNE   TSNEXT
197D A9 51   TSONE  LDAIM DOT    DOT SYMBOL
197F 91 20          STAIY SCRL   DUMP IT TO SCREEN
1981 20 BD 19 TSNEXT JSR  NXTADR FETCH NEXT ADDRESS
1984 F0 ED          BEQ   TSLOAD
```

```
1986 20 A6 19          JSR    RSTORE RESTORE INIT ADDRESSES
1989 60                RTS

198A 20 A6 19   SCRTMP JSR    RSTORE GET INIT ADDRESSES
198D B1 20      STLOAD LDAIY SCRL   READ DATA FROM SCREEN
198F C9 51             CMPIM DOT    TEST FOR DOT
1991 F0 06             BEQ    STONE  BRANCH IF DOT
1993 A9 00             LDAIM $00    OTHERWISE ITS A BLANK
1995 91 26             STAIY TEMPL  STORE IT
1997 F0 04             BEQ    STNEXT UNCOND. BRANCH
1999 A9 01      STONE  LDAIM $01    A DOT WAS FOUND
199B 91 26             STAIY TEMPL  STORE IT
199D 20 BD 19   STNEXT JSR    NXTADR FETCH NEXT ADDRESS
19A0 F0 EB             BEQ    STLOAD
19A2 20 A6 19          JSR    RSTORE RESTORE INIT ADDRESSES
19A5 60                RTS

19A6 A9 00      RSTORE LDAIM $00    ZERO A, X, Y
19A8 AA                TAX
19A9 A8                TAY
19AA 85 20             STAZ   SCRL   INIT VALUES
19AC 85 26             STAZ   TEMPL
19AE 85 39             STAZ   RCSL
19B0 A5 25             LDAZ   SCRHO
19B2 85 21             STAZ   SCRH
19B4 A5 29             LDAZ   TEMPHO
19B6 85 27             STAZ   TEMPH
19B8 A5 36             LDAZ   RCSHO
19BA 85 3A             STAZ   RCSH
19BC 60                RTS

19BD E6 26      NXTADR INCZ   TEMPL  GET NEXT LOW ORDER
19BF E6 20             INCZ   SCRL   BYTE ADDRESS
19C1 E6 39             INCZ   RCSL
19C3 E8                INX
19C4 E4 33             CPXZ   SCRLL  IS IT THE LAST?
19C6 F0 0C             BEQ    PAGECH IS IT THE LAST PAGE?
19C8 E0 00             CPXIM $00    IS IT A PAGE BOUNDARY?
19CA D0 0E             BNE    NALOAD IF NOT, THEN NOT DONE
19CC E6 27             INCZ   TEMPH  OTHERWISE ADVANCE TO
19CE E6 21             INCZ   SCRH   NEXT PAGE
19D0 E6 3A             INCZ   RCSH
19D2 D0 06             BNE    NALOAD UNCONDITIONAL BRANCH
19D4 A5 34      PAGECH LDAZ   SCRLH  CHECK FOR LAST PAGE
19D6 C5 21             CMPZ   SCRH
19D8 F0 03             BEQ    NADONE IF YES, THEN DONE
19DA A9 00      NALOAD LDAIM $00    RETURN WITH A=0
19DC 60                RTS
19DD A9 01      NADONE LDAIM $01    RETURN WITH A=1
19DF 60                RTS

19E6                   ORG    $19E6

19E6 20 A6 19   TMPRCS JSR    RSTORE INIT ADDRESSES
19E9 B1 26      TRLOAD LDAIY TEMPL  FETCH DATA FROM TEMP
19EB D0 06             BNE    TRONE  IF NOT ZERO THEN ITS ALIVE
```

```
19ED A9 20              LDAIM BLANK  BLANK SYMBOL
19EF 91 39              STAIY RCSL   STORE IT IN SCREEN COPY
19F1 D0 04              BNE   NEWADR THEN ON TO A NEW ADDRESS
19F3 A9 51     TRONE    LDAIM DOT    THE DOT SYMBOL
19F5 91 39              STAIY RCSL   STORE IT IN SCREEN COPY
19F7 20 BD 19  NEWADR   JSR   NXTADR FETCH NEXT ADDRESS
19FA F0 ED              BEQ   TRLOAD IF A=0, THEN NOT DONE
19FC 20 A6 19           JSR   RSTORE ELSE DONE. RESTORE
19FF 60                 RTS

1A00 20 A6 19  GENER    JSR   RSTORE INIT ADDRESSES
1A03 20 2F 1A  AGAIN    JSR   NBRS   FETCH NUMBER OF NEIGHBORS
1A06 B1 39              LDAIY RCSL   FETCH CURRENT DATA
1A08 C9 51              CMPIM DOT    IS IT A DOT?
1A0A F0 0C              BEQ   OCC    IF YES, THEN BRANCH
1A0C A5 32              LDAZ  N      OTHERWISE ITS BLANK
1A0E C9 03              CMPIM $03    SO WE CHECK FOR
1A10 D0 14              BNE   GENADR A BIRTH
1A12 A9 01     BIRTH    LDAIM $01    IT GIVES BIRTH
1A14 91 26              STAIY TEMPL  STORE IT IN TEMP
1A16 D0 0E              BNE   GENADR INCONDITIONAL BRANCH
1A18 A5 32     OCC      LDAZ  N      FETCH NUMBER OF NEIGHBORS
1A1A C9 03              CMPIM $03    IF IT HAS 3 OR 2
1A1C F0 08              BEQ   GENADR NEIGHBORS IT SURVIVES
1A1E C9 02              CMPIM $02
1A20 F0 04              BEQ   GENADR
1A22 A9 00     DEATH    LDAIM $00    IT DIED!
1A24 91 26              STAIY TEMPL  STORE IT IN TEMP
1A26 20 BD 19  GENADR   JSR   NXTADR FETCH NEXT ADDRESS
1A29 F0 D8              BEQ   AGAIN  IF 0, THEN NOT DONE
1A2B 20 A6 19           JSR   RSTORE RESTORE INIT ADDRESSES
1A2E 60                 RTS

1A2F 98        NBRS     TYA          SAVE Y AND X ON STACK
1A30 48                 PHA
1A31 8A                 TXA
1A32 48                 PHA
1A33 A0 00              LDYIM $00    SET Y AND N = 0
1A35 84 32              STYZ  N
1A37 A2 08              LDXIM $08    CHECK 8 NEIGHBORS
1A39 B5 29     OFFS     LDAZX OFFSET -01
1A3B 10 15              BPL   ADD    ADD IF OFFSET IS POSITIVE
1A3D 49 FF              EORIM $FF    OTHERWISE GET SET TO
1A3F 85 37              STAZ  TMP    SUBTRACT
1A41 38                 SEC          SET CARRY BIT FOR SUBTRACT
1A42 A5 39              LDAZ  RCSL
1A44 E5 37              SBCZ  TMP    SUBTRACT TO GET THE
1A46 85 22              STAZ  CHL    CORRECT NEIGHBOR ADDRESS
1A48 A5 3A              LDAZ  RCSH
1A4A 85 23              STAZ  CHH
1A4C B0 11              BCS   EXAM   OK, FIND OUT WHAT'S THERE
1A4E C6 23              DECZ  CHH    PAGE CROSS
1A50 D0 0D              BNE   EXAM   UNCOND. BRANCH
1A52 18        ADD      CLC          GET SET TO ADD
1A53 65 39              ADCZ  RCSL   ADD
1A55 85 22              STAZ  CHL    STORE THE LOW PART
```

```
1A57 A5 3A          LDAZ  RCSH   FETCH THE HIGH PART
1A59 85 23          STAZ  CHH
1A5B 90 02          BCC   EXAM   OK, WHAT'S THERE
1A5D E6 23          INCZ  CHH    PAGE CROSSING
1A5F B1 22   EXAM   LDAIY CHL    FETCH THE NEIGHBOR
1A61 C9 51          CMPIM DOT    DATA BYTE AND SEE IF ITS
1A63 D0 02          BNE   NEXT   OCCUPIED
1A65 E6 32          INCZ  N      ACCUMULATE NUMBER OF NEIGHBORS
1A67 CA      NEXT   DEX
1A68 D0 CF          BNE   OFFS   NOT DONE
1A6A 68             PLA          RESTORE X, Y FROM STACK
1A6B AA             TAX
1A6C 68             PLA
1A6D A8             TAY
1A6E 60             RTS
```

```
SYMBOL TABLE 2000 2186
BLANK  0020   SCRL   0020   SCRH   0021   CHL    0022
CHH    0023   SCRLO  0024   SCRHO  0025   TEMPL  0026
TEMPH  0027   TEMPLO 0028   TEMPHO 0029   OFFSET 002A
UP     002A   DOWN   002B   RIGHT  002C   LEFT   002D
UR     002E   UL     002F   LR     0030   LL     0031
N      0032   SCRLL  0033   SCRLH  0034   RCSLO  0035
RCSHO  0036   TMP    0037   TIMES  0038   RCSL   0039
RCSH   003A   DOT    0051   LIFE   1900   MAIN   1900
GEN    1910   INIT   1930   LOAD   1932   DATA   193B
TMPSCR 1970   TSLOAD 1973   TSONE  197D   TSNEXT 1981
SCRTMP 198A   STLOAD 198D   STONE  1999   STNEXT 199D
RSTORE 19A6   NXTADR 19BD   PAGECH 19D4   NALOAD 19DA
NADONE 19DD   TMPRCS 19E6   TRLOAD 19E9   TRONE  19F3
NEWADR 19F7   GENER  1A00   AGAIN  1A03   BIRTH  1A12
OCC    1A18   DEATH  1A22   GENADR 1A26   NBRS   1A2F
OFFS   1A39   ADD    1A52   EXAM   1A5F   NEXT   1A67
BASIC  C38B
```

```
SYMBOL TABLE 2000 2186
ADD    1A52   AGAIN  1A03   BASIC  C38B   BIRTH  1A12
BLANK  0020   CHH    0023   CHL    0022   DATA   193B
DEATH  1A22   DOT    0051   DOWN   002B   EXAM   1A5F
GENADR 1A26   GENER  1A00   GEN    1910   INIT   1930
LEFT   002D   LIFE   1900   LL     0031   LOAD   1932
LR     0030   MAIN   1900   N      0032   NADONE 19DD
NALOAD 19DA   NBRS   1A2F   NEWADR 19F7   NEXT   1A67
NXTADR 19BD   OCC    1A18   OFFS   1A39   OFFSET 002A
PAGECH 19D4   RCSH   003A   RCSHO  0036   RCSL   0039
RCSLO  0035   RIGHT  002C   RSTORE 19A6   SCRH   0021
SCRHO  0025   SCRL   0020   SCRLH  0034   SCRLL  0033
SCRLO  0024   SCRTMP 198A   STLOAD 198D   STNEXT 199D
STONE  1999   TEMPH  0027   TEMPHO 0029   TEMPL  0026
TEMPLO 0028   TIMES  0038   TMPRCS 19E6   TMPSCR 1970
TMP    0037   TRLOAD 19E9   TRONE  19F3   TSLOAD 1973
TSNEXT 1981   TSONE  197D   UL     002F   UP     002A
UR     002E
```

A SIMPLE 6502 ASSEMBLER FOR THE PET

Michael J. McCann
28 Ravenswood Terrace
Cheektowaga, NY 14225

Most computer hobbyists do all or most of their programming in BASIC. This is unfortunate since there is much to be gained from machine code level programming. On the average, machine language programs are 100 times faster than their BASIC equivalents. In addition, machine language programs are very compact, making efficient use of memory. I have written a simple 6502 assembler in Commodore BASIC (see listing) with the following functions:

1. Input source code and assemble
2. Save object code on tape
3. Load object code from tape
4. Run machine language program with SYS
5. Run machine language program with USR
6. List machine language program

INPUT SOURCE CODE AND ASSEMBLE
- Symbolic addresses and operands are not permitted
- All addresses and operands must be supplied in base 10
- Each line of source code is assembled after entry
- Source code is inputted in the following format:
  (mnemonic)(one or more spaces)(operand)
- Three pseudoinstructions are supported
  ORG-Start with this address
  NOTE:if the user does not specify the origin, it will be set at 826 base 10
  DC-Define constant, place the operand value in the next location in memory
  END-End of program source code

SAVE OBJECT CODE ON TAPE
- Object code saved under file name supplied by user
- Origin address saved with program

LOAD OBJECT CODE FROM TAPE
- Loads object program under file name supplied by user
- Object code is stored in memory with the same origin address used when the program was assembled

RUN MACHINE LANGUAGE PROGRAM WITH SYS
- Transfers control of the 6502 to an address supplied by the user

RUN MACHINE LANGUAGE PROGRAM WITH USR
- Transfers a user supplied value to the 6502 accumulator
- Transfers control of the 6502 to an address supplied by the user

LIST MACHINE LANGUAGE PROGRAM
- Listing is produced by disassembling object code
- Disassembly is in the following format:
  (decimal address)(hexadecimal address)(byte#1)(byte#2)(byte#3)(mnemonic)(operand)

The following areas of memory are available for your machine language programs when this assembler is in memory: locations 7884-8184 and, if tape #2 is not used, locations 826-1024.

There are two ways of returning control to BASIC from machine language. The RTS (Return from Subroutine) instruction may be used at any time except when in a user machine language subroutine. RTS returns control to the calling BASIC program. In contrast the BRK (Force Break) instruction does not return control to the calling BASIC program; instead control is returned to the user, i.e. system prints READY with the cursor.

I have included a short machine language program. When run this program will leave a pattern of small white dots on the upper half of PET's CRT.

SAMPLE MACHINE LANGUAGE PROGRAM LISTING

```
826 033A  A9 66     LDAIM   102
828 033C  A2 00     LDXIM   0
830 033E  9D 00 80  STAX    32768
833 0341  E8        INX
834 0342  F0 03     BEQ     3
836 0344  4C 3E 03  JMP     830
839 0347  EA        NOP
840 0348  EA        NOP
841 0349  9D 00 81  STAX    33024
844 034C  E8        INX
845 034D  F0 03     BEQ     3
847 034F  4C 49 03  JMP     841
850 0352  00        BRK
```

SAMPLE MACHINE LANGUAGE PROGRAM
AS INPUTTED FROM THE KEYBOARD

```
? ORG 826
? LDAIM 102
? LDXIM 0
? STAX 32768
? INX
? BEQ 3
? JMP 830
? NOP
? NOP
? STAX 33024
? INX
? BEQ 3
? JMP 841
? BRK
? END
```

Two additional thoughts before you start:

1. After entering the program from the keyboard you must save it on tape before going through "RUN" again. If you don't EN and ZZ are set to zero.

2. When using the "BRK" command the system outputs the error statement "ILLEGAL QUANTITY ERROR IN 10020", READY.

```
1  REM 6502 ASSEMBLER PROGRAM
2  REM BY MICHAEL J. MCCANN
3  REM FOR USE ON THE COMMODORE PET
10  DIM MN$(256),BY%(256),CO$(16)
20  FOR E=0 TO 255
30  READ MN$(E),BY%(E)
40  NEXT
60  FOR E=0 TO 15
70  READ CO$(E)
80  NEXT
90  PRINT CHR$(147):PRINT
100  PRINT"1-INPUT SOURCE CODE AND ASSEMBLE":PRINT
110  PRINT"2-SAVE OBJECT CODE ON TAPE":PRINT
120  PRINT"3-LOAD OBJECT CODE FROM TAPE":PRINT
130  PRINT"4-RUN MACHINE LANGUAGE PROGRAM WITH SYS"
140  PRINT"5-RUN MACHINE LANGUAGE PROGRAM WITH USR"
150  PRINT"6-LIST MACHINE LANGUAGE PROGRAM"
180  GET A$:IF A$="" GOTO 180
190  IF VAL(A$)=0 OR VAL(A$)>6 GOTO 180
200  ON VAL(A$) GOSUB 14000,20000,9000,10000,11000,2900
210  GOTO 90
1000  SX=INT(DC/16)
1010  UN=DC-(SX*16)
1020  SX$=CO$(SX)
1030  UN$=CO$(UN)
1040  HX$=SX$+UN$
1050  RETURN
2900  PRINT CHR$(147)
2910  INPUT"START ADDRESS";AD:I=0
3000  IF I=24 GOTO 5050
3001  I=I+1
3005  IB=PEEK(AD)
3015  IF MN$(IB)<>"NULL" GOTO 3050
3025  DC=IB:GOSUB 1000:GOSUB 13000
3030  PRINT AD;AD$ TAB(12) HX$ "*"
3040  AD=AD+1:GOTO 3000
3050  ON BY%(IB) GOTO 3060,3090,4050
3060  DC=IB:GOSUB 1000:GOSUB 13000
3070  PRINT AD;AD$ TAB(12);HX$;TAB(21);MN$(IB)
3075  AD=AD+1
3080  GOTO 5030
3090  DC=IB:GOSUB 1000
4000  B1$=HX$
4010  DC=PEEK(AD+1):GOSUB 1000
4011  B2$=HX$
4024  GOSUB 13000:P=DC
4030  PRINT AD;AD$ TAB(12);B1$;" ";B2$;TAB(21);MN$(IB);TAB(27);I
4035  AD=AD+2
4040  GOTO 5030
4050  DC=IB:GOSUB 1000
4060  B1$=HX$
4070  DC=PEEK(AD+1):GOSUB 1000
4080  B2$=HX$
4090  DC=PEEK(AD+2):GOSUB 1000
```

```
5000  B3$=HX$
5010  OP=PEEK(AD+1)+(PEEK(AD+2)*256)
5011  GOSUB 13000
5020  PRINT AD;AD$ TAB(12);B1$;" ";B2$;" ";B3$;TAB(21);MN$(IB);TAB(27);OP
5025  AD=AD+3
5030  GOTO 3000
5050  GET A$:IF A$="" GOTO 5050
5051  IF A$=CHR$(19) THEN I=0:RETURN
5052  IF A$<>CHR$(13) GOTO 5050
5070  I=0:PRINT CHR$(147)
5080  GOTO 3000
6000  DATA BRK,1,ORAIX,2,NULL,0,NULL,0,NULL,0,ORAZ,2,ASL,2,NULL,0,PHP,1
6010  DATA ORAIM,2,ASLA,1,NULL,0,NULL,0,ORA,3,ASL,3,NULL,0,BPL,2,ORAIY,2
6020  DATA NULL,0,NULL,0,NULL,0,ORAZX,2,ASLZX,2,NULL,0,CLC,1,ORAY,3
6030  DATA NULL,0,NULL,0,NULL,0,ORAX,3,ASLX,3,NULL,0,JSR,3,ANDIX,2,NULL,0
6040  DATA NULL,0,BITZ,2,ANDZ,2,ROLZ,2,NULL,0,PLP,1,ANDIM,2,ROLA,1,NULL,0
6050  DATA BIT,3,AND,3,ROL,3,NULL,0,BMI,2,ANDIY,2,NULL,0,NULL,0,NULL,0
6060  DATA ANDZX,2,ROLZX,2,NULL,0,SEC,1,ANDY,3,NULL,0,NULL,0, NULL,0,ANDX,3
6070  DATA ROLX,3,NULL,0,RTI,1,EORIX,2,NULL,0,NULL,0,NULL,0,EORZ,2,LSRZ,2
6080  DATA NULL,0,PHA,1,EORIM,2,LSRA,1,NULL,0,JMP,3,EOR,3,LSR,3,NULL,0
6090  DATA BVC,2,EORIY,2,NULL,0,NULL,0,NULL,0,EORZX,2,LSRZX,2,NULL,0
6100  DATA CLI,1,EORY,3,NULL,0,NULL,0,NULL,0,EORX,3,LSRX,3,NULL,0,RTS,1
6110  DATA ADCIX,2,NULL,0,NULL,0,NULL,0,ADCZ,2,RORZ,2,NULL,0,PLA,1,ADCIM,2
6120  DATA RORA,1,NULL,0,JMPI,3,ADC,3,ROR,3,NULL,0,BVS,2,ADCIY,2,NULL,0
6130  DATA NULL,0,NULL,0,ADCZX,2,RORZX,2,NULL,0,SEI,1,ADCY,3,NULL,0,NULL,0
6140  DATA NULL,0,ADCX,3,RORX,3,NULL,0,NULL,0,STAIX,2,NULL,0,NULL,0,STYZ,2
6150  DATA STAZ,2,STXZ,2,NULL,0,DEY,1,NULL,0,TXA,1,NULL,0,STY,3,STA,3
6160  DATA STX,3,NULL,0,BCC,2,STAIY,2,NULL,0,NULL,0,STYZX,2,STAZX,2,STXZY,2
6170  DATA NULL,0,TYA,1,STAY,3,TXS,1,NULL,0,NULL,0,STAX,3,NULL,0,NULL,0
6180  DATA LDYIM,2,LDAIX,2,LDXIM,2,NULL,0,LDYZ,2,LDAZ,2,LDXZ,2,NULL,0
6190  DATA TAY,1,LDAIM,2,TAX,1,NULL,0,LDY,3,LDA,3,LDX,3,NULL,0,BCS,2
6200  DATA LDAIY,2,NULL,0,NULL,0,LDYZX,2,LDAZX,2,LDXZY,2,NULL,0,CLV,1
6210  DATA LDAY,3,TSX,1,NULL,0,LDYX,3,LDAX,3,LDXY,3,NULL,0,CPYIM,2,CMPIX,2
6220  DATA NULL,0,NULL,0,CPYZ,2,CMPZ,2,DECZ,2,NULL,0,INY,1,CMPIM,2,DEX,1
6230  DATA NULL,0,CPY,3,CMP,3,DEC,3,NULL,0,BNE,2,CMPIY,2,NULL,0,NULL,0
6240  DATA NULL,0,CMPZX,2,DECZX,2,NULL,0,CLD,1,CMPY,3,NULL,0,NULL,0,NULL,0
6250  DATA CMPX,3,DECX,3,NULL,0,CPXIM,2,SBCIX,2,NULL,0,NULL,0,CPXZ,2,SBCZ,2
6260  DATA INCZ,2,NULL,0,INX,1,SBCIM,2,NOP,1,NULL,0,CPX,3,SBC,3,INC,3
6270  DATA NULL,0,BEQ,2,SBCIY,2,NULL,0,NULL,0,NULL,0,SBCZX,2,INCZX,2,NULL,0,SED,1
6280  DATA SBCY,3,NULL,0,NULL,0,NULL,0,SBCX,3,INCX,3,NULL,0
6290  DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
9000  PRINT CHR$(147)
9010  INPUT "ENTER FILE NAME";N$
9020  OPEN 1,1,0,N$
9030  INPUT#1,ZZ
9040  INPUT#1,EN
9050  FOR AD=ZZ TO EN
9060  INPUT#1,DA%
9070  POKE AD,DA%
9080  NEXT
9090  CLOSE 1
9100  RETURN
```

```
10000  PRINT CHR$(147)
10010  INPUT "ENTER ADDRESS IN BASE 10";AD
10015  IF AD>65535 GOTO 10000
10020  SYS(AD)
10030  RETURN
11000  PRINT CHR$(147)
11010  INPUT"ENTER ACCUMULATOR VALUE";AC
11015  IF AC<0 OR AC>255 GOTO 11010
11020  INPUT"ENTER ADDRESS IN BASE 10";AD
11030  POKE 2,INT(AD/256)
11040  POKE 1,AD-(INT(AD/256)*256)
11050  X=USR(AC)
11060  RETURN
13000  A=AD:S3=INT(AD/4096)
13002  A=A-S3*4096
13010  S2=INT(A/256)
13012  A=A-S2*256
13020  S=INT(A/16)
13060  U=AD-(S3*4096+S2*256+S*16)
13070  S3$=CO$(S3)
13080  S2$=CO$(S2)
13090  S$=CO$(S)
13100  U$=CO$(U)
13110  AD$=S3$+S2$+S$+U$
13120  RETURN
14000  PRINT CHR$(147):AD=826:ZZ=826
14010  PRINT "(MNEMONIC)(SPACE)(OPERAND)"
14020  GOSUB 15000
14030  F=0
14040  FOR E=0 TO 255
14050  IF MN$=MN$(E) THEN BY=BY%(E):F=1:CD=E:E=256
14060  NEXT
14070  IF F=0 GOTO 14260
14080  ON BY GOSUB 14100,14130,14180
14090  GOTO 14020
14100  POKE AD,CD
14110  AD=AD+1
14120  RETURN
14130  IF OP>255 OR OP<0 THEN PRINT "ERROR":RETURN
14140  POKE AD,CD
14150  POKE AD+1,OP
14160  AD=AD+2
14170  RETURN
14180  IF OP>65535 OR OP<0 THEN PRINT "ERROR":RETURN
14190  POKE AD,CD
14200  B2=INT(OP/256)
14210  B1=OP-(B2*256)
14220  POKE AD+1,B1
14230  POKE AD+2,B2
14240  AD=AD+3
14250  RETURN
14260  IF MN$="ORG" OR MN$="END" OR MN$="DC" GOTO 14280
14270  PRINT "ERROR":GOTO 14020
14280  IF MN$="ORG" GOTO 14300
14290  GOTO 14340
14300  IF FO=1 THEN PRINT "ERROR":GOTO 14020
14310  FO=1
14320  AD=OP:ZZ=OP
14330  GOTO 14020
```

```
14340  IF MN$="END" GOTO 14360
14350  GOTO 14480
14360  EN=AD-1
14370  RETURN
14480  POKE AD,OP
14510  AD=AD+1
14520  GOTO 14020
15000  INPUT A$
15010  IF LEN(A$)<3 THEN PRINT "ERROR":GOTO 15000
15020  IF LEN(A$)=3 THEN MN$ A$:OP=0:RETURN
15030  S=0:FOR M=1 TO LEN(A$)
15040  IF MID$(A$,M,1)=" " THEN S=M:M=LEN(A$)
15050  NEXT
15060  IF S=0 THEN MN$=A$:RETURN
15070  MN$=LEFT$(A$,S-1)
15080  OP=VAL(RIGHT$(A$,LEN(A$)-S))
15090  RETURN
20000  PRINT CHR$(147):SZ=0
20010  INPUT "ENTER PROGRAM NAME";N$
20020  OPEN 1,1,1,N$
20030  PRINT#1,ZZ:DA%=ZZ:GOSUB 20110
20040  PRINT#1,EN:DA%=EN:GOSUB 20110
20050  FOR AD=ZZ TO EN
20060  DA%=PEEK(AD)
20070  PRINT#1,DA%:GOSUB 20110
20080  NEXT
20090  CLOSE 1
20100  RETURN
20110  SZ=LEN(STR$(DA%))+SZ+1
20120  IF SZ<192 THEN RETURN
20130  POKE 59411,53
20140  T=TI
20150  IF (TI-T)<6 GOTO 20150
20160  POKE 59411,61
20170  SZ=SZ-191
20180  RETURN
```

A BASIC 6502 DISASSEMBLER FOR APPLE AND PET

Michael J. McCann
28 Ravenswood Terrace
Cheektowaga, NY 14225

A disassembler is a program that accepts machine language (object code) as input and produces a symbolic representation that resembles an assembler listing. Although disassemblers have a major disadvantage viz., that they cannot reproduce the labels used by the original programmer, they can prove very useful when one is attempting to transplant machine code programs from one 6502 system to another. This article describes a disassembler program written in Commodore BASIC.

The disassembler (see listing and sample run) uses the mnemonics listed in the "Best of MICRO", on page 176A. The output is in this format: (address) (byte#1) (byte#2) (byte#3) (mnemonic) (bytes #2 and #3)

The address is outputted in decimal (base 10). The contents of the byte(s) making up each instruction are printed in hexadecimal (base 16) between the address and the mnemonic. In three byte instructions the high order byte is multiplied by 256 and added to the contents of the low order byte, giving the decimal equivalent of the absolute address. This number is printed in the (bytes #2 and #3) field. In two byte instructions the decimal equivalent of the second byte is printed in the (bytes #2 and #3) field.

Programming Comments

Lines 10-40 initialize the BY% and MN$ arrays (BY% contains the number of bytes in each instruction and MN$ contains the mnemonic of each instruction)

Lines 60-80 initialize the decimal to hexadecimal conversion array (CO$)

Lines 100-130 input the starting address

Lines 1000-1050 decimal to hexadecimal conversion subroutine

Lines 3000-5030 do the disassembly

Lines 3010-3030 take care of illegal operation codes

Line 3050 transfers control to one of three disassembly routines, the choice is determined by the number of bytes in the instruction

Lines 6000-6290 contain the data for the arrays

Although this was originally written in Commodore BASIC, it will work with the APPLESOFT BASIC of the APPLE computer.

SAMPLE RUN

```
RUN

START ADDRESS

? 64004

64004  4C 7E E6 JMP  59006

64007  AD 0A 02 LDA  522

64010  F0 08    BEQ  8

64012  30 04    BMI  4
```

```
  1 REM A 6502 DISASSEMBLER
  2 REM BY MICHAEL J. MCCANN
  3 REM WILL RUN ON AN 8K PET OR AN APPLE WITH APPLESOFT BASIC
  10 DIM MN$(256)BY%(256),CO$(16)
  20 FOR E=0 TO 255
  30 READ MN$(E),BY%(E)
  40 NEXT E
  60 FOR E=0 TO 15
  70 READ CO$(E)
  80 NEXT E
  100 PRINT CHR$(147)
  110 PRINT:PRINT "START ADDRESS"
  120 INPUT AD
* 130 PRINT
  140 I=0
  150 GOTO 3000
  1000 SX=INT(DC/16)
  1010 UN=DC-(SX*16)
  1020 SX$=CO$(SX)
  1030 UN$=CO$(UN)
  1040 HX$=SX$+UN$
  1050 RETURN
  3000 IF I=16 THEN 5050
  3005 I=I+1
  3010 IB=PEEK(AD)
  3015 IF MN$(IB)<>"NULL" GOTO 3050
  3020 DC=IB:GOSUB 1000
  3030 PRINT AD;TAB(8);HX$;"*"
  3035 AD=AD+1
  3040 GOTO 5030
  3050 ON BY%(IB) GOTO 3060,3090,4050
  3060 DC=IB:GOSUB 1000
  3070 PRINT AD;TAB(8);HX$;TAB(17);MN$(IB)
  3075 AD=AD+1
  3080 GOTO 5030
  3090 DC=IB:GOSUB 1000
  4000 B1$=HX$
  4010 DC=PEEK(AD+1):GOSUB 1000
  4020 B2$=HX$
  4030 PRINT AD;TAB(8);B1$;" ";B2$;TAB(17);MN$(IB);TAB(21);PEEK(AD+1)
  4035 AD=AD+2
  4040 GOTO 5030
  4050 DC=IB:GOSUB 1000
  4060 B1$=HX$
  4070 DC=PEEK(AD+1):GOSUB 1000
  4080 B2$=HX$
  4090 DC=PEEK(AD+2):GOSUB 1000
  5000 B3$=HX$
  5010 OP=PEEK(AD+1)+(PEEK(AD+2)*256)
  5020 PRINT AD;TAB(8);B1$;" ";B2$;" ";B3$;TAB(17);MN$(IB);TAB(21);OP
  5025 AD=AD+3
  5030 GOTO 3000
  5050 INPUT A
* 5060 PRINT
  5070 I=0
  5080 GOTO 3000
```

Note: The two PRINT statements with an * are required by APPLESOFT to prevent the first output line from being mis-aligned. They may not be required by the PET BASIC.

```
6000 DATA BRK,1,ORAIX,2,NULL,0,NULL,0,NULL,0,ORAZ,2,ASLZ,2,NULL,0,PHP,1
6010 DATA ORAIM,2,ASLA,1,NULL,0,NULL,0,ORA,3,ASL,3,NULL,0,BPL,2,ORAIY,2
6020 DATA NULL,0,NULL,0,NULL,0,ORAZX,2,ASLZX,2,NULL,0,CLC,1,ORAY,3
6030 DATA NULL,0,NULL,0,NULL,0,ORAX,3,ASLX,3,NULL,0,JSR,3,ANDIX,2,NULL,0
6040 DATA NULL,0,BITZ,2,ANDZ,2,ROLZ,2,NULL,0,PLP,1,ANDIM,2,ROLA,1,NULL,0
6050 DATA BIT,3,AND,3,ROL,3,NULL,0,BMI,2,ANDIY,2,NULL,0,NULL,0,NULL,0
6060 DATA ANDZX,2,ROLZX,2,NULL,0,SEC,1,ANDY,3,NULL,0,NULL,0,NULL,0,ANDX,3
6070 DATA ROLX,3,NULL,0,RTI,1,EORIX,2,NULL,0,NULL,0,NULL,0,EORZ,2,LSRZ,2
6080 DATA NULL,0,PHA,1,EORIM,2,LSRA,1,NULL,0,JMP,3,EOR,3,LSR,3,NULL,0
6090 DATA BVC,2,EORIY,2,NULL,0,NULL,0,NULL,0,EORZX,2,LSRZX,2,NULL,0
6100 DATA CLI,1,EORY,3,NULL,0,NULL,0,NULL,0,EORX,3,LSRX,3,NULL,0,RTS,1
6110 DATA ADCIX,2,NULL,0,NULL,0,NULL,0,ADCZ,2,RORZ,2,NULL,0,PLA,1,ADCIM,2
6120 DATA RORA,1,NULL,0,JMPI,3,ADC,3,ROR,3,NULL,0,BVS,2,ADCIY,2,NULL,0
6130 DATA NULL,0,NULL,0,ADCZX,2,RORZX,2,NULL,0,SEI,1,ADCY,3,NULL,0,NULL,0
6140 DATA NULL,0,ADCX,3,RORX,3,NULL,0,NULL,0,STAIX,2,NULL,0,NULL,0,STYZ,2
6150 DATA STAZ,2,STXZ,2,NULL,0,DEY,1,NULL,0,TXA,1,NULL,0,STY,3,STA,3
6160 DATA STX,3,NULL,0,BCC,2,STAIY,2,NULL,0,NULL,0,STYZX,2,STAZX,2,STXZY,2
6170 DATA NULL,0,TYA,1,STAY,3,TXS,1,NULL,0,NULL,0,STAX,3,NULL,0,NULL,0
6180 DATA LDYIM,2,LDAIX,2,LDXIM,2,NULL,0,LDYZ,2,LDAZ,2,LDXZ,2,NULL,0
6190 DATA TAY,1,LDAIM,2,TAX,1,NULL,0,LDY,3,LDA,3,LDX,3,NULL,0,BCS,2
6200 DATA LDAIY,2,NULL,0,NULL,0,LDYZX,2,LDAZX,2,LDXZY,2,NULL,0,CLV,1
6210 DATA LDAY,3,TSX,1,NULL,0,LDYX,3,LDAX,3,LDXY,3,NULL,0,CPYIM,2,CMPIX,2
6220 DATA NULL,0,NULL,0,CPYZ,2,CMPZ,2,DECZ,2,NULL,0,INY,1,CMPIM,2,DEX,1
6230 DATA NULL,0,CPY,3,CMP,3,DEC,3,NULL,0,BNE,2,CMPIY,2,NULL,0,NULL,0
6240 DATA NULL,0,CMPZX,2,DECZX,2,NULL,0,CLD,1,CMPY,3,NULL,0,NULL,0,NULL,0
6250 DATA CMPX,3,DECX,3,NULL,0,CPXIM,2,SBCIX,2,NULL,0,NULL,0,CPXZ,2,SBCZ,2
6260 DATA INCZ,2,NULL,0,INX,1,SBCIM,2,NOP,1,NULL,0,CPX,3,SBC,3,INC,3
6270 DATA NULL,0,BEQ,2,SBCIY,2,NULL,0,NULL,0,NULL,0,SBCZX,2,INCZX,2,NULL,0,SED,1
6280 DATA SBCY,3,NULL,0,NULL,0,NULL,0,SBCX,3,INCX,3,NULL,0
6290 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
```

Apple II

Apple Computer Inc.
10260 Bandley Drive
Cupertino, CA 95014

* includes a perforated "tear-out" reference card

INSIDE THE APPLE II

Arthur Ferruzzi
69 Hauman Street
Revere, MA 02151

If you've seen the colorful Apple II ads, you know that this is a fun machine. However, with a typical system price roughly equal to a trip to Europe, I suspect that the real computer nuts are going to let the wife and kids "byte" into the fun features while they make use of the Apple's extensive software and hardware power. Let's look at the Apple II as a complete, yet easily expandable, system on a board.

There's Memory

There are three rows of RAM memory sockets, each of which will hold 4K or 16K bytes of dynamic RAM for a maximum of 48K bytes of RAM. There are six sockets each of which will hold 2K of ROM. Four ROMs are supplied, containing a 6K integer BASIC and a 2K System Monitor. The full feature System Monitor supports commands that examine and deposit data into memory, move and compare blocks of memory, load and store blocks on cassette, assemble and disassemble op codes, run trace and single step programs, display and modify 6502 registers, and perform hexadecimal arithmetic. In addition, the monitor is chock full of user accessible subroutines including a floating point package and simulated 16 bit microprocessor.

One of the more interesting possibilities for the Apple II is expansion of the software in ROM. The 8K supplied is in 2K byte 9316B ROM chips. A check of the pinouts shows that these ROMs are nearly identical to the 2716/2708 EPROM. Check the manual at your nearest Apple dealer before you burn in your favorite version of PASCAL, APL, or ANIMATION.

There's Video

The second 1K block of memory is shared by the processor (during phase 2 of the clock) and the display (during phase 1 of the clock, when it is also refreshing every dynamic RAM chip on the board). Thus, the display is fast, and it is colorful. Options are 24 lines of 40 upper case characters, or 40 x 40 graphics in 15 colors plus four lines of text, or 40 x 48 graphics in 15 colors. Colors, point plotting and line plotting are also accessible from BASIC. With an 8K chunk of memory, you have high resolution 280 horizontal by 192 vertical graphics in four colors, or 32 fewer vertical lines with four lines of text at the bottom of the screen. The speed of the video display and the 6502 itself as well as the machine language subroutines in the monitor and available on cassette tape make the 8K graphics extremely useful.

There's I/O

First, of course, is a full typewriter style ASCII keyboard and a 1500 bits per second cassette interface. But these are only the obvious I/O devices. In addition there are four 8 bit analog inputs which measure resistance by timing a variable delay generator. Normally set up as four game paddles, you might set them up as two joysticks to control the parameters in an interactive system which provides feedback via the display. Before considering some of the I/O possibilities, it is worth noting that the Apple II has a lot of address decoding done directly on the board. For example, writing to the eight addresses C058--C05F sets or clears four TTL output lines. Reading from addresses C061--C063 tests three switch inputs. Further, keyboard input and strobe, cassette input and output, speaker output, paddle timers, and eight "software switches" which set the graphics and text modes, are all accessible by reading and writing specific memory locations. Thus, all ports may be set or tested by user programs including BASIC (peek and poke).

If you want to expand the system, there are eight peripheral connectors on a 50 pin (x .10") fully buffered bus. Accessing any address in the range C800--CFFF sends an I/O strobe to pin 20 on all cards (0 through 7). Accessing addresses C1xx--C7xx sends an I/O select pulse to pin 1 on the appropriate cards (1 through 7). Accessing addresses C08x--C0Fx sends a device select pulse to pin 41 on the appropriate card (0 through 7). Thus, the 8 peripheral cards are fully decoded, saving the overhead of address decoding logic. Provision is made for daisy-chained interrupt and DMA. Presumably Apple will be supplying low cost peripherals making use of these features.

There's the Power Supply

While the peripheral bus makes it easy to design custom I/O devices, the major limitation appears to be the power supply. It is a switching type supply where the AC is rectified and sent to an oscillator whose frequency varies with the load so that the +5 volt DC line is never more than 3% off. The other voltages basically respond to the loading on the +5 volt line and are not as well regulated. I/O cards should draw less than 1.5 watts and the power cord MUST BE VERY WELL GROUNDED.

The Bottom Line

In terms of serious applications, here is what you get: a fast 6502 microprocessor with all of its inherent features, 4 to 48K of RAM, a fast 6K integer BASIC in ROM, a classy 2K System Monitor in ROM, a "transparent" 1K color video with graphics, 8K color graphics, an ASCII keyboard with interface, four analog inputs, a fast cassette interface, three digital I/O bits, an audio "beeper", eight decoded peripheral connectors, a stylish package, and a (small) power supply.

Just add up what all that good stuff costs separately. Then, if your application falls within or near these specifications, the Apple II will be a better buy than a homebrew system. In shopping for a computer, remember to try before you buy, from a reputable dealer. The Apple II is up and running at many computer stores across the country. (For the dealer nearest you see the dealer list on page 16 of the October 1977 issue of BYTE, or write to Apple Computer Inc., 20863 Stevens Creek Boulevard, Cupertino, CA 95014.) With a little savvy and careful picking, we can have the fine products we want and put the squeeze on the lemons.

A WORM IN THE APPLE?

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

There may be a serious problem hidden deep within the Apple II according to John Conway and Jack Hemenway of EDN magazine. As part of their system design project based on a bare-board Apple - "Project Indecomp" - they tried to interface a 6820 PIA to the Apple, and uncovered a potentially serious problem. The normal way to operate a 6502 based system is to provide an external clock [phase 0] to the 6502 which then generates two non-overlapping clock signal [phase 1 and phase 2] which are used to control all system timing. For some reason, the design of the Apple II violated this basic clock scheme and uses the phase 0 external clock instead of the 6502 generated phase 2 clock. While these two clocks are very similar, they are not identical. Phase 1 and phase 0 have an overlap of about 50 nanoseconds. For many parts of the system this is not important, as indicated by the fact that the Apple II works. For other devices, however, such as the 6820 PIA, this difference is critical to the extent that the device simply will not work. A report in EDN scheduled for 20 May will cover this problem in detail, and we will try to get more info for the next issue of MICRO. Is the problem serious? Critical? Fatal? It is probably too early to judge the effect of this problem. It may not have an adverse effect in many systems. It may be possible to correct. Or it may be a very serious system problem.

HALF A WORM IN THE APPLE

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Last issue we reported a potential problem that had been discovered in the Apple II, relating to using PIA'a. The problem had been uncovered by the staff of EDN in the course of developing a system based on an Apple II board. The matter is not totally resolved, but the following is what we have heard.

I called Steve Wozniak of Apple and asked about the problem. He said that he had sent a chip to EDN which had cleared up the problem. He did not indicate that there was any more to it.

I then talked to John Conway of EDN. He maintained that a problem still does exist with Apple II interfacing to 6520 or 6522 PIAs. It can be done, but requires the addition of a chip to slow down the phase 0 signal to make it the equivalent of the phase 2 signal. The PIA can not be directly interfaced, as would normally be expected in a 6502-based system. John stated that the chip required costs about $7.00.

Another angle on the picture was also reported to me by John. He had found a company on the West Coast that is making interfaces for the Apple II. The engineer there had discovered the same problem.

There is a fairly complete discussion of the problem and the solution in the May 20, 1978 edition of EDN. If anyone has additional information to shed on the situation, MICRO will be happy to publish it. The problem does not seem to be all that serious, and we do not want to dwell on it, but we hope that this discussion has prevented some of our readers from going nuts trying to add a PIA to their Apple II.

APPLE PI

Robert J. Bishop
1143 W. Badillo, Apt E
Covina, CA 91722

Everyone knows that the value of Pi is about 3.1416. In fact, its value was known this accurately as far back as 150 A.D. But it wasn't until the sixteenth century that Francisco Vieta succeeded in calculating Pi to ten decimal places.

Around the end of the sixteenth century the German mathematician, Ludolph von Ceulen, worked on calculating the value of Pi until he died at the age of 70. His efforts produced Pi to 35 decimal places.

During the next several centuries a great deal of effort was spent in computing the value of PI to evern greater precision. In 1699 Abraham Sharp calculated Pi to 71 decimal places. By the mid 1800's its value was known to several hundred decimal places. Finally, in 1873, an English mathematician, Shanks, determined Pi to 707 decimal places, an accuracy which remained unchallenged for many years.

I was recently rereading my old copy of Kasner & Newman"s Mathematics and the Imagination

I was recently rereading my old copy of Kasner & Newman's Mathematics and Imagination (Simon & Schuster, 1940), where I found the series expansion:

$$\pi = \sum_{k=1}^{\infty} \frac{16(-1)^{k+1}}{(2k-1)5^{2k-1}} - \sum_{k=1}^{\infty} \frac{4(-1)^{k+1}}{(2k-1)239^{2k-1}}$$

The book indicated that this series converged rather quickly but "... it would require ten years of calculation to determine Pi to 1000 decimal places." Clearly this statement was made before modern digital computers were available. Since then, Pi has been computed to many thousands of decimal places. But Kasner & Newman's conjecture of a ten-year calculation for Pi aroused my curiousity to see just how long it would take my little Apple-II computer to perform the task.

Program Description

My program to compute the value of Pi is shown in Figure 1. It was written using the Apple II computer's Integer BASIC and requires a 16K system (2K for the program inself; 12K for data storage). The program is fairly straightforward but a brief discussion may be helpful.

The main calculation loop consists of lines 100 through 300; the results are printed in lines 400 through 600. The second half of the listing contains the multiple precision arithmetic subroutines. The division, addition, and subtraction routines start at lines 1000, 2000, and 3000, respectively.

In order to use memory more efficiently, PEEK and POKE statements were used for arrays instead of DIM statements. Three such arrays are used by the program: POWER, TERM, and RESULT. Each are up to 4K bytes long and start at the memory locations specified in line 50 of the program.

The three arrays mentioned above each store partial and intermediate results of the calculations. Each byte of an array contains either one or two digits, depending on the value of the variable, TEN. If the number of requested digits for Pi is less than about 200, it is possible to store two digits per byte; otherwise, each byte must contain no more than one digit. (The reason for this distinction occurs in line 1070 where an arithmetic overflow can occur when trying to evaluate higher order terms of the series if too many digits are packed into each byte.)

The program evaluates the series expansion for Pi until the next term of the series results in a value less than the requested precision. Line 1055 computes the variable, ZERO, which can be tested to see if an underflow in precision has occurred. This value is then passed back to the main program where, in line 270, it determines whether or not the next term of the series is needed.

Results

Figure 2 shows the calculated value of Pi to 1000 decimal places. Running the program to get these results took longer than it did to write the program! (The program ran for almost 40 hours before it spit out the answer.) However it took less than two minutes to produce Pi to 35 decimal places, the same accuracy to which Ludolph von Ceulen spent his whole life striving for!

Since the program is written entirely in BASIC it is understandably slow. By rewriting all or part of it in machine language its performance could be vastly improved. However, I will leave this implementation as an exercise for anyone who is interested in pursuing it.

Note: You must set HIMEM:4096.

Figure 1.

Program Listing

```
>LIST
    0 REM  *** APPLE-PI ***
      WRITTEN BY: BOB BISHOP
    5 CALL -936: VTAB 10: TAB 5: PRINT
      "HOW MANY DIGITS DO YOU WANT"
      ;
   10 INPUT SIZE
   15 CALL -936
   20 TEN=10: IF SIZE>200 THEN 50
   30 TEN=100: SIZE=(SIZE+1)/2
   50 POWER=4096: TERM=8192: RESULT=
      12288
   60 DIV=1000: ADD=2000: SUB=3000:
      INIT=4000: COPY=5000
   70 DIM CONSTANT(2): CONSTANT(1)
      =25: CONSTANT(2)=239
```

```
100 REM  MAIN LOOP
125 FOR PASS=1 TO 2
150 GOSUB INIT
200 GOSUB COPY
210 POINT=TERM: DIVIDE=EXP:  GOSUB
    DIV
220 IF SIGN>0 THEN GOSUB ADD
230 IF SIGN<0 THEN GOSUB SUB
240 EXP=EXP+2: SIGN=-SIGN
250 POINT=POWER: DIVIDE=CONSTANT(
    PASS):  GOSUB DIV
260 IF PASS=2 THEN GOSUB DIV
270 IF ZERO<>0 THEN 200
300 NEXT PASS
400 REM  PRINT THE RESULT
500 PRINT : PRINT
510 PRINT "THE VALUE OF PI TO "
    ;(TEN/100+1)*SIZE;" DECIMAL PLAC
    ES:": PRINT
520 PRINT PEEK (RESULT);".";
530 FOR PLACE=RESULT+1 TO RESULT+
    SIZE
540 IF TEN=10 THEN 570
560 IF PEEK (PLACE)<10 THEN PRINT
    "0";
570 PRINT PEEK (PLACE);
580 NEXT PLACE
590 PRINT
600 END
1000 REM  DIVISION SUBROUTINE
1010 DIGIT=0: ZERO=0
1020 FOR PLACE=POINT TO POINT+SIZE
1030 DIGIT=DIGIT+ PEEK (PLACE)
1040 QUOTIENT=DIGIT/DIVIDE
1050 RESIDUE=DIGIT MOD DIVIDE
1055 ZERO=ZERO OR (QUOTIENT+RESIDUE)

1060 POKE PLACE, QUOTIENT
1070 DIGIT=TEN*RESIDUE
1080 NEXT PLACE
1090 RETURN
2000 REM  ADDITION SUBROUTINE
2010 CARRY=0
2020 FOR PLACE=SIZE TO 0 STEP -1

2030 SUM= PEEK (RESULT+PLACE)+ PEEK
     (TERM+PLACE)+CARRY
2040 CARRY=0
2050 IF SUM<TEN THEN 2080
2060 SUM=SUM-TEN
2070 CARRY=1
2080 POKE RESULT+PLACE, SUM
2090 NEXT PLACE
2100 RETURN
3000 REM  SUBTRACTION SUBROUTINE
3010 LOAN=0
3020 FOR PLACE=SIZE TO 0 STEP -1
3030 DIFFERENCE= PEEK (RESULT+PLACE)
     - PEEK (TERM+PLACE)-LOAN
3040 LOAN=0
3050 IF DIFFERENCE>=0 THEN 3080
3060 DIFFERENCE=DIFFERENCE+TEN
3070 LOAN=1
3080 POKE RESULT+PLACE, DIFFERENCE
3090 NEXT PLACE
3100 RETURN
4000 REM   INITIALIZE REGISTERS
4010 FOR PLACE=0 TO SIZE
4020 POKE POWER+PLACE, 0
4030 POKE TERM+PLACE, 0
4040 IF PASS=1 THEN POKE RESULT+
     PLACE, 0
4050 NEXT PLACE
4060 POKE POWER, 16/PASS ↑ 2
4070 IF PASS=1 THEN DIVIDE=5
4080 IF PASS=2 THEN DIVIDE=239
4090 POINT=POWER:  GOSUB DIV
4100 EXP=1: SIGN=3-2*PASS
4110 RETURN
5000 REM  COPY "POWER" INTO "TERM"
5010 FOR PLACE=0 TO SIZE
5020 POKE TERM+PLACE,  PEEK (POWER+
     PLACE)
5030 NEXT PLACE
5040 RETURN
```

Even "Apple Pi" isn't simple any more! Neil D. Lipson of the Philadelphia Apple Users Group writes that "The Pi article by Bob Bishop (MICRO 6:15) is missing one thing. Add HIMEM:4096." But, that's not all! John Paladini writes that: "The value of Pi was not computed to 1000 decimal places, but rather 998. Such inaccuracies occur when computing a series where billions of calculations are required. My best guess is that in order to calculate Pi to 1,000 places using the given series one would have to compute to 1,004 places. The last two digits should read 89 not 96."

```
THE VALUE OF PI TO 1000 DECIMAL PLACES:

3. 14159265358979323846264338327950288419
7169399375105820974944592307816406286208
9986280348253421170679821480865132823066
4709384460955058223172535940812848111745
0284102701938521105559644622948954930381
9644288109756659334461284756482337867831
6527120190914564856692346034861045432664
8213393607260249141273724587006606315588
1748815209209628292540917153643678925903
6001133053054882046652138414695194151160
9433057270365759591953092186117381932611
7931051185480744623799627495673518857527
2489122793818301194912983367336244065664
3086021394946395224737190702179860943702
7705392171762931767523846748184676694051
3200056812714526356082778577134275778960
9173637178721468440901224953430146549585
3710507922796892589235420199561121290219
6086403441815981362977477130996051870721
1349999998372978049951059731732816096318
5950244594553469083026425223082533446850
3526193118817101000313783875288658753320
8381420617177669147303598253490428755468
7311595628638823537875937519577818577805
3217122680661300192787661119590921642019
96
```

Figure 2.

Pi to 1000 Decimal Places

THE APPLE II POWER SUPPLY REVISITED

Rod Holt
Chief Engineer
Apple Computer Inc.
20863 Stevens Creek Blvd., B3-C
Cupertino, CA 95014

Your review of the Apple II ("Inside the Apple II" by Arthur Ferruzzi, BEST of MICRO, p.83) was most gratifying. However, your comment about the "small" power supply invites a reply.

The power supply has no function other than running the Apple II and its peripherals, and as it does this very well, then what's "small"? Apple Computer is far enough along in peripheral card development to state categorically that with an EPROM card, a ROM card, a parallel printer card, a floppy disk card, and several more all plugged in, the power supply isn't even breathing hard.

We do recommend that users keep their designs to a reasonable minimum power. But the reason for this is the same as one of the reasons Apple designed a switching regulator in the first place: to keep temperature rises to a minimum. The general rule of thumb is that a 25 degree C increase in ambient will drop the mean time between failures by a factor of 10. For the user, the watts saved mean literally thousands of hours more of trouble free system operation. The switcher design cuts the input power nearly in half over conventional regulators and the overall temperature rise is reduced by approximately 25 C. And, of course, the use of low-power schottky and a tight and economic hardware design is key as well.

A second point needs to be made. It's quite common to have well over a thousand dollars in semiconducters in an Apple II system. The Apple switcher is designed to protect those semiconducters under all fault conditions (including possible failure modes internal to the power supply itself). Never has an Apple II been damaged by its own power supply. In contrast, Apple can document many cases of blown RAM and other IC's where customers have used homemade or "off the shelf" power supplies. See the sad story in EDN, November 20, 1977 page 232. There are many more such sad stories. The power supply manufacturers of the world are just beginning to see that a supply failure means much more than just an equipment shut-down nuisance. Thus it's important to know what happens when, for example, the +12 volt supply is shorted to the -5 volt supply. What happens to the +5 volts? With the Apple switcher, all supplies neatly go to zero, and they all recover smoothly when the short is removed.

I close by murmuring -

"Small is Beautiful".

PRINTING WITH THE APPLE II

C. R. (Chuck) Carpenter W5USJ
2228 Montclair Place
Carrollton, TX 75006

Hardcopy output from your Apple II is a practical reality. All you need is a TELPAR thermal printer, a simple one-transistor adapter circuit and a machine language printing routine. The printing routine slows the data rate down to 110 (or 300) baud and directs the data stream to AN0 (the game paddle connector - annunciator output, port zero). I have the TELPAR PS-40-3C (now PS-48) connected to my Apple II and I am printing everything from Biorhythms to Manpower Planning programs. Here are the details for hooking up the printer.

The TELPAR PS-40-3C

The PS-40 (Photo 1) is a 48 column thermal printer using 5.5 inch width paper. The model I have is a 3 chip F8 controlled unit. The current, more compact models use a single chip F8/3870. Inputs are provided for serial TTL, RS 232 and 20 MA current loop. You can also connect a parallel port to the printer and software controllable options are available. The printer can be used as the only I/O if a keyboard is connected as the parallel source. The paper is not too expensive at $3.00 per 164 foot roll.

Power supply voltages are critical and several are required. (This is the only shortcoming I found with this general purpose printer.) Good regulation is a must from your power supply. Especially the printhead supply voltage (16). Excessive positive deviations here can blow the printhead. Telpar can supply a switching type power supply that will do the job. The connections to the 56 pin edge connector are shown in Figure 1. The connector actually has numbers and letters to designate pins. Somewhere along the line, numbers were assigned to both sides. Be sure you transpose the numbers correctly and connect it to the circuit board properly. Telpar has good repair service, but it still takes time.

Interface Adapter

All that is needed to connect the Apple II to an RS 232 printer input is the adapter circuit shown in Figure 2 (from an Apple application note). I built this circuit on a 16 pin IC header and plugged it in. There is some inconvenience if you want to use the game paddles too, but I think there is a way around this if you choose to do some rewiring.

You can get the -12 volts for this circuit from the main power connector. A short lead and a small connector pin will work. If the pin is small enough, it will slide down inside the -12 volt terminal on the power connector. There are other places like the keyboard where -12 volts is also available. Use caution making this connection.

Making it Print

Now the only part left is a way to get the data slowed down and directed to the AN0 output port. Apple has taken care of this detail with the routine shown in Figure 3. You can key in this routine and save it on tape. Each time you have a printing task the program is easily loaded using Apple's system monitor commands. I've used it with machine language programs and both forms of BASIC: Apple's Integer BASIC and Applesoft Floating Point 8K BASIC. The routine is called as follows:

$380G and RETURN in machine language

CALL 896 in Apple Integer BASIC

X=USR(896) in Applesoft 8K BASIC

Note: A line number is not needed to call the print routine. (380 hex = 896 decimal).

Using RESET will stop the print routine in machine language and in Apple Integer BASIC (return to BASIC with the soft entry CONTROL-C). With Applesoft in RAM, exiting via RESET and re-entry the soft way with 0G works sometimes but usually causes a glitch in BASIC and messes up the program. I avoid this problem by waiting to do any printing until the last thing. Any further changes are made at the slower speed. I would speculate that things like this will clear up when Applesoft is in ROM. I'm still looking for a way to get out of the print routine directly from the BASIC program.

The Tale is Told

As I indicated at the beginning, I'm printing most anything I want to. The 5.5 inch paper width presents some limitations but most programs can be formatted to work okay. There are several features and details I've alluded to but an article to do them justice would take several issues of MICRO to cover. Telpar has a technical paper that describes them and would be happy to send you one. For a simple, effective, general purpose printer, I have not found a better choice than my Telpar thermal. I think you would find it a good choice too.

For more info, write to:

Telpar Inc.
4132 Billy Mitchell Road
P.O. Box 796
Addison, TX 75001

[Editor's Note: One problem I have found with this thermal printer is that the print is light blue. This can cause great difficulty if you want to copy the output since most xerox-type copiers and many plate-making films are "blind" to light blue.]

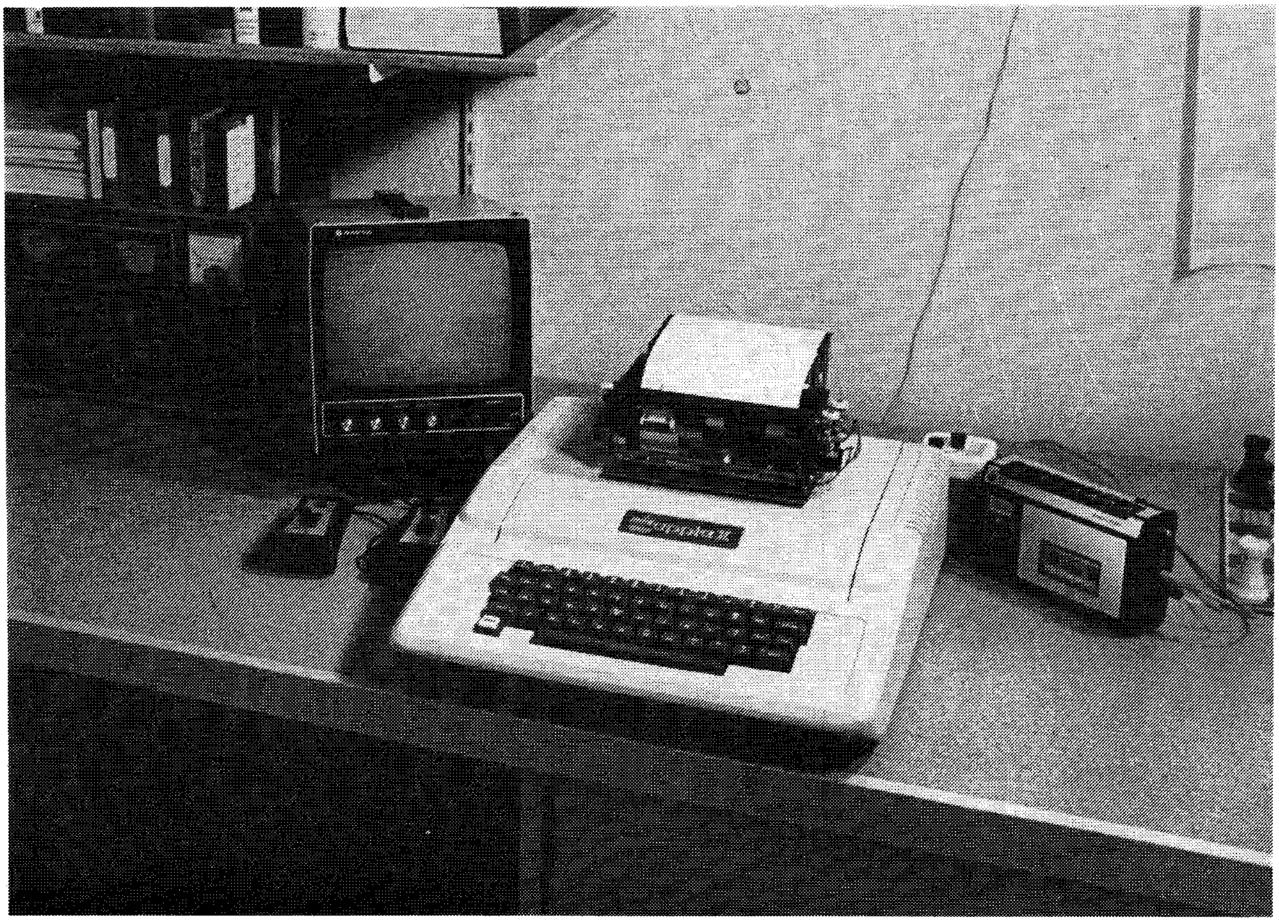


Photo 1 (by Jim Chamberlain)

APPLE II and TELPAR Thermal Printer

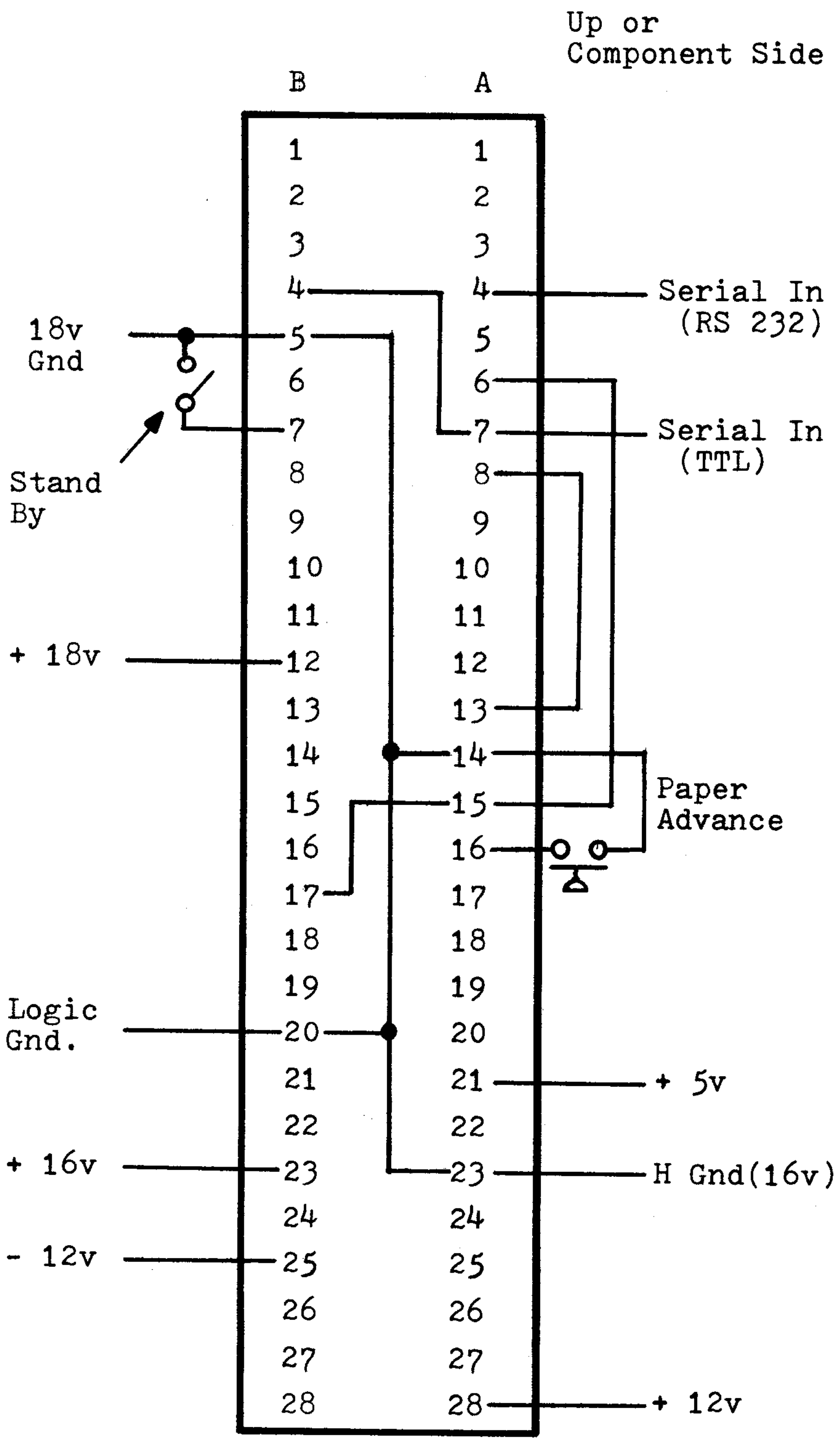


Figure 1

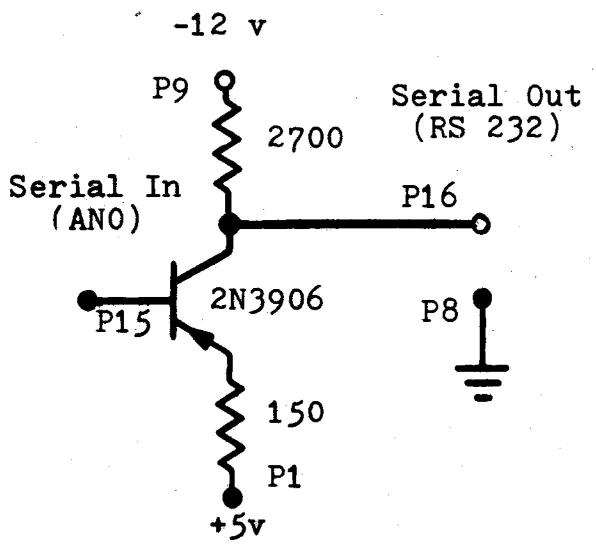
PS-40 Connector Diagram:
Input and Power Connections

```
*380LLL

0380-   A9 89       LDA   #$89
0382-   85 36       STA   $36
0384-   A9 03       LDA   #$03
0386-   85 37       STA   $37
0388-   60          RTS
0389-   84 35       STY   $35
038B-   48          PHA
038C-   20 A5 03    JSR   $03A5
038F-   68          PLA
0390-   C9 8D       CMP   #$8D
0392-   D0 0C       BNE   $03A0
0394-   A9 8A       LDA   #$8A
0396-   20 A5 03    JSR   $03A5
0399-   A9 58       LDA   #$58
039B-   20 A8 FC    JSR   $FCA8
039E-   A9 8D       LDA   #$8D
03A0-   A4 35       LDY   $35
03A2-   4C F0 FD    JMP   $FDF0
03A5-   A0 0B       LDY   #$0B
03A7-   18          CLC
03A8-   48          PHA
03A9-   B0 05       BCS   $03B0
03AB-   AD 58 C0    LDA   $C058
03AE-   90 03       BCC   $03B3
03B0-   AD 59 C0    LDA   $C059
03B3-   A9 D3       LDA   #$D3
03B5-   48          PHA
03B6-   A9 20       LDA   #$20
03B8-   4A          LSR
03B9-   90 FD       BCC   $03B8
03BB-   68          PLA
03BC-   E9 01       SBC   #$01
03BE-   D0 F5       BNE   $03B5
03C0-   68          PLA
03C1-   6A          ROR
03C2-   88          DEY
03C3-   D0 E3       BNE   $03A8
03C5-   60          RTS
03C6-   00          BRK
03C7-   00          BRK

*
```

Apple II AN0 output routine in machine language to provide serial data output at 110 and 300 baud. Change location $3B4 to $4D for 300 baud.



Resistors are in Ohms, 1/2 Watt, 5%
P No's refer to game connector pins - P9 and P16 are used a tie points.

Figure 2

Single Transistor Adapter Circuit and Interface

```
*380.3C7

0380- A9 89 85 36 A9 03 85 37
0388- 60 84 35 48 20 A5 03 68
0390- C9 8D D0 0C A9 8A 20 A5
0398- 03 A9 58 20 A8 FC A9 8D
03A0- A4 35 4C F0 FD A0 0B 18
03A8- 48 B0 05 AD 58 C0 90 03
03B0- AD 59 C0 A9 D3 48 A9 20
03B8- 4A 90 FD 68 E9 01 D0 F5
03C0- 68 6A 88 D0 E3 60 00 00
*
```

[Note: This listing and dump were made on the Telpar printer.]

Figure 3

Machine Language Print Routine and HEX Dump

APPLE II PRINTING UPDATE

C. R. (Chuck) Carpenter W5USJ
2228 Montclair Place
Carrollton, TX 75006

"Printing with the Apple II" [Page #88] included information that has been revised. Since the article was written, I've improved some things and I'd like to pass them along.

The Adapter Didn't

After using the adapter circuit for a couple of months, I took a good look at what was happening. The conclusion was nothing! Initially, it didn't work when I connected it to the RS-232 receiver on the PS-40. I connected it to the serial TTL input (pin A7) and it worked. The voltage swing wasn't excessive (clamped with some diodes), so I left it hooked-up. Should have been a clue. But at the time I didn't see it, and anyway, it worked.

During one of our (infrequent) snowed-in days here in Texas, I had time to think about it. There wasn't any apparent reason not to hook it up directly; and I did. It worked the way it should so I had a no-interface-required computer to printer system. When I received my new Apple Operator's Manual I noticed a new interface circuit, not the one I used as originally provided.

All that is needed is to connect a signal lead and ground from the Apple to the printer. The signal lead connects to Pin 15 of Apple's game paddle connector. Also to Pin A7, TTL serial data in, on the printer. I soldered the game paddle connector to the 16 pin header. No other connections needed.

Now You Can Start and Stop

Ted Spradley, a programmer/engineer at work, helped me with the machine language print program. His analysis suggested restoring the page zero registers to make the print routine stop. As you more experienced programmers would know, it worked. I rewrote the program to store and restore the page zero data and now the routine turns on and off under program control. The program, shown in Figure 1, was a revelation to me. Again, my thanks to Ted for his assistance.

The Blues Are Gone

Most of my programs are printed on the paper that turns blue (and fades). Telpar has a black on off-white paper now. This new paper makes a much sharper copy too. The blue paper was also susceptible to smearing. This did not help the copy quality either, photographically or Xerographically.

There! Now that the problems are resolved, what's holding you back? Let's get printing.

Author's Note: Even if you don't have a printer, the print routine is useful. Use it to slow the screen speed down. This way you can read a listing during a slow scroll.

Getting Decimal Values From Hex Data

For some other program, POKE was used to enter machine language from BASIC. I did this for the print routine. All the HEX values have to be converted to decimal. At first I did this with the TI Programmer. Then I "discovered" what PEEK is all about. A BASIC program to print the decimal values simplifies the job. Convert the first and last addresses (to do a range of addresses) to their decimal values These values are 875 and 967 for the print program. Then use them in a FOR-NEXT routine like this:

```
100 FOR I=875 TO 967:PRINT PEEK(I);:
    PRINT" ";:NEXT I:END
```

This reduced a two hour job to about ten minutes. Hooray for progress.

Listing

```
*36BLLL

036B-   A5 36       LDA   $36
036D-   8D C6 03    STA   $03C6
0370-   A5 37       LDA   $37
0372-   8D C7 03    STA   $03C7
0375-   A9 89       LDA   #$89
0377-   85 36       STA   $36
0379-   A9 03       LDA   #$03
037B-   85 37       STA   $37
037D-   60          RTS
037E-   AD C6 03    LDA   $03C6
0381-   85 36       STA   $36
0383-   AD C7 03    LDA   $03C7
0386-   85 37       STA   $37
0388-   60          RTS
0389-   84 35       STY   $35
038B-   48          PHA
038C-   20 A5 03    JSR   $03A5
038F-   68          PLA
0390-   C9 8D       CMP   #$8D
0392-   D0 0C       BNE   $03A0
0394-   A9 8A       LDA   #$8A
0396-   20 A5 03    JSR   $03A5
0399-   A9 58       LDA   #$58
039B-   20 A8 FC    JSR   $FCA8
039E-   A9 8D       LDA   #$8D
03A0-   A4 35       LDY   $35
03A2-   4C F0 FD    JMP   $FDF0
03A5-   A0 0B       LDY   #$0B
03A7-   18          CLC
03A8-   48          PHA
03A9-   B0 05       BCS   $03B0
03AB-   AD 58 C0    LDA   $C058
03AE-   90 03       BCC   $03B3
03B0-   AD 59 C0    LDA   $C059
03B3-   A9 D3       LDA   #$D3
03B5-   48          PHA
03B6-   A9 20       LDA   #$20
03B8-   4A          LSR
03B9-   90 FD       BCC   $03B8
03BB-   68          PLA
03BC-   E9 01       SBC   #$01
03BE-   D0 F5       BNE   $03B5
03C0-   68          PLA
03C1-   6A          ROR
03C2-   88          DEY
03C3-   D0 E3       BNE   $03A8
03C5-   60          RTS
03C6-   F0 FD       BEQ   $03C5
```

HEX Dump

```
*36B.3C7

036B- A5 36 8D C6 03
0370- A5 37 8D C7 03 A9 89 85
0378- 36 A9 03 85 37 60 AD C6
0380- 03 85 36 AD C7 03 85 37
0388- 60 84 35 48 20 A5 03 68
0390- C9 8D D0 0C A9 8A 20 A5
0398- 03 A9 58 20 A8 FC A9 8D
03A0- A4 35 4C F0 FD A0 0B 18
03A8- 48 B0 05 AD 58 C0 90 03
03B0- AD 59 C0 A9 D3 48 A9 20
03B8- 4A 90 FD 68 E9 01 D0 F5
03C0- 68 6A 88 D0 E3 60 F0 FD
*
```

Print Routine

START Print	STOP Print
*36BG	*37EG
>CALL 875	>CALL 894
]SP=USR(875)	]EP=USR(894)

Type in one of above and then type RETURN to activate the command.

* = from Apple Monitor
> = from Integer BASIC
] = from Applesoft BASIC

Change 03B4 to 4D for 300 baud.

Figure 1

Listing and HEX Dump of Machine Language Print Routine

A SLOW LIST FOR APPLE BASIC

Bob Sander-Cederlof
8413 Midpark Road No. 3072
Dallas, TX 75240

One of the nicest things about Apple BASIC is its speed. It runs circles around most other hobby systems! Yet there are times when I honestly wish it were a little slower.

Have you ever typed in a huge program, and then wanted to review it for errors? You type "LIST", and the whole thing flashes past your eyes in a few seconds! That's no good, so you list it piecemeal -- painfully typing in a long series like:

```
LIST 0,99
LIST 100,250
   .
   .
   .
LIST 21250,21399
```

As the reviewing and editing process continues, you have to type these over and over and over . . . Ouch!

At the March meeting of the Dallas area "Apple Corps" several members expressed the desire to be able to list long programs slowly enough to read, without the extra effort of typing separate commands for each screen-full. One member suggested appending the series of LIST commands to the program itself, with a subroutine to wait for a carriage return before proceeding from one screen-full to the next. For example:

```
9000 LIST 0,99:GOSUB 9500
9010 LIST 100,250: GOSUB 9500
        .
        .
        .
9250 LIST 21250,21399:GOSUB 9500
9260 END
9500 INPUT A$:RETURN
```

While this method will indeed work, it is time-consuming to figure out what line ranges to use in each LIST command. It is also necessary to keep them up-to-date after adding new lines or deleting old ones.

But there is a better way! I wrote a small machine language program which solves our problem. After this little 64-byte routine is loaded and activated the LIST command has all the features we wanted.

1. The listing proceeds at a more leisurely pace, allowing you to see what is going by.
2. The listing can be stopped temporarily, by merely pressing the space bar. When you are ready, pressing the space bar a second time will cause the listing to resume.
3. The listing can be aborted before it is finished, by typing a carriage return.

The routine as it is now coded resides in page three of memory, from $0340 to $037F. It is loaded from cassette tape in the usual way: *340.37FR.

After the routine is loaded, you return to BASIC. The slow-list features are activated by typing "CALL 887". They may be de-activated by typing "CALL 878" or by hitting the RESET key.

How does it work? The commented assembly listing should be self-explanatory, with the exception of the tie-in to the Apple firmware. All character output in the Apple funnels through the same subroutine: COUT, at location $FDED. The instruction at $FDED is JMP ($0036) This means that the address which is stored in locations $0036 and $0037 indicates where the character output subroutine really is. Every time you hit the RESET key, the firmware monitor sets up those two locations to point to $FDF0, which is where the rest of the COUT subroutine is located. If characters are supposed to go to some other peripheral device, you would patch in the address of your device handler at these same two locations. In the case of the slow-list program, the activation routine merely patches locations $0036 and $0037 to point to $0340. The de-activation routine makes them point to $FDF0 again.

Every time slow-list detects a carriage return being output, it calls a delay subroutine in the firmware at $FCA8. This has the effect of slowing down the listing. Slow-list also keeps looking at the keyboard strobe, to see if you have typed a space or a carriage return. If you have typed a carriage return, slow-list stops the listing and jumps back into BASIC at the soft entry point ($E003). If you have typed a space, slow-list goes into a loop waiting for you to type another character before resuming the listing.

That is all there is to it! Now go turn on your Apple, type in the slow-list program, and list to your heart's content!

```
0340                      ORG   $0340

                     ROUTINE TO SLOW DOWN APPLE BASIC LISTINGS

0340 C9 8D     SLOW   CMPIM $8D    CHECK IF CHAR IS CARRIAGE RETURN
0342 D0 1A            BNE   CHROUT NO, SO GO BACK TO COUT
0344 48               PHA          SAVE CHARACTER ON STACK
0345 2C 00 C0         BIT   $C000  TEST KEYBOARD STROBE
0348 10 0E            BPL   WAIT   NOTHING TYPED YET
034A AD 00 C0         LDA   $C000  GET CHARACTER FROM KEYBOARD
034D 2C 10 C0         BIT   $C010  CLEAR KEYBOARD STROBE
0350 C9 A0            CMPIM $A0    CHECK IF CHAR IS A SPACE
0352 F0 10            BEQ   STOP   YES - STOP LISTING
0354 C9 8D            CMPIM $8D    CHECK IF CHAR IS A CARRIAGE RETURN
0356 F0 09            BEQ   ABORT  YES - ABORT LISTING
0358 A9 00     WAIT   LDAIM $00    MAKE A LONG DELAY
035A 20 A8 FC         JSR   $FCA8  CALL MONITOR DELAY SUBROUTINE
035D 68               PLA          GET CHARACTER FROM STACK
035E 4C F0 FD  CHROUT JMP   $FDF0  REJOIN COUT SUBROUTINE
0361 4C 03 E0  ABORT  JMP   $E003  SOFT ENTRY INTO APPLE BASIC
0364 2C 00 C0  STOP   BIT   $C000  WAIT UNTIL KEYBOARD STROBE
0367 10 FB            BPL   STOP   APPEARS ON THE SCENE
0369 8D 10 C0         STA   $C010  CLEAR THE STROBE
036C 30 EA            BMI   WAIT   UNCONDITIONAL BRANCH

                     SUBROUTINE TO DE-ACTIVATE SLOW LIST

036E A9 F0     OFF    LDAIM $F0    RESTORE $FDF0 TO
0370 85 36            STAZ  $36    LOCATIONS 36 AND 37
0372 A9 FD            LDAIM $FD
0374 85 37            STAZ  $37
0376 60               RTS

                     SUBROUTINE TO ACTIVATE SLOW LIST

0377 A9 40     ON     LDAIM $40    SET $0340 INTO
0379 85 36            STAZ  $36    LOCATIONS 36 AND 37
037B A9 03            LDAIM $03
037D 85 37            STAZ  $37
037F 60               RTS

SYMBOL TABLE
ABORT  0361    CHROUT 035E    OFF    036E    ON     0377
SLOW   0340    STOP   0364    WAIT   0358

SYMBOL TABLE
SLOW   0340    WAIT   0358    CHROUT 035E    ABORT  0361
STOP   0364    OFF    036E    ON     0377
```

AN APPLE-II PROGRAMMER'S GUIDE

(You Can Get There From Here!)

Rick Auricchio
59 Plymouth Avenue
Maplewood, NJ 07040

Most of the power of the APPLE-II comes in a "secret" form - almost undocumented software. After several months of coding, experimenting, digging, and writing to APPLE, most of the APPLE's pertinent software details have come to light.

Although most of the ROM software has been printed in the APPLE Reference Manual, its Integer Basic has not been listed; as a result, this article will be limited to Monitor software. Perhaps when a source listing of Integer Basic becomes available, we'll be able to interface with some of its many routines.

First Things First

When I took delivery of my Apple (July 1977), all I had was a "preliminary" manual - no goodies like listings or programming examples. My first letter to Apple brought a listing of the Monitor. Seeing what appeared to be a big jumble of instructions, I set out dividing the listing into logical routines while deciphering their input and output parameters. Once this was done, I could look at portions of the code without becoming dizzy.

The Monitor's code suffers from a few ills:

1 Subroutines lack a descriptive "preamble" stating function, calling seqquences, and interface details.

2 Many subroutines have several entry points, each of which does something slightly different.

3 Useful routines are not documented in a concise form for user access.

I will concede that, while using a "shoehorn" to squeeze as much function as possible into those tiny ROM's, some shortcuts are to be expected. However, those valuable Comment Cards don't use up any memory space in the finished product - 'nuff said.

The Good Stuff

The best way to present the Apple's software interface details is to describe them in tabular form, with further explanation about the more complex ones. The following tables will be found on the back cover of this issue:

Table 1 outlines the important data areas used by the Monitor. These fields are used both internally by the Monitor, and in user communication with many Monitor routines. Not all of the data fields are listed in Table 1.

Table 2 gives a quick description of most of the useful Monitor routines: it contains Name, Location, Function, Input/Output parameters, and Volatile (clobbered) Registers.

Don't hesitate to experiment with these routines - since all the important software is in ROM, you can't clobber anything by trying them out (except what you might have in RAM, so beware).

Using the "User Exits"

The Monitor provides a few nice User Exits for us to get our hands into the Monitor. With these, it is a simple matter to "hook in" special I/O and command-processing routines to extend the Apple's capabilities.

Two of the most useful exits are the KEYIN and COUT exits. These routines, central to the function of the Monitor, are called to read the keyboard and output characters to the screen. By placing the address of a user routine in CSWH/L or KSWH/L, we will get control from the Monitor whenever it attempts to read the keys or output to the screen.

As an example of this exit's action, try this: with no I/O board in I/O Slot 5, key-in "Kc5" (5, followed by control K, then Return). You'll have to hit Reset to stop the system.

AN APPLE II PROGRAMMER'S GUIDE

Rick Auricchio
59 Plymouth Avenue
Maplewood, NJ 07040

MONITOR Data Areas in Page Zero

Name	Loc.	Function
WNDLEFT	20	Scrolling window: left side (0-$27)
WNDWDTH	21	Scrolling window: width (1-$28)
WNDTOP	22	Scrolling window: top line (0-$16)
WNDBTM	23	Scrolling window: bottom line (1-$17)
CH	24	Cursor: horizontal position (0-$27)
CV	25	Cursor: vertical position (0-$17)
COLOR	30	Current COLOR for PLOT/HLIN/VLIN functions
INVFLG	32	Video Format Control Mask: $FF=Normal, $7F=Blinking, $3F=Inverse
PROMPT	33	Prompt character: printed on GETLN CALL
CSWL	36	Low PC for user exit on COUT routine
CSWH	37	High PC for user exit on COUT routine
KSWL	38	Low PC for user exit on KEYIN routine
KSWH	39	High PC for user exit on KEYIN routine
PCL	3A	Low User PC saved here on BRK to Monitor
PCH	3B	High User PC saved here on BRK to Monitor
A1L	3C	A1 to A5 are pairs of Monitor work bytes
A1H	3D	
A2L	3E	
A2H	3F	
A3L	40	
A3H	41	
A4L	42	
A4H	43	
A5L	44	
A5H	45	
ACC	45	User AC saved here on BRK to Monitor
XREG	46	User X saved here on BRK to Monitor
YREG	47	User Y saved here on BRK to Monitor
STATUS	48	User P status saved here on BRK to Monitor
SPNT	49	User Stack Pointer saved here on BRK

Page 2 ($0200-$02FF) is used as the KEYIN Buffer.

Pages 4-7 ($0400-$07FF) are used as the Screen Buffer.
Page 8 ($0800-$08FF) is the "secondary" Screen Buffer.

Table 1.

MONITOR ROUTINES

Name	Loc.	Steps On	Function
PLOT	F800	AC	Plot a point. COLOR contains color in both halves of byte ($00-$FF). AC: y-coord, Y: x-coord.
CLRSCR	F832	AC,Y	Clear screen - graphics mode.
SCRN	F871	AC	Get screen color. AC: y-coord, Y: x-coord.
INSTDSP	F8D0	ALL	Disassemble instruction at PCH/PCL.
PRNTYX	F940	AC	Print contents of Y and X as 4 hex digits.
PRBL2	F94C	AC,X	Print blanks: X is number to print.
PREAD	FB1E	AC,Y	Read paddle. X: paddle number 0-3.
SETTXT	FB39	AC	Set TEXT mode.
SETGR	FB40	AC	Set GRAPHIC mode (GR).
VTAB	FC22	AC	VTAB to row in AC (0-$17).
CLREOP	FC42	AC,Y	Clear to end-of-page.
HOME	FC58	AC,Y	Home cursor and clear screen.
SCROLL	FC70	AC,Y	Scroll up one line.
CLREOL	FC9C	AC,Y	Clear to end-of-line.
NXTA4	FCB4	AC	Increment A4 (16 bits), then do NXTA1.
NXTA1	FCBA	AC	Increment A1 (16 bits). Set carry if result >= A2.
RDKEY	FD0C	AC,Y	Get a key from the keyboard.
RDCHAR	FD35	AC,Y	Get a key, also handles ESCAPE functions.
GETLN	FD6A	ALL	Get a line of text from the keyboard, up to the carriage return. Normal mode for Monitor. X returned with number of characters typed in.
CROUT	FD8E	AC,Y	Print a carriage return.
PRBYTE	FDDA	AC	Print contents of AC as 2 hex digits.
COUT	FDED	AC,Y	Print character in AC; also works for CR, BS, etc.
PRERR	FF2D	AC,Y	Print "ERR" and bell.
BELL	FF3A	AC,Y	Print bell.
RESET	FF59	--	RESET entry to Monitor - initialize.
MON	FF65	--	Normal entry to 'top' of Monitor when running.
SWEET16	F689	None	SWEET16 is a 16-bit machine language interpreter. [See: SWEET16: The 6502 Dream Machine, Steve Wozniak,] [BYTE, Vol. 2, No. 11, November 1977, pages 150-159.]

Table 2.

Here's what happened: setting the keyboard to device 5 causes the Monitor to install $C500 as the "user-exit" address in KSWH/L. This, of course, is the address assigned to I/O Slot 5. Since no board is present, a BRK opcode eventually occurs; the Monitor prints the break and the registers, then reads for another command. Since we still exit to $C500, the process repeats itself endlessly. Reset removes both user exits; you must "re-hook" them after every Reset.

These two exits can enable user editing of keyboard input, printer driver programs, and many other ideas. Their use is limited to your ingenuity.

Another useful exit is the Control Y command exit. Upon recognition of Control Y, the Monitor issues a JSR to location $03F8. Here the user can process commands by scanning the original typed line or reading another. This exit is often very useful as a shorthand method of running a program. For example, when you're going back and forth between the Monitor and the Mini-Assembler, typing "F666G" is a bit tiresome. By placing a JMP $F666 in location $03F8, you can enter the Mini-Assembler via a simple Control Y.

Upon being entered from the Monitor at $03F8, the registers are garbage. Locations A1 and A2 contain converted values from the command (if any), and an RTS gets you neatly back into the Monitor. Figure 1 shows this in more detail.

Figure 1: Control Y Interface

Command typed:

```
*1234.F5A7Yc
```

Upon entry at $03F8, the following exists:

```
A1L ($3C) contains $34
A1H ($3D) contains $12
A2L ($3E) contains $A7
A2H ($3F) contains $F5
```

Hardware Features

One of the best hardware facilities of the Apple-II, the screen display, is also the "darkest" - somewhat unknown. Here's what I've found out about it.

The screen buffer resides in memory pages 4 through 7, locations $0400 through about $07F8. The Secondary screen page, although not accessed by the Monitor, occupies locations $0800 through $0BF8. Screen lines are not in sequential memory order; rather, they are addressed by a somewhat complex calculation carried out in the routine BASCALC. What BASCALC does is to compute the base address for a particular line and save it; whenever the cursor's vertical position changes, BASCALC recomputes the base address. Characters are stored into the screen buffer by adding the base address to the cursor's horizontal position.

I haven't made too much use of directly storing characters into the screen buffer; usually just storing new cursor coordinates will do the trick via the Monitor routines. Be careful, though - only change vertical position via the VTAB routine since the base address must get recomputed!

Characters themselves are internally stored in 6-bit format in the screen buffer. Bit 7 ($80), when set, forces normal (white-on-black) video display for the character. If Bit 7 is reset, the character appears inverse (black-on-white) video. Bit 6 ($40), when set, enables blinking for the character; this occurs only if Bit 7 is off. Thus an ASCII "A" in normal mode is $81; in inverse mode, $01; in blinking mode, $41.

Reading the keyboard via location $C000 is easy; if Bit 7 ($80) is set, a key has been pressed. Bits 0 - 6 are the ASCII keycode. In order to enable the keyboard again, its strobe must be cleared by accessing location $C010. Since the keyboard is directly accessible, there is no reason you can't do "special" things in a user program based on some keyboard input - if you get keys directly from the keyboard, you can bypass ALL of the Control and Escape functions.

APPLE INTEGER BASIC SUBROUTINE PACK AND LOAD

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

[Although this article is Copyrighted by The COMPUTERIST, Inc., at the authors request premission is hereby given to use the subroutine and to distribute it as part of other programs.]

The first issue of CONTACT, the Apple Newsletter, gave a suggestion for loading assembly language routines with a BASIC program. Simply summarized, one drops the pointer of the BASIC beginning below the assembly language portion, adds a BASIC instruction that will restore the pointer and SAVEs. The procedure is simple and effective but has two limitations. First, it is inconvenient if BASIC and the routines are widely separated (and is very tricky if the routines start at $800, just above the display portion of memory). Second, a program so saved cannot be used with another HIMEM, and is thus inconvenient to share or to submit to a software exchange.

The subroutine presented here avoids these difficulties at the expense of the effort to implement it. It is completely position independent; it may be moved from place to place in core with the monitor move command and used at the new location without modification. It makes extensive use of SWEET16, the 16 bit interpreter supplied as part of the Apple Monitor ROM.

To use the routine from Apple Integer BASIC, CALL MKUP, where MKUP is 128 (decimal) plus the first address of the routine. The prompt shown is "@". Respond with the hex limits of the routine to be stored, as BBBB.EEEE (BBBB is the beginning address, EEEE is the ending; the same format that the monitor uses). Several groups may be specified on one line separated by spaces or several lines. Type S after the last group to complete the pack and return to BASIC. The program can now be saved.

To load, enter BASIC and LOAD. When complete, RUN. The first RUN will move all routines back to their original location and return control to BASIC. It will not RUN the program; subsequent RUNs will.

A LIST of the program after calling MKUP and before the first RUN will show one BASIC statement (which initiates the restoration process) and gibberish. If this is done, RESET followed by CTRL C will return control to BASIC.

WARNING #1: The routine must be placed in core where it will not overwrite itself during the Pack. The start of the routine must be above HIMEM (e.g. in the high resolution display region) or $17A + 4*N + W below the start of the BASIC program, where N is the number of routines stored and W is the total number of words in all of these routines. Also, those routines that are highest in memory should be packed first to avoid overwriting during pack or restore. Otherwise it is not necessary to worry about overwriting during the restore process; only $1A words just below the BASIC program are used.

WARNING #2: Do not attempt to edit the program after calling MKUP. If editing is necessary, RUN once to unpack, then edit and call MKUP again.

The routine works as follows. It first packs the restore routine just below the BASIC program. It then packs other routines as requested, with first address and number of bytes (words). When S is given, it packs itself with the information to restore LOMEM and the beginning of the BASIC program. The first $46 words of the routine form a BASIC statement which will initiate the restoration process when RUN is typed.

If a particular HIMEM is needed by the program (e.g. for high resolution programs) it must be entered before LOADing. The LOMEM will be reset by the restoration process to the value it had when MKUP was called.

I do not have a SWEET16 assembler, hence all of those op codes are listed as tables of data. In the listing, comments indicate where constants and relative displacements are differences between labels in the routine.

Some convenient load and entry points are:

BASO (load) hex	MKUP (entry) hex	MKUP (entry) decimal
800	880	2176
A90	B10	2832
104C	10CC	4300
2050	20D0	8400
3054	30D4	12500

Editor's Note: While we encourage the use and distribution of this subroutine, we do request that proper credit be given. Please place the following notice on any copies that you make:

"This PACK & LOAD Subroutine was written by: Richard F. Suitor and published in MICRO #6."

```
                :INT BASIC SUBR PACK & LOAD
                :CALL BAS0+128(DEC)
                ACCL .DL 0000
                BSOL .DL 0002
                TABL .DL 0004
                TBCL .DL 0006
                HIMS .DL 0008
                LMRT .DL 000A
                BPRG .DL 000C
                FRML .DL 000E
                NBYT .DL 0010
                BPR2 .DL 0012
                PTLL .DL 0014
                XTAB .DL 0016
                SKPL .DL 0018
                MODE .DL 0031
                YSAV .DL 0034
                PRMP .DL 0033
                LMML .DL 004A
                HIML .DL 004C
                LMWL .DL 00CC
                BBSL .DL 00CA
                JSRL .DL 00CE
                BSC2 .DL E003       BASIC
                BUFF .DL 0200
                GTNM .DL FFA7
                PBL2 .DL F94A
                COUT .DL FDED
                BELL .DL FF3A
                GTLN .DL FD67
                SW16 .DL F689
                :BASIC INST. TO RESTORE
0800  460000    BAS0 .HS 46000064B101
0803  64B101
0806  0065B7         .HS 0065B74C000364B2
0809  4C0003
080C  64B2
080E  020065         .HS 020065382E3FB2CA
0811  382E3F
0814  B2CA
0816  007212         .HS 007212B74600721F
0819  B74600
081C  721F
081E  B20001         .HS B200010364B30300
0821  0364B3
0824  0300
0826  65382E         .HS 65382E3FB2CB0072
0829  3FB2CB
082C  0072
082E  12382E         .HS 12382E3FB2CA0072
0831  3FB2CA
0834  0072
0836  12B746         .HS 12B746007215B200
0839  007215
083C  B200
083E  017203         .HS 0172034DB1010001
0841  4DB101
0844  0001
                :INIT. RESTORE OP
0846  D8        PTBK CLD
0847  A201           LDX 01
0849  B5CA      PT02 LDA *BBSL,X
084B  9502           STA *BSOL,X
084D  B54C           LDA *HIML,X
084F  9508           STA *HIMS,X
0851  CA             DEX
0852  10F5           BPL PT02
0854  2089F6         JSR SW16
```

SYMBOL TABLE

ACCL	0000
BSOL	0002
TABL	0004
TBCL	0006
HIMS	0008
LMRT	000A
BPRG	000C
FRML	000E
NBYT	0010
BPR2	0012
PTLL	0014
XTAB	0016
SKPL	0018
MODE	0031
YSAV	0034
PRMP	0033
LMML	004A
HIML	004C
LMWL	00CC
BBSL	00CA
JSRL	00CE
BSC2	E003
BUFF	0200
GTNM	FFA7
PBL2	F94A
COUT	FDED
BELL	FF3A
GTLN	FD67
SW16	F689
BAS0	0800
PTBK	0846
PT02	0849
PT04	0870
MKUP	0880
MK21	0882
MK22	08B3
MK01	08B4
MK06	08CA
MERR	08D1
MK05	08DE
MK02	08E1
MV51	08EB
MV52	08F5
SM02	0909
SM03	090B
MK09	090C
MK11	091A
MK12	091B
MK10	0932
SM04	0946
PTLP	0952
PLP0	0955
PLP1	095A
PLP2	0966
ST16	096A

```
0857  105201   0520        .HS 105201    PLTP-BAS0
085A  185701   0530`       .HS 185701    PLTP+5-BAS0
085D  A13767   0540        .HS A13767356736
0860  356736
0863  24B636   0550        .HS 24B636
0866  1A1100   0560        .HS 1A1100    ST16+1-PLP1
0869  BA3A     0570        .HS BA3A
086B  6733     0580        .HS 6733
086D  00       0590        .HS 00
086E  A201     0600        LDX 01
               0610   :SET LOMEM & BASIC PROG START
0870  B50A     0620   PT04 LDA ◆LMRT,X
0872  954A     0630        STA ◆LMML,X
0874  95CC     0640        STA ◆LMWL,X
0876  B50C     0650        LDA ◆BPRG,X
0878  95CA     0660        STA ◆BBSL,X
087A  CA       0670        DEX
087B  10F3     0680        BPL PT04
087D  6C1400   0690        JMP (PTLL)    TO RESTORE LP
               0700   :SUBR TO SET UP PACK
0880  A201     0710   MKUP LDX 01
0882  B54A     0720   MK21 LDA ◆LMML,X
0884  950A     0730        STA ◆LMRT,X
0886  B5CA     0740        LDA ◆BBSL,X
0888  9512     0750        STA ◆BPR2,X
088A  950C     0760        STA ◆BPRG,X
088C  B5CE     0770        LDA ◆JSRL,X
088E  9504     0780        STA ◆TABL,X
0890  B54C     0790        LDA ◆HIML,X
0892  9508     0800        STA ◆HIMS,X
0894  CA       0810        DEX
0895  10EB     0820        BPL MK21
               0830   :INIT & PACK RESTORE LP
0897  2089F6   0840        JSR SW16
089A  24B939   0850        .HS 24B939
089D  118000   0860        .HS 118000    MKUP-BAS0
08A0  22B131   0870        .HS 22B131
08A3  105201   0880        .HS 105201    PLTP-BAS0
08A6  A13218   0890        .HS A132181800 ST16-PTLP
08A9  1800
08AB  A833E3   0900        .HS A833E3
08AE  1C5000   0910        .HS 1C5000
08B1  0C42     0920        .HS 0C42      MV52-MK22
08B3  00       0930   MK22 .HS 00
08B4  A9C0     0940   MK01 LDA 0C0
               0950   :GET LIMITS & PACK PROGS
08B6  8533     0960        STA ◆PRMP
08B8  A900     0970        LDA 0
08BA  8531     0980        STA ◆MODE
08BC  2067FD   0990        JSR GTLN
08BF  8616     1000        STX ◆XTAB
08C1  A000     1010        LDY 00
08C3  B90002   1020        LDA BUFF,Y
08C6  C9D3     1030        CMP 0D3       S
08C8  F068     1040        BEQ MK10
08CA  20A7FF   1050   MK06 JSR GTNM
08CD  C9A7     1060        CMP 0A7       F('.')
08CF  F010     1070        BEQ MK02
08D1  98       1080   MERR TYA
08D2  AA       1090        TAX
08D3  204AF9   1100        JSR PBL2      ERROR INDICATOR
08D6  A95E     1110        LDA '^
08D8  20EDFD   1120        JSR COUT
08DB  203AFF   1130        JSR BELL
08DE  18       1140   MK05 CLC
08DF  90D3     1150        BCC MK01
08E1  E631     1160   MK02 INC ◆MODE
08E3  20A7FF   1170        JSR GTNM
```

```
                 1180  :A1 & A3 NOW HAVE 1ST #,A2 2D
                 1190  :SET UP MOVE TO JUST BELOW (BBSL)
                 1200  :AND LOWER BBSL
08E6   2089F6    1210        JSR SW16
08E9   011E      1220        .HS 011E       SM02-MV51
08EB   183C00    1230  MV51  .HS 183C0068326833
08EE   683268
08F1   33
08F2   B238E3    1240        .HS B238E3
08F5   839623    1250  MV52  .HS 839623D207FA
08F8   D207FA
08FB   283318    1260        .HS 2833180800
08FE   0800
0900   889688    1270        .HS 8896889688968896
0903   968896
0906   8896
0908   0B        1280        .HS 0B
0909   0CE0      1290  SM02  .HS 0CE0       MV51-SM03
090B   00        1300  SM03  .HS 00
090C   C9EC      1310  MK09  CMP 0EC        F('S')
090E   F022      1320        BEQ MK10
0910   C9C6      1330        CMP 0C6        F(CR)
0912   F0A0      1340        BEQ MK01
0914   C999      1350        CMP 99         BLANK
0916   F003      1360        BEQ MK12
0918   D0B7      1370        BNE MERR
091A   C8        1380  MK11  INY
091B   B90002    1390  MK12  LDA BUFF,Y
091E   C416      1400        CPY *XTAB
0920   B092      1410        BCS MK01
0922   C9A0      1420        CMP 0A0        BLANK
0924   F0F4      1430        BEQ MK11
0926   C98D      1440        CMP 8D
0928   F08A      1450        BEQ MK01
092A   C9D3      1460        CMP 0D3        S
092C   F004      1470        BEQ MK10
092E   C631      1480        DEC *MODE
0930   F098      1490        BEQ MK06       ALWAYS
                 1500  :PACK 1ST PART & CLEAN UP
0932   2089F6    1510  MK10  JSR SW16
0935   2132      1520        .HS 2132
0937   185201    1530        .HS 185201     PTLP-BAS0
093A   A83725    1540        .HS A83725772977
093D   772977
0940   2177      1550        .HS 2177
0942   2733      1560        .HS 2733
0944   0CAF      1570        .HS 0CAF       MV52-SM04
0946   6666      1580  SM04  .HS 6666
0948   00        1590        .HS 00
0949   A50C      1600        LDA *BPRG
094B   85CA      1610        STA *BBSL
094D   A50D      1620        LDA *BPRG+01
094F   85CB      1630        STA *BBSL+01
0951   60        1640        RTS
                 1650  :RESTORE LOOP
0952   2089F6    1660  PTLP  JSR SW16
0955   613361    1670  PLP0  .HS 6133613800 GET POINT
0958   3800
095A   2089F6    1680  PLP1  JSR SW16
095D   4153F8    1690        .HS 4153F804FB
0960   04FB
0962   21D605    1700        .HS 21D605
0965   EF        1710        .HS EF         PLP0-PLP2
0966   00        1720  PLP2  .HS 00
0967   4C03E0    1730        JMP BSC2
096A   00        1740  ST16  .HS 00
                 1750        .EN
```

NUMERICAL VARIABLES STRING VARIABLES - ADD ($) APPLE II VARIABLES FOR APPLE SOFT BASIC

A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ
B	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ
C	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV	CW	CX	CY	CZ
D	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	DG	DH	DI	DJ	DK	DL	DM	DN	DO	DP	DQ	DR	DS	DT	DU	DV	DW	DX	DY	DZ
E	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	EG	EH	EI	EJ	EK	EL	EM	EN	EO	EP	EQ	ER	ES	ET	EU	EV	EW	EX	EY	EZ
F	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF	FG	FH	FI	FJ	FK	FL	FM	FN	FO	FP	FQ	FR	FS	FT	FU	FV	FW	FX	FY	FZ
G	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	GA	GB	GC	GD	GE	GF	GG	GH	GI	GJ	GK	GL	GM	GN	GO	GP	GQ	GR	GS	GT	GU	GV	GW	GX	GY	GZ
H	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	HA	HB	HC	HD	HE	HF	HG	HH	HI	HJ	HK	HL	HM	HN	HO	HP	HQ	HR	HS	HT	HU	HV	HW	HX	HY	HZ
I	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	IA	IB	IC	ID	IE	IF	IG	IH	II	IJ	IK	IL	IM	IN	IO	IP	IQ	IR	IS	IT	IU	IV	IW	IX	IY	IZ
J	J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	JA	JB	JC	JD	JE	JF	JG	JH	JI	JJ	JK	JL	JM	JN	JO	JP	JQ	JR	JS	JT	JU	JV	JW	JX	JY	JZ
K	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	KA	KB	KC	KD	KE	KF	KG	KH	KI	KJ	KK	KL	KM	KN	KO	KP	KQ	KR	KS	KT	KU	KV	KW	KX	KY	KZ
L	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LL	LM	LN	LO	LP	LQ	LR	LS	LT	LU	LV	LW	LX	LY	LZ
M	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	MA	MB	MC	MD	ME	MF	MG	MH	MI	MJ	MK	ML	MM	MN	MO	MP	MQ	MR	MS	MT	MU	MV	MW	MX	MY	MZ
N	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	NA	NB	NC	ND	NE	NF	NG	NH	NI	NJ	NK	NL	NM	NN	NO	NP	NQ	NR	NS	NT	NU	NV	NW	NX	NY	NZ
O	O0	O1	O2	O3	O4	O5	O6	O7	O8	O9	OA	OB	OC	OD	OE	OF	OG	OH	OI	OJ	OK	OL	OM	ON	OO	OP	OQ	OR	OS	OT	OU	OV	OW	OX	OY	OZ
P	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	PA	PB	PC	PD	PE	PF	PG	PH	PI	PJ	PK	PL	PM	PN	PO	PP	PQ	PR	PS	PT	PU	PV	PW	PX	PY	PZ
Q	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	QA	QB	QC	QD	QE	QF	QG	QH	QI	QJ	QK	QL	QM	QN	QO	QP	QQ	QR	QS	QT	QU	QV	QW	QX	QY	QZ
R	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	RA	RB	RC	RD	RE	RF	RG	RH	RI	RJ	RK	RL	RM	RN	RO	RP	RQ	RR	RS	RT	RU	RV	RW	RX	RY	RZ
S	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO	SP	SQ	SR	SS	ST	SU	SV	SW	SX	SY	SZ
T	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	TA	TB	TC	TD	TE	TF	TG	TH	TI	TJ	TK	TL	TM	TN	TO	TP	TQ	TR	TS	TT	TU	TV	TW	TX	TY	TZ
U	U0	U1	U2	U3	U4	U5	U6	U7	U8	U9	UA	UB	UC	UD	UE	UF	UG	UH	UI	UJ	UK	UL	UM	UN	UO	UP	UQ	UR	US	UT	UU	UV	UW	UX	UY	UZ
V	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	VA	VB	VC	VD	VE	VF	VG	VH	VI	VJ	VK	VL	VM	VN	VO	VP	VQ	VR	VS	VT	VU	VV	VW	VX	VY	VZ
W	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	WA	WB	WC	WD	WE	WF	WG	WH	WI	WJ	WK	WL	WM	WN	WO	WP	WQ	WR	WS	WT	WU	WV	WW	WX	WY	WZ
X	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF	XG	XH	XI	XJ	XK	XL	XM	XN	XO	XP	XQ	XR	XS	XT	XU	XV	XW	XX	XY	XZ
Y	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	YA	YB	YC	YD	YE	YF	YG	YH	YI	YJ	YK	YL	YM	YN	YO	YP	YQ	YR	YS	YT	YU	YV	YW	YX	YY	YZ
Z	Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	ZA	ZB	ZC	ZD	ZE	ZF	ZG	ZH	ZI	ZJ	ZK	ZL	ZM	ZN	ZO	ZP	ZQ	ZR	ZS	ZT	ZU	ZV	ZW	ZX	ZY	ZZ

LUDWIG VON APPLE II

Marc Schwartz
220 Everit Street
New Haven, CT 06511

Owners of the Apple II know from the demonstration tapes that the Apple can make sounds. Not all know that it can make music. Having prepared a horse racing program, I decided that it would be fitting to start out the game with the bugle call heard at the track. The following program does just that!

A few words of explanation are in order. The series of "pokes" in lines 30 to 240 set up a musical tone subroutine that is called in line 460.

Each note is represented by a four digit code in A$. The first three digits of the code determine the note, and the last digit determines the the length of the note. Line 410 decodes the first three digits by converting each digit to ASCII (Apple ASCII), subtracting 176 from each to give three numbers, from zero to nine, and then multiplying the first number by the second and adding the third. This is one of many possible ways of generating all the numbers from zero to a large number (ninety in this case) using single digits.

Line 420 takes the number just generated and subtracts it from forty. This is done because the subroutine as written is a bit confusing if you want to make music, since the tones go up as the numbers go down. This step corrects for that.

Line 440 determines how long each tone will be. As "ASC(A$(Z + 3) - 176)" increases, the note lengthens: a "1" produces a very short note, and a "6" makes a very long note. For some reason, higher tones come out more brief than lower tones.

Line 450 determines the tempo. A larger number speeds up the tune; a smaller one slows it down. Tempo numbers can go from 1 to 255.

When the program reaches line 470, it returns to line 400 to begin decoding the next four digits and playing the next note.

I don't think that Chopin would need to worry about competition from anyone using this program, but it is fun to have a musical computer.

THE APPLE II BUGLE CALL

```
10  REM MAKING MUSIC WITH THE APPLE II
20  DIM A$(255)
30  POKE 2,173
40  POKE 3,48
50  POKE 4,192
60  POKE 5,165
70  POKE 6,0
80  POKE 7,32
90  POKE 8,168
100 POKE 9,252
110 POKE 10,165
120 POKE 11,1
130 POKE 12,208
140 POKE 13,4
150 POKE 14,198
160 POKE 15,24
170 POKE 16,240
180 POKE 17,5
190 POKE 18,198
200 POKE 19,1
210 POKE 20,76
220 POKE 21,2
230 POKE 22,0
240 POKE 23,96

300 A$="0011007152117201720172O1"
310 A$(25)="5211521152110071521100710012"

400 FOR Z=1 TO LEN(A$)-3 STEP 4
410 Z1=(ASC(A$(Z))-176)*(ASC(A$(Z+1))-176)
    +ASC(A$(Z+2))-176
420 Z2=40-Z1
430 POKE 0,Z2
440 POKE 24,ASC(A$(Z+3))-176
450 POKE 1,75
460 CALL 2
470 NEXT Z
480 END
```

MACHINE LANGUAGE USED IN "LUDWIG VON APPLE II"

C. R. (Chuck) Carpenter W5USJ
2228 Montclair Place
Carrollton, TX
Carrollton, TX 75006

As an Apple II owner, I found the article "Ludwig von Apple II" (by Marc Schwartz, MICRO #2, page 19) quite interesting. The machine language routine used by Marc is put into the BASIC program by use of the POKE statement and I was curious to see the type of program used to activate the Apple II on-board speaker. To do this, I converted the decimal values used for the POKE statements into HEX with my TI Programmer. Then I loaded the values into the computer using the system monitor commands that are part of the Apple II functions.

Once I had the program loaded, I used the monitor commands to list an assembled version of the routine, as shown in Figure 1. The assembler provides a listing of the program and the mnemonics used with the machine language opcodes. This made it easier to determine what was happening in Marc's program. At this point I wanted to see what would happen if I ran the program by itself - as a machine language rouine only.

```
0000-   0F         ???
0001-   00         BRK
0002-   AD 30 C0   LDA   $C030
0005-   A5 00      LDA   $00
0007-   20 A8 FC   JSR   $FCA8
000A-   A5 01      LDA   $01
000C-   D0 04      BNE   $0012
000E-   C6 18      DEC   $18
0010-   F0 05      BEQ   $0017
0012-   C6 01      DEC   $01
0014-   4C 02 00   JMP   $0002
0017-   60         RTS
0018-   00         BRK
0019-   00         BRK
001A-   05 4B      ORA   $4B
001C-   B6 00      LDX   $00,Y
001E-   0F         ???
001F-   08         PHP
0020-   00         BRK
0021-   28         PLP
```

Figure 1.

Because it is somewhat easier to call the routine from a BASIC routine, I entered the BASIC routine shown in Figure 2. This way I could also change the values stored in memory location $0000 by using the POKE statement. To initialize the beginning of the routine, I entered a value of $05 into location $0000. According to Marc, this would produce a high frequency output tone and this turned out to be the case.

Now that I had everything set up, I was curious to see why the duration of playing time is not the same for the different tones. To start with, I entered the program with 3 different values at location $0000. As I ran the program I timed the length of playing with a stop watch. The value of 5 played for .18 min., 10 played for .45 min. and 15 played for .85 min. This was in agreement with Marc's findings. As it turns out, the length of time a particular frequency plays is a function of the duration of a cycle. The output continues for a number of cycles and the shorter cycles (higher frequencies) get done sooner. To get the correct musical timing you would need to include variable delay time for each note played. (The time between zero crossings adds up to the same total time per note.)

```
>LIST
   10 POKE 0,5
   99 END

>CALL 2

>10 POKE 0,10
>RUN

>CALL 2

>10 POKE 0,15
>RUN

>CALL 2
```

Figure 2.

APPLAYER MUSIC INTERPRETER

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

There have been several routines for making music with the APPLE II, including one in MICRO and one in the APPLE documentation. The program described here is more than a tone-making routine, it is a music interpreter. It enables one to generate a table of bytes that specify precisely the half-tone and duration of a note with a simple coding. Its virtue over the simpler routines is similar to that of any interpreter (such as Sweet 16, or, more tenuously, BASIC) over an assembler or hand coding - it is easier to achieve one's goal and easier to decipher the coding six months later.

The immediate motivation for this interpreter was Martin Gardner's Mathematical Games Column in the April 1978 Scientific American. Several types of algorithmically generated music are discussed in that column; this program provides a means of experimenting with them as well as a convenient method of generating familiar tunes.

The program is written in 6502 assembly language. It would be usable on a system other than the APPLE if a speaker were interfaced in a similar way. Accessing a particular address (C030) changes the current through the APPLE speaker from on to off or from off to on; it acts like a push button on/off switch (or, of course, a flip-flop). Thus this program makes sound by accessing this address periodically with an LDA C030. Any interface that could likewise be activated with a similar (4 clock cycles) instruction could be easily used. A different interfacing software procedure would change the timing and require more extensive modification.

The tone is generated with a timing loop that counts for a certain number of clock cycles, N (all of the cycles in a period including the toggling of the speaker are counted). Every N cycles a 24 bit pattern is rotated and the speaker is toggled if the high order bit is set. Four cycles are wasted (to keep time) if the bit is not set. There is a severe limit to the versatility of a waveshape made from on/off transitions, but tones resembling a variety of (cheap) woodwinds and pipes are possible, with fundamentals ranging from about 20 Hz to 8 KHz.

Applayer interprets bytes to produce different effects. There are two types of bytes:

Note bytes	Bit 7 Not Set
Control bytes	Bit 7 Set to 1

A note byte enables one to choose a note from one of 16 half tones, and from one to eight eighth notes in duration. The low order nybble is the half-tone; the high order nybble is the duration (in eighth notes) minus one.

Bit	7	6 5 4	3 2 1 0
Note Byte	0	(Duration)	(Half-Tone)

The control bytes enable one to change the tempo, the tonal range which the 16 half-tones cover, rests, the waveshape of the tone and to jump from one portion of the table to another.

Control Byte Table

HEX	DECIMAL	FUNCTION
81	129	The next three bytes are the new waveshape pattern
82	130	JMP - New table address follows. Low order byte first , then page byte
83	131	JSR - new table address follows. When finished, continuing this table at byte after address byte
9N	144+N	N is the number of 16th notes to be silent at the tail of a note. Controls rests and note definition
AN	160+N<32	Selects the tonal range. Half-tone #0 is set to one of 32 half-tones giving a basic range of four octaves
CN	192+N<62	Controls the tempo. Length of a note is proportional to N. Largest value gives a whole note lasting about 3.5 sec.
FF	255	RETURN. Stop interpreting this table. Acts as return for 83 JSR instruction or causes return from Applayer.

To use Applayer with sheet music, one must first decide on the range of the half tones. This must sometimes be changed in the middle of the song. For example, the music for "Turkey in the Straw", which appears later, was in the key of C; for the first part of the song I used the following table.

NOTE	C	D	E	F	G	A	B	C	D
TONE #	0	2	4	5	7	9	B	C	E

The tonal range was set with a control byte, B0. In the chorus, the range of the melody shifts up; there the tonal range is set with a B7 and the table is

NOTE	G	A	B	C	D	E	F	G	A
TONE#	0	2	4	5	7	9	A	C	E

(The actual key is determined by the wave shape pattern as well as the tonal range control byte. For the pattern used, 05 05 05, the fundamental for the note written as C would be about 346Hz, which is closer to F.)

Rests can be accomplished with a 9N control byte and a note byte. For example, 94 10 is a quarter rest, 98 30 is a half rest etc. This control is normally set at 91 for notes distinctly separated, or to 90 for notes that should run together.

Let's try to construct a table that Applayer can use to play a tune. We can start simply with "Twinkle, Twinkle Little Star". That tune has four lines the first and fourth are identical, as are the second and third. So our table will be constructed to:

1. Set up the tonal range, tone pattern and tempo that we want
2. JSR to a table for the first line
3. JSR to a table for the second line
4. Repeat #3
5. Repeat #2
6. Return
7. First line table and return
8. Second line table and return

Since unfortunately Applayer is not symbolic, it will be easier to construct the tables in reverse, so that we can know where to go in steps 2-6. The note table for the first line can go at 0B00 and looks like:

```
0B00-  10 10 17 17 19 19 37 15
0B08-  15 14 14 12 12 30 FF FF
```

The second line can follow at 0B10:

```
0B10-  17 17 15 15 14 14 32 FF
```

Now we can start on step 1. I'll suggest the following to start; you'll want to make changes:

```
0B20-  B0 81 05 05 05 E0 91
```

The above determines the tonal range, the tone wave shape, the tempo, and a sixteenth note rest out of every note to keep the notes distinct. To run them together, use 90 instead of 91. Steps 2 - 6 can follow immediately:

```
0B20-                      83
0B28-  00 0B 83 10 0B 83 10 0B
0B30-  83 00 0B FF
```

That completes the table for "Twinkle, Twinkle". We now have to tell Applayer where it is and turn it on. From BASIC we must set up some zero page locations first and then JSR to Applayer: (Don't forget to set LOMEM before running; 2900 will do for this table.)

```
100 POKE 19,32    (low order byte of the
                   table address, 0B20)
110 POKE 20,11    (high order byte of the
                   table address, 0B20)
120 POKE 1,8      (high order byte of 1st
                  pg of Applayer program)
130 POKE 17,8      (16 & 17 contain the
                  tone table address)
140 POKE 16,0
120 CALL 2346     (jump subroutine to
                   092A)
```

We can also make a short program in assembly language to set up the zero page locations. See routine ZERO, location 09C0 in the listing.

This initialization can be used most easily by reserving the A00 page, or much of it, as a "Table of Contents" for the various note tables elsewhere in memory. To do this with "Twinkle, Twinkle" we add the following table:

```
0A20-  82 20 0B
```

Which jumps immediately to the table at 0B20. With this convention, we can move from table to table by changing only the byte at 9D0 (2512 decimal).

We can use this initialization from BASIC, too, by changing the last instruction to RTS:

```
100 POKE 2512,32   LOW ORDER TABLE BYTE
110 POKE 2538,96   CHANGE INST. AT 09EA
120 CALL 2496      TO RTS.
```

From the monitor: *9D0:20
*9C0G

will do.

If, as I, you quickly tire of "Twinkle, Twinkle", you may wish to play with "Turkey in the Straw". The table follows; its structure will be left as an exercise.

From the monitor: *9D0:0
*9C0G

will play it.

```
0A00:  03 90 0F 83 90 0F FF

0F00:  90 1C 1A 92 38 90 18 1A
0F08:  18 13 10 11 91 13 13 33
0F10:  33 90 18 1A 92 3C 3C 90
0F18:  1C 1A 18 1A 91 1C 38 18
0F20:  38 90 1C 1A 92 38 90 18
0F28:  1A 18 13 91 10 11 13 53
0F30:  33 90 18 1A 91 3C 3F 90
0F38:  1F 1C 18 1A 1C 18 92 3A
0F40:  94 78 91 FF
0F50:  81 55 55 55 FF
0F58:  81 05 05 05 FF
0F60:  15 18 18 15 78 FF
0F68:  16 1A 1A 16 7A FF
0F70:  1D 1D 1D 1D 18 18 18 18
0F78:  35 15 15 33 90 11 13 91
0F80:  15 18 18 18 90 18 15 11
0F88:  13 91 15 15 13 13 71 FF
0F90:  83 58 0F D4 B0 83 50 0F 83
0F98:  B7 83 60 0F 83 50 0F 83
0FA0:  60 0F 83 50 0F 83 68 0F
0FA8:  83 50 0F 83 68 0F 83 50
0FB0:  0F 83 70 0F FF
```

Tone Table

```
0800:  A0 03 68 03 38 03 08 03
0808:  E0 02 B8 02 90 02 68 02
0810:  48 02 28 02 08 02 E8 01
0818:  D0 01 B4 01 9C 01 84 01
0820:  70 01 5C 01 48 01 34 01
0828:  24 01 14 01 04 01 F4 00
0830:  E8 00 DA 00 CE 00 C2 00
0838:  B8 00 AE 00 A4 00 9A 00
0840:  92 00 8A 00 82 00 7A 00
0848:  74 00 6D 00 67 00 61 00
0850:  5C 00 57 00 52 00 4D 00
0858:  49 00 45 00 41 00 3D 00
```

```
                    APPLAYER MUSIC INTERPRETER

                    R. F. SUITOR   APRIL 1978

                    TIMING LOOP
                    LOCATIONS 0 THROUGH 7 ARE SET BY CALLING ROUTINE
                    8 CYCLE LOOP TIMES Y REG PLUS 0-7 CYCLES
                    DETERMINED BY ENTRY POINT

0860                         ORG    $0860

0860 EA             TIME     NOP
0861 EA                      NOP
0862 EA                      NOP
0863 88             TIMEA    DEY
0864 85 45                   STA    $0045  ANY INNOCUOUS 3 CYCLE INSTRUCTION
0866 D0 FB                   BNE    TIMEA  BASIC 8 CYCLE LOOP
0868 F0 05                   BEQ    TIMEC
086A 88             TIMEB    DEY
086B EA                      NOP
086C EA                      NOP
086D D0 F4                   BNE    TIMEA
086F 24 04          TIMEC    BIT    $0004  START CHECK OF BIT PATTERN
0871 38                      SEC           IN 2, 3, AND 4
0872 30 02                   BMI    TIMED
0874 EA                      NOP
0875 18                      CLC
0876 26 02          TIMED    ROL    $0002
0878 26 03                   ROL    $0003
087A 26 04                   ROL    $0004
087C 90 03                   BCC    TIMEE
087E AD 30 C0                LDA    $C030  TOGGLE SPEAKER
0881 C6 06          TIMEE    DEC    $0006  DURATION OF NOTE IN
0883 D0 05                   BNE    TIMEF  NO. OF CYCLES IN LOCATIONS
0885 C6 07                   DEC    $0007  6 AND 7
0887 D0 05                   BNE    TIMEG
0889 60                      RTS
088A EA             TIMEF    NOP           TIMING EQUALIZATION
088B EA                      NOP
088C D0 00                   BNE    TIMEG
088E A4 05          TIMEG    LDY    $0005
0890 6C 00 00                JMI    $0000

                    SCALING ROUTINE FOR CYCLE DURATION
                    CALCULATION  LOC 6,7 = A REG * LOC 50,51

0893 85 45          SCALE    STA    $0045
0895 A9 00                   LDAIM  $00
0897 85 06                   STA    $0006
0899 85 07                   STA    $0007
089B A2 05                   LDXIM  $05
089D 18                      CLC
089E 66 07          SCALEX   ROR    $0007
08A0 66 06                   ROR    $0006
08A2 46 45                   LSR    $0045
08A4 90 0C                   BCC    SCALEA
```

```
08A6 A5 06             LDA    $0006
08A8 65 50             ADC    $0050
08AA 85 06             STA    $0006
08AC A5 07             LDA    $0007
08AE 65 51             ADC    $0051
08B0 85 07             STA    $0007
08B2 CA         SCALEA DEX
08B3 10 E9             BPL    SCALEX
08B5 E6 07             INC    $0007  DUE TO SIMPLE LOGIC IN TIMING ROUTINE
08B7 60                RTS

08BE                   ORG    $08BE

                NOTE PLAYING ROUTINE
                Y REG HAS HALF-TONE INDEX

08BE A5 12      NOTE   LDA    $0012  NOTE LENGTH
08C0 85 52             STA    $0052
08C2 A5 0F             LDA    $000F  NOTE TABLE OFFSET
08C4 85 10             STA    $0010
08C6 B1 10             LDAIY $0010  LOW ORDER BYTE OF MACHINE
08C8 38                SEC           CYCLES PER PERIOD
08C9 85 54             STA    $0054
08CB E9 35             SBCIM $35    CYCLES USED UP TIMING OVERHEAD
08CD 85 08             STA    $0008
08CF C8                INY
08D0 B1 10             LDAIY $0010  HIGH ORDER BYTE OF MACHINE
08D2 85 55             STA    $0055  CYCLES PER PERIOD
08D4 E9 00             SBCIM $00
08D6 85 09             STA    $0009
08D8 A9 00             LDAIM $00
08DA 85 50             STA    $0050
08DC 85 51             STA    $0051
08DE 85 53             STA    $0053
08E0 A0 10             LDYIM $10
08E2 20 86 FB          JSR    $FB86

                THIS PART IS PARTICULAR TO APPLE.  THE DIVIDE
                ROUTINE AT FB86 IS USED.  OR, PROVIDE A ROUTINE
                WHICH DIVIDES LOCS 54,55 BY 52,53 AND LEAVES THE
                RESULT IN 50,51 FOR THE SCALING ROUTINE.

08E5 A5 08             LDA    $0008
08E7 48                PHA
08E8 46 09             LSR    $0009
08EA 6A                RORA
08EB 46 09             LSR    $0009
08ED 6A                RORA
08EE 46 09             LSR    $0009
08F0 6A                RORA
08F1 85 05             STA    $0005  NO. OF 8 CYCLE LOOPS
08F3 68                PLA
08F4 29 07             ANDIM $07    LEFT OVER CYCLES DETERMINT
08F6 AA                TAX           ENTRY POINT
08F7 BD F8 09          LDAX   TTABLE TABLE OF ENTRY POINTS FOR TIMING LOOP
08FA 85 00             STA    $0000
```

```
08FC A5 0E              LDA   $000E  NOTE DURATION, QUARTER, HALF
08FE 38                 SEC
08FF E5 0D              SBC   $000D  REST PART OF NOTE
0901 F0 0F              BEQ   NOTEB  IF NOTHING TO DO
0903 20 93 08           JSR   SCALE  SCALING ROUTINE
0906 A2 02              LDXIM $02    START PATTERN LOAD
0908 B5 0A       NOTEA  LDAZX $0A
090A 95 02              STAZX $02
090C CA                 DEX
090D 10 F9              BPL   NOTEA
090F 20 6F 08           JSR   TIMEC  TIMING ROUTINE
0912 A5 0D       NOTEB  LDA   $000D  REST PART OF NOTE
0914 F0 0E              BEQ   MAIN   IF NOTHING TO DO
0916 20 93 08           JSR   SCALE  SCALING ROUTINE
0919 A9 00              LDAIM $00
091B 85 02              STA   $0002  ZERO OUT PATTERN FOR
091D 85 03              STA   $0003  REST PART
091F 85 04              STA   $0004
0921 20 6F 08           JSR   TIMEC  TIMING

0924                    ORG   $0924

                 MAIN PART OF INTERPRETER
                 ENTRY AT "ENTRY"

0924 E6 13       MAIN   INC   $0013  TABLE ADDRESS
0926 D0 02              BNE   ENTRY
0928 E6 14              INC   $0014

092A A0 00       ENTRY  LDYIM $00
092C B1 13              LDAIY $0013  NEXT TABLE BYTE
092E 30 12              BMI   MAINA  TO CONTROL SECTION
0930 48                 PHA
0931 29 0F              ANDIM $0F    TONE
0933 0A                 ASLA
0934 A8                 TAY
0935 68                 PLA
0936 29 70              ANDIM $70    DURATION
0938 4A                 LSRA
0939 4A                 LSRA
093A 4A                 LSRA
093B 69 02              ADCIM $02    TOTAL DURATION IN 16THS
093D 85 0E              STA   $000E
093F 4C BE 08           JMP   NOTE   PAY NOTE

0942 C9 FD       MAINA  CMPIM $FD    C0 + 3D IS LONGEST NOTE FOR
0944 90 01              BCC   MAINB  FOR SCALING REASONS
0946 60                 RTS

0947 48          MAINB  PHA
0948 0A                 ASLA
0949 10 07              BPL   MAINC
094B 68                 PLA
094C 29 3F              ANDIM $3F    NOTE LENGTH
094E 85 12              STA   $0012
0950 B0 D2              BCS   MAIN   UNCONDITIONAL BRANCH
```

```
0952 0A        MAINC  ASLA
0953 10 08            BPL   MAIND
0955 68               PLA
0956 29 1F            ANDIM $1F    TONAL RANGE INDEX
0958 0A               ASLA
0959 85 0F            STA   $000F
095B 90 C7            BCC   MAIN   UNCONDITIONAL BRANCH

095D 0A        MAIND  ASLA
095E 10 07            BPL   MAINE
0960 68               PLA
0961 29 0F            ANDIM $0F    REST FRACTION
0963 85 0D            STA   $000D
0965 90 BD            BCC   MAIN   UNCONDITIONAL BRANCH

0967 0A        MAINE  ASLA
0968 10 03            BPL   MAING
096A 68        MAINF  PLA
096B 90 B7            BCC   MAIN   DUMMY, CONTROLS NOT INTERPRETED
096D 0A        MAING  ASLA
096E 30 FA            BMI   MAINF
0970 0A               ASLA
0971 10 2B            BPL   MAINI
0973 68               PLA
0974 AA               TAX          JSR AND JMP SECTION
0975 4A               LSRA
0976 90 0A            BCC   MAINH
0978 A5 13            LDA   $0013  JSR SECTION, PUSH RETURN TABLE
097A 69 01            ADCIM $01    ADDRESS ON TO STACK
097C 48               PHA
097D A5 14            LDA   $0014
097F 69 00            ADCIM $00
0981 48               PHA
0982 C8        MAINH  INY
0983 B1 13            LDAIY $0013  GET NEW ADDRESS
0985 48               PHA
0986 C8               INY
0987 B1 13            LDAIY $0013
0989 85 14            STA   $0014
098B 68               PLA
098C 85 13            STA   $0013
098E 8A               TXA          AND STORE IT FROM BEGINNING
098F 4A               LSRA         OF SELECTION
0990 90 98            BCC   ENTRY  JMP
0992 20 2A 09         JSR   ENTRY  JSR
0995 68               PLA
0996 85 14            STA   $0014  PULL ADDRESS AND STORE IT
0998 68               PLA
0999 85 13            STA   $0013
099B 18               CLC
099C 90 86            BCC   MAIN   UNCONDITIONAL BRANCH

099E 68        MAINI  PLA
099F A0 03            LDYIM $03    GET NEW PATTERN AND
09A1 B1 13     MAINJ  LDAIY $0013  STORE IT
```

```
09A3 99 09 00          STAY  $0009
09A6 88                DEY
09A7 D0 F8             BNE   MAINJ
09A9 A5 13             LDA   $0013
09AB 69 03             ADCIM $03    JUMP OVER PATTERN
09AD 85 13             STA   $0013
09AF 90 02             BCC   MAINK
09B1 E6 14             INC   $0014
09B3 4C 24 09   MAINK  JMP   MAIN

09C0                   ORG   $09C0

                INITIALIZATION FOR ZERO PAGE

09C0 D8         ZERO   CLD          JUST IN CASE
09C1 A9 00             LDAIM $00
09C3 85 10             STA   $0010
09C5 A9 08             LDAIM $08
09C7 85 11             STA   $0011
09C9 85 01             STA   $0001
09CB A9 0A             LDAIM $0A
09CD 85 14             STA   $0014  NOTE TABLE PAGE
09CF A9 20             LDAIM $20
09D1 85 13             STA   $0013  NTOE TABLE BYTE
09D3 A9 01             LDAIM $01
09D5 85 0D             STA   $000D  REST 16THS
09D7 A9 20             LDAIM $20
09D9 85 12             STA   $0012  NOTE LENGTH, CONTROLS TEMPO
09DB A9 20             LDAIM $20
09DD 85 0F             STA   $000F  TONAL RANGE INDEX
09DF A9 05             LDAIM $05
09E1 85 0A             STA   $000A  WAVE SHAPE PATTERN
09E3 85 0B             STA   $000B
09E5 85 0C             STA   $000C
09E7 20 2A 09          JSR   ENTRY  TO APPLAYER
09EA 4C 69 FF          JMP   $FF69  TO MONITOR, AFTER THE BEEP

09F8                   ORG   $09F8

                TABLE OF ENTRY POINTS FOR TIMING ROUTINE

09F8 63         TTABLE =     $63
09F9 6A                =     $6A
09FA 62                =     $62
09FB 6D                =     $6D
09FC 61                =     $61
09FD 6C                =     $6C
09FE 60                =     $60
09FF 6B                =     $6B

ENTRY  092A    MAIN   0924    MAINA  0942    MAINB  0947
MAINC  0952    MAIND  095D    MAINE  0967    MAINF  096A
MAING  096D    MAINH  0982    MAINI  099E    MAINJ  09A1
MAINK  09B3    NOTE   08BE    NOTEA  0908    NOTEB  0912
SCALE  0893    SCALEA 08B2    TIME   0860    TIMEA  0863
TIMEB  086A    TIMEC  086F    TIMED  0876    TIMEE  0881
TIMEF  088A    TIMEG  088E    TTABLE 09F8    ZERO   09C0
```

APPLE II STARWARS THEME

Andrew H. Eliason
28 Charles Lane
Falmouth, MA 02540

Just for the fun of it, here are some routines to create something which sounds like the main battle scene from STARWARS. Enjoy!

Apple II Startrek Sounds Routine
Dis-assembler Listing

```
*3FA1L

3FA1-    A0 0E        LDY   #$0E
3FA3-    A2 00        LDX   #$00
3FA5-    8A           TXA
3FA6-    18           CLC
3FA7-    E9 01        SBC   #$01
3FA9-    D0 FC        BNE   $3FA7
3FAB-    8D 30 C0     STA   $C030
3FAE-    E8           INX
3FAF-    E0 8C        CPX   #$8C
3FB1-    D0 F2        BNE   $3FA5
3FB3-    88           DEY
3FB4-    D0 ED        BNE   $3FA3
3FB6-    60           RTS
3FB7-    00           BRK
3FB8-    00           BRK
3FB9-    00           BRK
3FBA-    00           BRK
3FBB-    00           BRK
3FBC-    00           BRK
3FBD-    00           BRK
*
```

Load via monitor starting at 3FA1:

```
3FA1.3FB6

3FA1- A0 0E A2 00 8A 18 E9
3FA8- 01 D0 FC 8D 30 C0 E8 E0
3FB0- 8C D0 F2 88 D0 ED 60
*
```

Enter BASIC and set HIMEM:16288.
Enter this program and RUN:

```
LIST

>LIST
  10 PRINT "STAR BATTLE SOUND EFFECTS"
  20 I= RND (15)+1: REM   SHOTS
  30 J= RND (11)*10+120: REM  DURATION
  40 POKE 16290,I: POKE 16304,J
  50 CALL 16289
  60 N= RND (1000): FOR K=1 TO N: NEXT K
  70 GOTO 20
 999 END

>
```

Try I = RND(30)+1 and J = RND(255).

The above material is based on the "Phaser" sound effect from Apple II Startrek.

SHAPING UP YOUR APPLE

Michael Faraday
246 Bronxville Road
Bronxville, NY 10708

Even though, as a programming novice, it took me a while to take on Apple II's Hi-Resolution Graphics I have to admit that the seeming complexity of constructing a Shape Table held a certain fascination for me from the first time I opened the Reference Manual. With Gary Dawkin's delightful program appearing in Creative Computing recently there is no longer any real need to apply the original technique, but a good understanding of something never hurt anyone, if only to verify other working arrangements.

If you have a TI Programmer, or any convenient way of converting from one base to another, here's a simplified method of untangling that unsightly jumble of arrows and binary digits on page 53 of the "Big Red Book". The key is in recognizing that the conversion chart is nothing more than an OCTal representation of our 8-bit

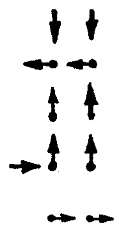
A/B	C	OCT
000	00	0
001	01	1
010	10	2
011	11	3
100		4
101		5
110		6
111		7

To the Code list we will add the OCTal number that each arrow represents.

byte. OCTal is binary broken into groups of three just as HEX is binary broken into groups of four. The fog lifts a little and we can now see why the "C" digit is limited to two bits: we only have a total of eight to start with. Looking a little further along the same page we come to the Conversion Codes and it's here we can begin to make things really easy.

C	B	A
0 0	0 1 0	0 1 0
0 0	1 1 1	1 1 1
0 0	1 0 0	0 0 0
0 1	1 0 0	1 0 0
0 0	1 0 1	1 0 1
...	...	...

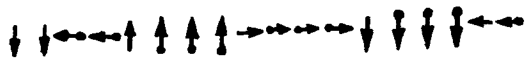
C B A



To the Code list we will add the OCTal number each arrow represents.

Going back to the original example in the manual we can replace the entire chart of binary digits with an OCTal number put directly above our "unwrapped" arrows, like so:

OCT 2 2 7 7 0 4 4 4 1 5 5 5 2 6 6 6 3 7

Shape 

We are going to construct either two- or three-digit numbers from this list and now come the only rules required to deal with in the whole procedure:

1. While always trying to make a three-digit number, the "last" digit of a three-digit group can ONLY be a 1, 2 or 3 (remember that the "C" digit is only 2 binary digits, which can represent the OCTal number three at most).

2. As usual, these numbers appear Least Significant Digit first and therefore the "last" digit is, in reality, the first digit of the new OCTal number.

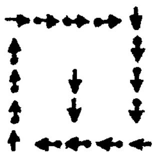
So we can now divide the long string of numbers into two- and three-digit, reverse-order OCTal numbers with slashes:

OCTal 2 2/7 7/0 4/4 4 1/5 5/5 2/6 6/6 3/7 "unwrap" this list, reversing digits as we go, and converting to HEX:

OCT	HEX
22	12
77	3F
40	20
144	64
...	...

Even this can be a bit tedious and since I find the arrow Code conversion very easy to remember - No Plot, Up Clockwise to Left = 0 to 3; Plot, Up Clockwise to Left = 4 to 7 - I draw my diagrams on graph paper using these OCTal numbers only.

Thus, becomes



```
1 5 5 5 2
4       6
4   2   6
4   2   6
0 7 7 7 3
```

Some caveats. It's still a good idea to draft an original diagram with plain dots just to get the shape and scale to your liking. This also becomes a handy guide for the debugging you're almost certain to have to do. And too, it makes great fun for your non-computer friends who might like to play Connect-the-Dots after a couple of beers.

A big problem keeps cropping up using the scale feature. It seems that when blowing up the original drawing the Apple II uses the direction of motion associated with the plotted points as a base reference for the additional points. This often leads to strangely assymetrical pictures in larger scale with "lines" of dots going in unexpected directions. As always, a little playing around can really make you feel good. Have fun.

Hexidecimal - Octal Conversion Table

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
1	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37
2	40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57
3	60	61	62	63	64	65	66	67	70	71	72	73	74	75	76	77
4	100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
5	120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137
6	140	141	142	143	144	145	146	147	150	151	152	153	154	155	156	157
7	160	161	162	163	164	165	166	167	170	171	172	173	174	175	176	177
8	200	201	202	203	204	205	206	207	210	211	212	213	214	215	216	217
9	220	221	222	223	224	225	226	227	230	231	232	233	234	235	236	237
A	240	241	242	243	244	245	246	247	250	251	252	253	254	255	256	257
B	260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277
C	300	301	302	303	304	305	306	307	310	311	312	313	314	315	316	317
D	320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337
E	340	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357
F	360	361	362	363	364	365	366	367	370	371	372	373	374	375	376	377

BROWN AND WHITE AND COLORED ALL OVER

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

This article consists of two parts. The first is a brief discussion of the colors of the Apple and their relationships to each other and to the color numbers. Some of that information is used in the second part to generate a random color display according to certain principles suggested by Martin Gardner in his mathematical games column in **Scientific American**.

The Color of Your Apple

The color of your Apple comes from your color TV. The video signal has many components. Most of the signal carries the brightness information of the picture - a black and white set uses this part of the signal to generate its picture. Superimposed on this signal is the "color carrier:, a 3.58 MHz signal that carries the color information. The larger tnis signal, the more colorful that region of the picture. The hue (blue, green, orange, etc.) is determined by the phase of the color signal. Reference timing signals at the beginning of each scan line synchronize a "standard" color signal. The time during a 3.58 MHz period that the picture color signal goes high compared to when the standard goes high determines the hue. A color signal that goes high when the standard does gives orange. One that goes low at that time gives blue. Signals that are high while the standard goes from high to low or from low to high give violet and green. (This, at least, was the intention. Studio difficulties, transmission paths and the viewers antenna and set affect these relations, so the viewer is usually given final say with a hue or tint control.)

The time relation of the color signal to the standard signal is expressed as a "phase angle", is measured in angular measures such as degrees or radians and can run from 0 to 360 degrees. This phase angle corresponds to position on a color circle, with orange at the top and blue at the bottom, as shown in Figure 1.

The perimeter of the circle represents different colors or hues. The radial distance from the center represents amount of color, or saturation. The former is usually adjusted by the tint control, the latter by the color control. A color that can be reproduced by a color TV can be related to a point in this circle. The angular position is coded in the phase of the 3.58 MHz color carrier signal; the radial distance from the center is given by the amplitude of the color carrier.

The numerical coding of the Apple colors can be appreciated using this circle and binary representation of the color numbers. The low order bit corresponds to red (#1). The second bit corresponds to dark blue (#2), the third to dark green (#4) and the high order bit to brown (dark yellow, #8). To find the color for any color number, represent each 1 bit as a quarter-pie piece centered over its respective color, as indicated in Figure 1. The brightness or lightness of the color corresponds to the number of pie pieces and the color corresponds to the point where the whole collection balances. Black, #0, has no bits set, no pie and no brightness. White, #15, has four bits set, the whole pie, is of maximum brightness and balances in the center of the circle at neutral. Orange, #9 or 1001 in binary, has pie over the top hemisphere and balances on a point between neutral and orange. The #5, binary 0101, has two separate wedges, one over red and one over green. Since it is symmetric, it balances at the center. It represents a neutral gray of intermediate brightness. So does the #10. The #14 has pie over every sector except the red one. It is bright and balances on a line toward forest green. It gives a light, somewhat bluish green.

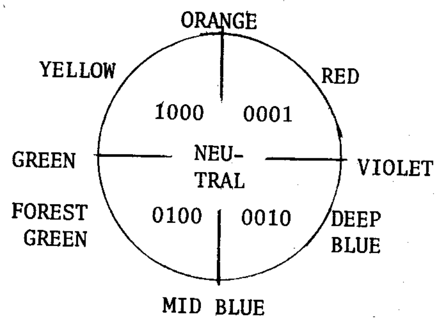


Figure 1.

Color circle shows relations of color to color number bit position.

A diagram representing the relations of all the colors is given in Figure 2. Each of the one, two and three bit numbers form planes, each corresponding to a color circle. One can think of these positions as points in space, with brightness increasing with vertical position and horizontal planes representing color circles of differing brightness.

The colors of the Apple are thus coded by the bit patterns of the numbers representing them. You can think of them as additive combinations of red, dark blue, dark green and brown, where adding two colors is represented by ORing the two numbers representing them. Subtractive combination can be represented by ANDing the light colors, pink, yellow, light green and light blue. The more bits set in a number, the brighter; the fewer, the darker. The bit patterns for 5 and 10 have no 3.58 MHz component and so generate a neutral tone. At a boundary between 5 and 10 however, this pattern is disturbed and two bits or spaces adjoin. Try the following program which has only grays dispplayed:

```
10 GR
20 FOR I = 0 TO 9
30 COLOR = 5
40 HLIN 0,39 AT 2*I
50 VLIN 20,39 AT 2*I
60 VLIN 20,39 AT 2*I+21
70 COLOR = 10
80 HLIN 0,39 AT 2*I + 1
90 VLIN 20,39 AT 2*I + 1
100 VLIN 20,39 AT 2*I + 20
110 NEXT I
120 RETURN
```

The top half of the display has HLIN's, alternating 5 and 10. The bottom half has VLIN's, alternating 5 and 10. What do you see? The bit pattern for a number is placed directly on the video signal, with the four bits occupying one color carrier period. When two bits adjoin at a

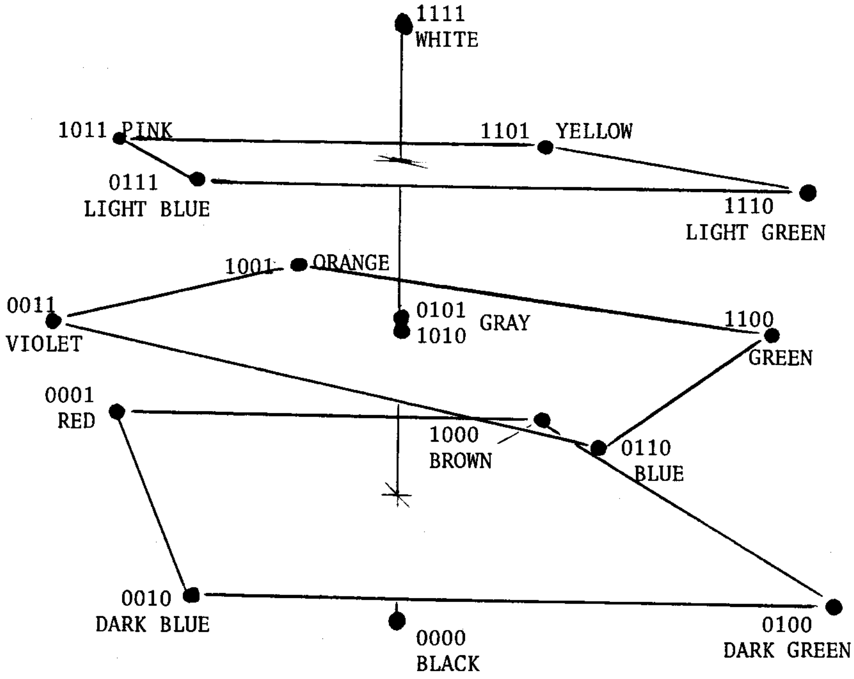


Figure 2.

Color space locations of the Apple II colors. Each horizontal plane forms a color circle of different brightness.

5,10 boundary, a light band is formed. When two spaces adjoin, a dark band is formed. The slight tints are due to the boundaries having some color component. Changing the 5,10 order reverses this tint.

Now is perhaps a good time to consider just how large a 3.58 MHz period is. The Apple text is generated with a 5x7 dot matrix, a common method of character generation. These same dots correspond to individual bits in the high resolution display memory. One dot is one-half of a 3.58 MHz period and corresponds to a violet (#3) or green (#12) color signal. This is why the test is slightly colored on a color TV and the high resolution display has two colors (other than black and white), green and violet. (But you can make others, due to effects similar to those seen in the BASIC program above.)

(The design of color TV has further implications for the display. The video black and white signal is limited to about 4 MHz, and many sets drop the display frequency response so that the color signal will not be obtrusive. A set so designed will not resolve the dots very well and will produce blurry text. Some color sets have adjustments that make the set ignore the color signal. Since the color signal processing involves subtracting and adding portions of the signal, avoiding this can sometimes improve the text resolution. Also reducing the contrast especially and the brightness somewhat can help with text material.)

The color TV design attempts to remove the color carrier from the picture (after duly providing the proper color), but you may be able to see the signal as 3 or 4 fine vertical lines per color block. They should not be apparent at all in the white or black or either gray (except possibly on a high resolution monitor).

Tan is Between Brown and White

This section presents a brief application of the concepts of the relationships in color space of the Apple colors. Many of you, I suspect, are regular readers of Martin Gardner's "Mathematical Games" column in Scientific American. I strongly recommend it to those of you who have not already been introduced. It publicized "Life" (MICRO 5:5) and motivated "Applayer" (MICRO 5:29), and was the motivation for this program. There's a lot of gold in the mine yet.

In April, the column discussed the aesthetic properties of random variations of different kinds. To summarize briefly, three kinds are:

WHITE — Each separate element is chosen randomly and is independent of every other element. Called "white" because a frequency spectrum of the result shows all frequencies occur equally, a qualitative description of white light.

BROWN — Each separate element is the previous element plus a randomly chosen deviation. Called "brown" because Brownian montion is an example.

1/F — So called because of its frequency spectrum, intermediate between "white" and "brown".

The column presented arguments, attributed to Richard Voss, that 1/f variations are prevalent and aesthetically more satisfying than "white" (not enough coherence) or "brown" (not enough variation). An algorithm was given for generating elements with 1/f random variations. Briefly, each element is the sum of N terms (three, say). One term is chosen randomly for each element. The next is chosen randomly for every other element. The next is chosen randomly for every fourth element, and so forth.

With the Apple, one can experiment with these concepts aurally (hence Applayer) and visually with the graphic displays. Color is a dimension that was not discussed much in the column. This section presents an attempt to apply these concepts to the Apple display.

Most of us know what "white" noise is like on the Apple display. An exercise that many try is to choose a random point, a random color, plot and repeat. For example:

```
10 GR
20 X = RND(40)
30 Y = RND(40)
40 COLOR = RND(16)
50 PLOT X,Y
60 GOTO 20
```

Dispite the garish display that results, this is a "white" type of random display. Except for all being within certain limits, the color of one square has no relationship to that of its neighbors and the plotting of one square tells nothing about which square is to be plotted next.

To implement the concept of "1/f", I used the following:

1. X and Y are each the sum of three numbers, one chosen randomly from each plot, one every 20 plots and the third every 200.

2. A table of color numbers was made (DIM(16) in the program) so that color numbers near each other would correspond to colors that are near each other. The choice given in the program satisfies the following restrictions:

   a. Adjacent numbers are from adjacent planes in Figure 2.

   b. No angular change (in the color planes) is greater than 45 degrees between adjacent numbers.

3. The color number is the same for 20 plots and then is changed by an amount chosen randomly from -2 to +2. This is a "brown" noise generation concept. However, most of the display normally has color patches that have been generated long before and hence are less correlated with those currently being plotted. I'll claim credit for good intentions and let someone else calculate the power spectrum.

4. Each "plot" is actually eight symmetric plots about the various major axes. I can't even claim good intentions here; it has nothing to do with 1/f and was put in for a kaleidoscope effect. Those who are offended and/or curious can alter statement 100. They may wish then to make X and Y the sum of more than three terms, with the fourth and fifth chosen at even larger intervals.

The program follows. A paddle and push buttons are used to control the tempo and reset the display. If your paddle is not connected, substitute 0 for PDL(0).

```
>LIST
  1 DIM A(16):A(1)=0:A(2)=2:A(3
    )=6:A(4)=7:A(5)=3:A(6)=1:A(
    7)=5:A(8)=11
  2 A(9)=9:A(10)=8:A(11)=10:A(12
    )=13:A(13)=15:A(14)=14:A(15
    )=12:A(16)=4
 10 GOTO 3000
100 PLOT X,Y: PLOT 38-X,Y: PLOT
    X,38-Y: PLOT 38-X,38-Y: PLOT
    Y,X: PLOT 38-Y,38-X: PLOT Y,
    38-X: PLOT 38-Y,X
110 RETURN
120 Z=16
125 L= RND (5)-2
130 U= RND (9):V= RND (9)
147 FOR B=1 TO 10
150 R=U+ RND (9):S=V+ RND (9)
155 IF PEEK (-16286)>127 THEN GR

160 K=K+L: IF K>16 THEN K=K-Z
165 IF K<0 THEN K=K+Z
170 COLOR=A(K)
180 Q=( PDL (0)/2) ^ 2
190 FOR I=-Q TO Q: IF PEEK (-16287
    )>127 THEN 200: NEXT I
200 FOR I=1 TO 20
210 X=R+ RND (6):Y=S+ RND (6): GOSUB
    100: NEXT I
220 NEXT B
230 GOTO 120
1010 K=1:L=5
1020 Z=16
2000 GOTO 120
3000 GR : CALL -936
3010 PRINT "PADDLE 0 CONTROLS PATTERN
     SPEED"
3020 PRINT "USE BUTTON 0 TO GO AT ONC
     E TO HI SPEED"
3030 PRINT "HOLD BUTTON 1 TO CLEAR SC
     REEN"
3040 GOTO 1010
9000 END

>CALL 858
```

GENERAL

* a perforated "tear-out" reference card

WE'RE NUMBER ONE!

An Editorial

We're number one in microcomputer systems. With over twelve thousand KIM-1 microcomputers in the field and a thousand per month being ordered, plus a good number of Apple I and Apple II systems, plus a variety of OSI units, plus the Jolts, Data Handlers, and other 6502-based systems, plus the huge numbers of PETs and Microminds that have been ordered, plus a lot of home-brew 6502 systems - it all adds up to a tremendous number of 6502-based microcomputer systems in use throughout the world. Adding to this number are the one and one-half million 650x chips purchased by Atari for some of their games. We've come a long way in the past year.

We're number one in microprocessor power. Microchess for the KIM-1 took 1.1K and for the 8080A took about 2.5K. Of thirty-one BASICs tested and reported in Kilobaud, the four 6502 versions placed in the top five spots, yielding only second place to the Z-80 running at 4 MHz. The 6502's many addressing modes make it very efficient and easy to program.

We're number one in user participation. Maybe there is some process of "natural selection" which attracts individuals who are industrious, able, cooperative, adventurous and communicative to the 6502. While users of other microprocessor chips have been "spoonfed" via company supported user notes and user libraries, the 6502 users have been "doing their own thing" as evidenced by the activity level of many local 6502 groups and the success of the KIM-1/6502 User Notes.

We're number one since this is our first issue. We would like to really become the most useful journal in the whole microcomputer field, not the largest, just the best. We are undertaking the venture with the conviction that there is a need for a journal to help bring all of the separate parts of the 6502 world together and with the belief that 6502 users will each do what they can to support the effort.

MICRO

COMPUTER CONTROLLED RELAYS

Robert M. Tripp
P.O. Box 3
S. Chelmsford, MA 01824

One of the easiest ways to expand the capabilities of a KIM-1 system is to provide a means of turning cassette tape recorders on and off under program control. This added capability permits a KIM-1, without a lot of additional memory, to perform editing, program assembly, mailing list maintenance, information retrieval, and other useful functions. One method of adding this computer control is by using relays as shown in the diagram below. To work reliably, a few components are required besides the relays.

The 7404 Hex Inverter is used to buffer the signals from the KIM's 6530 Port B I/O lines. There are many other IC chips which can also perform the buffering function. The 7404 was selected because it is so readily available.

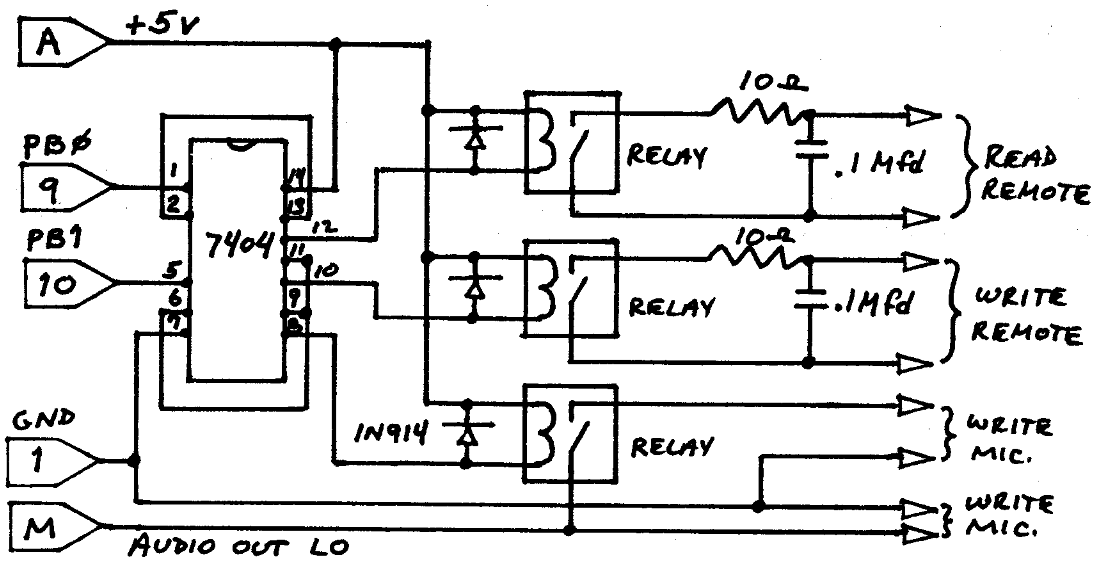
The clipping diodes on the coils of the relays are there to prevent a reverse voltage spike, generated when the relay is turned off, from damaging the buffer chip. Note that some relays may come with this diode already built-in.

The resistor on the contact side of the relay serves to limit the current drawn from the device connected to the relay. This is required where the device does have a current source, such as the "remote" switch in most cassette tape recorders.

The capacitor on the contact side of the relay serves to dump excess current that may occur during the initial surge when the relay makes its closure. Without this capacitor, many relays will have their contacts "welded" shut after a few operations.

Note that the contact side of relays which do not carry significant current do not require either the resistor or capacitor.

The KIM-1 circuitry is such that during a READ operation a signal is also present on the AUDIO OUT lines. This will cause a problem on tape recorders whose electronics are not turned off in the "remote" state, since the record head is active and the signal being generated by the READ will be written on the tape. This can wipe out data on the tape. A solution is provided by a third relay which is connected in parallel with the WRITE REMOTE relay and which is used to control the AUDIO OUT line. The record head is now active only when the WRITE REMOTE is selected. The AUDIO OUT line should also be brought out to another phono jack for use when writing tape using the normal KIM-1 Dump routine which does not know about the relays.



6502 INTERFACING FOR BEGINNERS: ADDRESS DECODING I

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

This is the first installment of a column which will appear on a regular basis as long as reader interest, author enthusiasm and the editor's approval exist. Your response will be vital for our deciding whether to continue the column. Do not be afraid to be critical or to make suggestions about what subjects you would like to see. Hopefully, the column will be of interest to anyone who owns a 6502 system. One of the more challenging aspects of being a computer hobbyist is understanding how your system works and being able to configure and construct I/O ports. Then one can begin to tie his computer to the outside world. Perhaps this column will give you the ability to produce flashing lights, clicking relays, whirring motors, and other remarkable phenomena to amaze your friends and make your mother proud.

An educational column has to make some assumptions about where the readers are in terms of their understanding. A familiarity with binary and hex numbers will be assumed, as will a nodding acquaintance with the 7400 series of integrated circuits. Lacking such a background I would recommend that you get a book like "Bugbook V" by Rony, Larsen, and Titus; "TTL Cookbook" by Lancaster; or an equivalent book from your local computer shop or mail order house. Ads in "Micro", "Byte", "Kilobaud", "Ham Radio", "73 Magazine", etc. will list places where both books and parts may be ordered. My own preference for "hands-on" experience would be "Bugbook V". Although this book has some material on the 8080A chip, most of the material is very general and the chapters covering the basic 7400 series integrated circuits are very good. Another indispensable book is the "TTL Data Book" published by Texas Instruments.

It would be a good idea to get a Proto Board or equivalent breadboarding system for the experiments which will be suggested. One can even find wire kits to go with the breadboards. I would not purchase all the Outboards from E & L Instruments since the same circuits can be constructed less expensively from parts. Please regard these suggestions as opinions which may not be shared by all experimenters.

Finally, let me introduce the column by saying that the title is not "Interfacing Made Easy". If it were easy there would be no challenge and no need for this column. Like mountain climbing, satisfaction comes from overcoming the difficult rather than achieving the obvious. The material which you see in this column will usually be something which I am in the process of learning myself. I am a hobbyist like yourselves: I keep the wolf from the door by teaching mathematics and physics, not computer science or digital electronics. Expert opinions from readers and guest contributions will always be welcome.

We begin at the beginning. The 6502 pins may be divided into four groups: power, address, data, and control pins. Pins 1 and 21 are grounds, and pin 8 is connected to the +5V supply, making the power connections. Pins 9 through 20 and 22 through 25 are connected to the address bus on the microcomputer, while the data pins, 26 through 33, are connected to the data bus. All of the remainder of the pins may be lumped in the general class of control pins. In subsequent issues the data bus and the control bus will be discussed. Our concern in the first two issues is with addressing.

The 6502 Address Bus

The 6502 receives data from a variety of devices (memory, keyboard, tape reader, floppy disc, etc.), processes it, and sends it back to one or more devices. The first process is called READ and is accomplished by the LDA or similar instruction. The last process is called WRITE and is achieved by a STA type instruction. The purpose of the address pins is to put out a signal on the address bus to select the device or location which is going to produce or accept the data. In the computer system, each device has a unique address, and when the 6502 puts that address on the address bus, the device must be activated. Each line on the address bus may have one of two possible values (high or low, H or L, 1 or 0, +5V or 0V are the names most frequently given to these values). A one-address-line system could select two devices; one activated by a 0 on the address line, the other by a 1. Figure 1 shows how to decode such an idiot microcomputer.

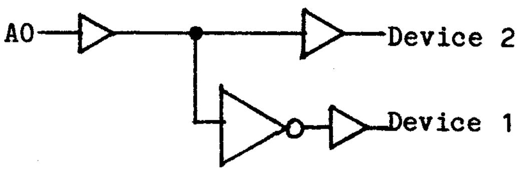


Figure 1. Decoding a One-Address Line Microprocessor.

Any device which when connected to the address bus puts out a unique signal (1 or 0) for a unique address is called a decoder. We have seen that a microcomputer with a single address line can select two devices, which could be memory locations or I/O ports. A somewhat smarter microprocessor might have two address lines. It could be decoded by the device shown in Figure 2, provided the truth table of the device were the one given in Table 1. Such a device could be implemented with NAND OR NOR gates, or with a 74139.

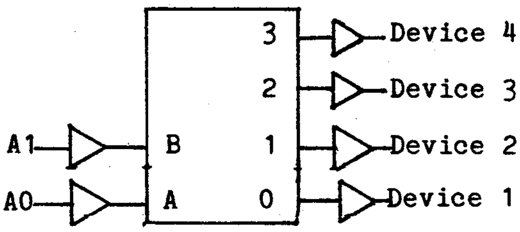


Figure 2. 74139 Decoder for a Two-Address Line Microprocessor.

Inputs		Outputs			
A	B	0	1	2	3
L	L	L	H	H	H
L	H	H	L	H	H
H	L	H	H	L	H
H	H	H	H	H	L

Table 1. Truth Table for Two-Line Decoder 74139.

The point is that two address lines allow the microprocessor to select four devices; three address lines give eight devices; four, 16; five, 32; six, 64; and so on. The 6502, being very smart, has 16 address lines. Anyone who can calculate how many telephones can be "addressed" by a 7-digit, base-ten phone number can also calculate how many locations can be addressed by a 16 digit, base-two address bus, The answers are 10^7=10 million and 2^{16}=65,536, respectively.

Earth people have not yet made a single device to simultaneously decode 16 address lines to produce 65,536 device select signals. Such a monster IC would need at least 65,554 pins. Many integrated circuits are constructed to decode the ten, low-order address lines (A0-A9) internally. For example, the 6530 PIA chips on the KIM and the 21L02 memory chips on my memory board decode the ten lowest address lines internally, that is, they select any one of the 2^{10} =1024 flip-flops to be written to or read. Consequently, our problem is to decode the high-order address lines, at least initially. These lines are usually decoded to form blocks of address space (not unlike home addresses in city blocks). Three address lines give eight (2^3=8) possible blocks, and the three highest address lines (A15-A13) divide the address space into eight blocks, each having $2^{(16-3)}$ $=2^{13}$ locations.

Now 1024 ($1024=2^{10}$) locations is usually referred to as 1K, so 2^{13} locations is 2^3 x 2^{10} locations, which is 8 x 2^{10} locations, which is 8K locations. Thus the top three address lines divide the address space into eight, 8K blocks. See Table 2 for more details. Each of these 8K blocks may be further divided

A15	A14	A13	Name	Hex Addresses
0	0	0	8K0	0000-1FFF
0	0	1	8K1	2000-3FFF
0	1	0	8K2	4000-5FFF
0	1	1	8K3	6000-7FFF
1	0	0	8K4	8000-9FFF
1	0	1	8K5	A000-BFFF
1	1	0	8K6	C000-DFFF
1	1	1	8K7	E000-FFFF

Table 2. "Blocking" the Memory Space.

into 1K blocks by decoding address lines A12-A10. Table 3 shows how block 8K4 is divided into eight, 1K blocks. Finally, as mentioned before, many devices decode the lowest 10 address lines, and consequently we have decoded all 16 address lines, at least on paper.

A12	A11	A10	Name	Hex Address
0	0	0	K32	8000-83FF
0	0	1	K33	8400-87FF
0	1	0	K34	8800-8BFF
0	1	1	K35	8C00-8FFF
1	0	0	K36	9000-93FF
1	0	1	K37	9400-97FF
1	1	0	K38	9800-9BFF
1	1	1	K39	9C00-9FFF

Table 3. Subdivision of 8K4 Block into 1K blocks.

To begin to see how this is done, construct the circuit shown in Figure 3.

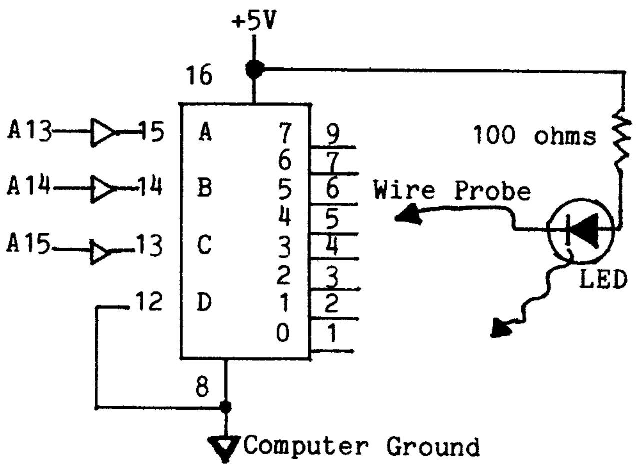


Figure 3. Decoding the Highest Three Address Lines.

(There are many decoding schemes and circuits, the circuit of Figure 3 is just one possible technique.) Here is where your breadboard becomes useful. Connect the address lines from your 6502 system to the 74145. (KIM owners can do this with no buffering because lines A15-A13 are not used on the KIM-1. Owners of other systems should check to see if the address lines are properly buffered.) Now perform the following experiments:

1. Load the following program somewhere between 0100 and 1FFF. The program is relocatable.

```
0200   18                 CLC
0201   8D XX 60   LOOP    STA 60XX
0204   90 FB              BCC LOOP
```

This routine stores Accum. in location 60XX. X means "don't care." Then loop back.

2. Run the program and with the wire probe shown in Figure 3, test each of the output pins (pins 1-7 and 9). Which ones cause the LED to glow?

3. Try to explain your results with the help of the truth table, Table 4.

4. Change the STA instruction to a LDA instruction (AD XX 60) and repeat steps 2 and 3 above.

5. In turn, change the location at which you are getting the data to a location in each of the 8K blocks in Table 2, e.g. 00XX, 20XX, 40XX, etc. and test the output pins on the 74145 to see if the LED glows. You should be able to explain your results with the truth table.

6. Stop the program and check the pins again.

Inputs			Outputs							
C	B	A	0	1	2	3	4	5	6	7
L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	H	H	H	H	H	H
L	H	L	H	H	L	H	H	H	H	H
L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	H	H	H	L	H	H	H
H	L	H	H	H	H	H	H	L	H	H
H	H	L	H	H	H	H	H	H	L	H
H	H	H	H	H	H	H	H	H	H	L

Table 4. Truth Table for 74LS145 when connected as shown in Figure 3.

In steps 2 and 4 the LED should glow when the probe touches pin 1 and pin 4. Why does it glow more brightly on pin 1? When the program is stopped, only pin 1 should cause the LED to light. The answers to these questions and the answers to questions you never asked will be given in the next issue.

What else is coming up in the next column? We will see how to take any of the 8 signals from the 74145 to enable a 74LS138 which in turn will decode address lines A12-A10, thus dividing any 8K block of address space which we may select into 1K blocks. Into one of these 1K blocks we will put some I/O ports.

(The more precocious of my attentive readers may already see that the scheme of Figure 3 could also be used to pre-set or clear a flip-flop to control an external device, for example, a heater, and all that without even using the data lines. If you see all that, you can take over this column.) See you next issue.

6502 INTERFACING FOR BEGINNERS: ADDRESS DECODING II

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

I hope you did not turn any expensive integrated circuits into cinders with last month's experiments. We will begin this month by considering the questions raised in the last column. You will need to refer to the circuits, tables, and the program described there. The following table describes the activity which takes place on the address bus and the data bus while the program is running. It is organized by clock cycles, each one microsecond long, starting with the op code fetch of the CLC instruction.

CYCLE	ADDRESS BUS	A15	A14	A13	DATA BUS	COMMENTS
0	0200	0	0	0	CLC op code	Pin 1 of LS145 is low because address lines A13-15 are low.
1	0201	0	0	0	STA op code	LED will glow when connected to pin 1, but not to other pins.
2	0201	0	0	0	STA op code	All other pins on LS145 are high.
3	0202	0	0	0	XX	Low order address of storage location on data lines.
4	0203	0	0	0	60	High order address of storage location on data lines.
5	60XX	0	1	1	accumulator contents	LED will light for 1 microsecond if connected to pin 4 on LS145.
6	0204	0	0	0	BCC op code	Pin 4 high, pin 1 low. LED will glow on pin 1 only.
7	0205	0	0	0	FB offset	6502 is now determing if and where to branch. Branch is to 0201 because
8	0206	0	0	0	garbage	carry was clear.

In the program loop address lines A14 and A13 go high only during cycle 5. Thus, for six cycles output 0 (pin 1) of the LS145 is low. The LS145 is an open collector device and acts like a switch to ground when the pin is in the L state, allowing current to flow through the LED. During cycle 5, when the address of the storage location is on the address bus, pin 4 is in the low state and will cause the LED to glow. Earth people do not perceive one microsecond flashes spaced six microseconds apart, so the LED appears to glow rather than flash. Since the majority of the loop time is spent with pin 1 at logic 0, a bright glow is observed on this pin. Changing the instruction from STA to LDA has no effect since the address bus goes through the same sequence for a LDA as it does for a STA. Changing the storage location from 60XX to something else will cause another pin of the LS145 to glow. The results of the LED test should agree with the truth table given for the LS145.

The pulse from the decoder which occurs when it responds to a particular address at its input pins is called a **device select pulse** or an **address select pulse**. The LS145 produces a logic 0 or active-low device select pulse, sometimes symbolized by ‾|_|‾ or $\overline{DS}$. This pulse is used to select or activate or enable another device in the computer system such as a memory chip, an I/O port, a PIA chip, or another decoder. As mentioned in the last column, the device select pulse from the LS145 could be used to enable a 74LS138 which would then decode address lines A10-12, dividing an 8K block into 1K blocks. Such a scheme is very similar to the expansion circuit suggested in the **KIM-1 USER MANUAL**, page 74. Similar circuits are also used on memory expansion boards. In the present circumstance I have decided to make a trade-off between wasting address space and minimizing the number of chips on the breadboard. Our purpose here is to configure some I/O ports as simply as possible.

The decoding circuit is shown in Figure 1. A total of eight device select pulses are available for eight I/O ports. Note that one of the 8K selects (8K4) from the LS145 enables the LS138 which decodes the three low-order address lines. All of the 8K4 space is used to get eight I/O ports. Using a 74LS154 instead of the LS138 and decoding on more address line would give 16 I/O ports in the event we need more. Or we could take another 8K select to enable another LS138 or LS145, giving us 8 or 32 ports, respectively. There is no doubt that address space is being wasted, but few users use all 64K, or even 32K, so the waste may be justified. In Figure 1, address lines A0-2 are extended downward to indicate that they could be decoded by other devices such as an LS138 or LS154.

The addresses which enable the device select pulses $\overline{DS}$0-7 are given in Figure 1. Note that since not all sixteen lines have been decoded to produce the pulses, the addresses shown are not the only ones which will work. For example, device select pulse 0 will be produced whenever the computer reads or writes to 8XX0 or 9XX0 (XX means any hex numbers). This should cause no difficulty unless we try to put other devices into the 8K4 block, in which case we could simply decode some other lines. If your system does not buffer the address lines, you should buffer them with the circuit shown in Figure 2.

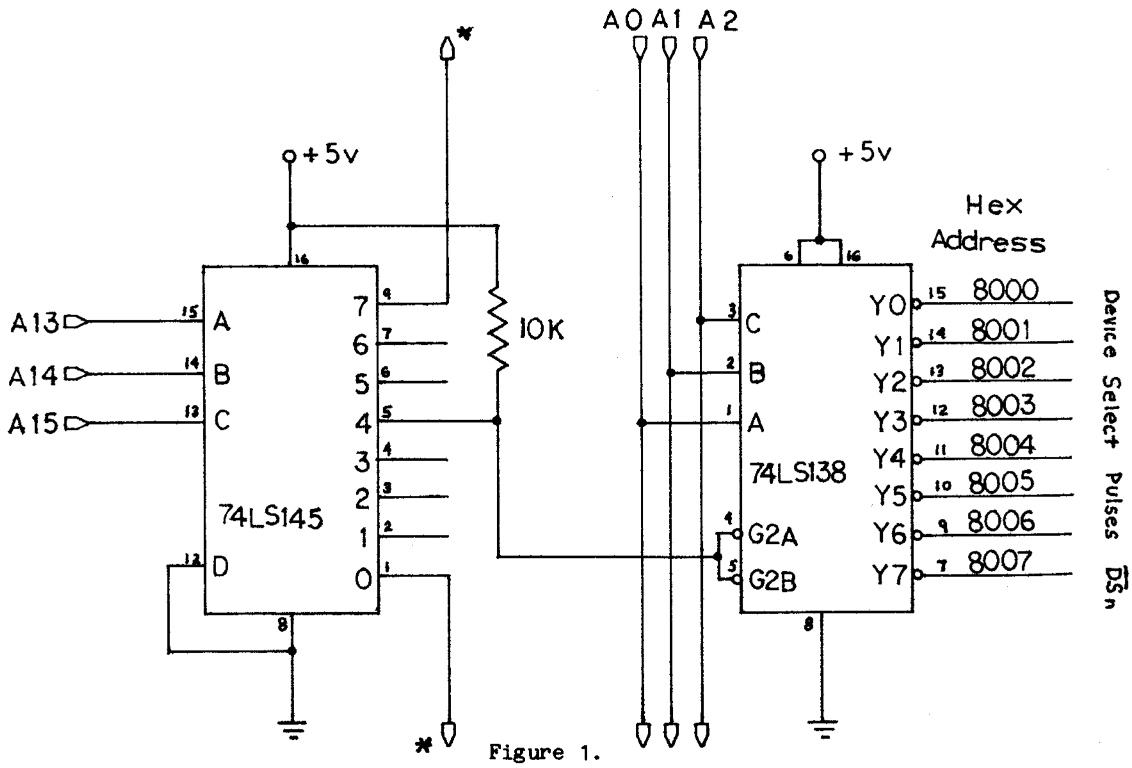


Figure 1.

Decoding Circuit to Select I/O Ports.
* See text for details.

Construct the circuits of Figures 1, 2, and 3. I managed to get them on one A P circuit board with no difficulty, with room for several more chips. I also found that the A P breadboard jumper wire kit is very handy for making neat layouts. Connect one of the device select lines from the LS138 to the flip-flop preset input (Test Circuit, Figure 3) and another device select line to the clear input. A pulse to the preset input will cause the Q output to go high, lighting the Q LED, whereas a pulse to the clear input will cause the $\overline{Q}$ output to go high, lighting the $\overline{Q}$ LED.

To test your decoding circuit write a one statement program, for example:

```
0200   AD 00 80          LDA DS0
```

If the line labeled 8000 is connected to the preset of the test circuit, the Q output will go high, lighting the LED, when the program is run. Running the program:

```
0200   AD 04 80          LDA DS4
```

will cause a switch of the flip-flop if the line 8004 is connected to the clear input. You should test all 8 device select lines from the LS138 with these programs by changing the connections and the addresses. Note that no data is being transferred since we have made no connections to the data bus. It should also be apparent that this scheme could be used to switch a motor, light, cassette recorder or other device off and on in a computer program. Eureka! We have made a simple I/O circuit.

To continue a little further, repeat the above experiments with a STA instruction replacing the LDA instruction. The results should be identical because in both cases it is the address of the device select on the address bus which produces the pulse which flips the flop. One more experiment: connect the R/W line from the 6502 to the G1 input on the LS138 after removing the connection from G1 (pin 6) to pin 16. Now try the programs above, using first a LDA instruction, then a STA instruction. You should find that the program with the LDA instruction

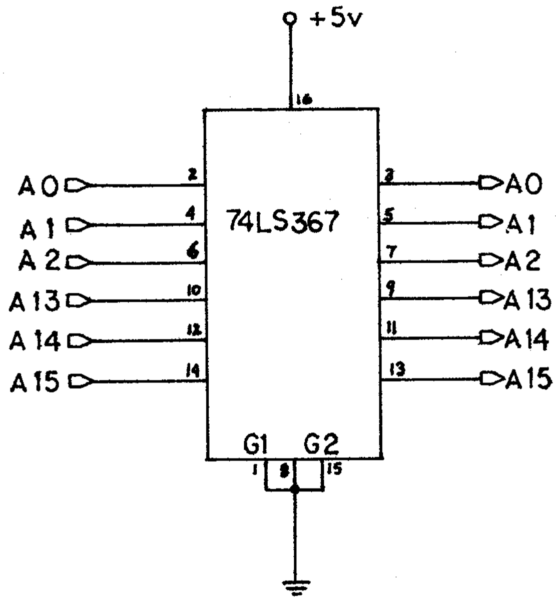


Figure 2.

Buffering the Address Lines.
The arrows pointing into the chip are the lines from the 6502, while those pointing away go to the circuit in Figure 1.

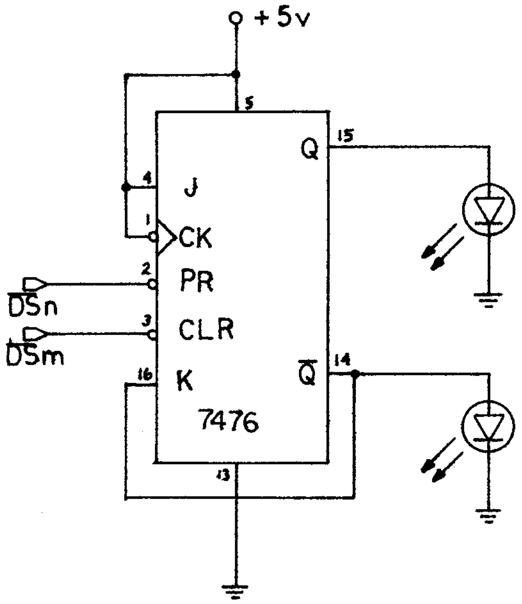


Figure 3. Test Circuit.

works, that is, the lights can be switched from off to on and vice versa, but the STA instruction does not work. Why?

Keep your circuit, as the material in the next column will refer to and make use of the circuit you have just completed.

A Note About Figure 1: The * lines in Figure 1 suggest that something should be done with them. For the experiments described above, nothing need be connected to these lines, however when we try to put data on the data bus these lines will become important. What you do depends on the system you are using. Since the KIM-1 is probably the most popular system among the readers, and since my own system is a KIM (expanded with a Riverside KEM and MVM-1024) the following details will be of most interest to KIM owners. Owners of other systems will have to dig into their manuals to make sure they are not de-selecting their on-board devices, or much worse, selecting two devices to put information on the data bus simultaneously. The KIM-1 has a 74145 decoder on-board which decodes lines A10-12; lines A13-15 are not decoded. Consequently, the lowest 8K0 block is already decoded, and the device select pulse from the LS145 in Figure 1 should enable the decoder on the KIM for all addresses in the 8K0 block. To do this simply connect the device select pulse from pin 1 on the 74LS145 in Figure 1 to pin K on the application connector on the KIM, making sure that the ground connection is first removed. A 10K pull-up resistor between pin 1 and +5V will also be necessary. The device select pulse from 8K7 should enable the device containing the restart and interrupt vectors. In the case of the KIM, pin 9 of the LS145 in Figure should enable the 6530-002 ROM by connecting it to pin J of the application connector. No pull-up is necessary.

Next issue we will examine the other pins on the 6502 which will be useful in configuring I/O ports, namely the bi-directional data bus, and the control signals. Hopefully we shall finish the circuitry needed to make an output port (8 bits), connect some LEDs to it, see if it works or smokes, and maybe think of a use for it.

A couple of parting shots: First, there is a very good educational series of articles in **KILOBAUD** magazine called KILOBAUD KLASSROOM. It assumes less experience than I have assumed so far. Second, I hope you have obtained a "TTL Databook" from either Texas Instruments or National so that you can study the truth tables and other specifications of the chips we are using.

An Additional Experiment

The address decoding circuit of Figure 1 produces a one microsecond negative going one-shot pulse when a LDA instruction addresses one of the locations shown in Figure 1. This one-shot can be used for a variety of purposes, one of which is triggering the flip-flop shown in Figure 3. The program listed below makes use of an interval timer (KIM-1 system addresses) to produce a square wave. By varying the time loaded into the timer, the frequency can be changed, and the duty cycle can be changed. Thus, we have produced a simple function generator with programmable period and duty cycle. The LEDs will show the results at low frequencies. Try this program and watch the LEDs. Amplify the Q output and connect it to a speaker; notice the effect of changing the time, the duty cycle, the wave shape (by filtering) or whatever else you can think of. Notice that I used device selects 8007 and 8001.

```
               DSEVEN *     $8007  DEVICE SELECT 7
               DSONE  *     $8001  DEVICE SELECT 1
               TIMER  *     $1707  KIM TIMER
               CLKRDI *     $1707  KIM CLOCK DONE TEST

0200 AD 07 80  START  LDA   DSEVEN INIT DS7 DEVICE SELECT PULSE
0203 A9 FF            LDAIM $FF    INIT TIMER
0205 8D 07 17         STA   TIMER  START DIVIDE-BY-1024 TIMER FOR 256
0208 AD 07 17  BACK   LDA   CLKRDI CYCLES, NOW CHECK TO SEE IF IT
020B 10 FB            BPL   BACK   IS FINISHED.  IF NOT, CHECK AGAIN,
020D AD 01 80         LDA   DSONE  OTHERWISE TRIGGER DS1.
0210 A9 FF            LDAIM $FF
0212 8D 07 17         STA   TIMER  START TIMER FOR SECOND HALF OF
0215 AD 07 17  AGN    LDA   CLKRDI CYCLE.  IS TIMER READY?
0218 10 FB            BPL   AGN    NO, CHECK AGAIN, OTHERWISE JUMP
021A 4C 00 02         JMP   START  TO START OVER.
```

TYPESETTING ON A 6502 SYSTEM

Robert M. Tripp
P.O. Box 3
S. Chelmsford, MA 01824

As Editor/Publisher of MICRO, I was bothered by the need to have typesetting done by an outside company for several reasons. First, of course, was the cost. A typeset page can cost from $12 to $30.00. Second, it takes time have a page set, anywhere from one to five days. Third, once you have the typeset material and are ready to paste up the final copy, it is very difficult to make any changes or corrections. It occurred to me that I should be able to do a reasonable job of typesetting with my existing equipment - a KIM-1 and a Diablo Hytype II based terminal. The results of my efforts are described in this article, and, this entire issue of MICRO has been produced with the equipment and program described.

Actually, "typesetting" is a misnomer for what is being done here. "type" is not being "set". Justification would probably be a better term, but still would not completely cover the features currently implemented. For lack of a term, I named this routine "JUSTIFY".

Features of Justify

JUSTIFY has four modes. The most useful is Full Justification in which a line is set justified at both the left and right margins. The lines you are reading now are an example of a Full Justification. In this mode the width of the column is specified as a parameter to the JUSTIFY routine which then pads the text as necessary to make the text exactly meet the right margin.

The second mode is No Justification. There are a number of instances in which you do not want the material to be justified: the last line of a paragraph, source listings, object listings, any type of tables, and so forth. The following listing makes the point quite graphically:

```
0120 A6 DB      JSTIFY LDXZ  CMND
0122 B5 00             LDAZX $00
0124 CA                DEX
```

which, if set with Full Justification would come out as

```
0120  A6  DB       JSTIFY LDXZ   CMND
0122  B5  00                LDAZX $00
0 1 2 4   C A                     D E X
```

Obviously not what was intended.

The third mode is Center. Title blocks of articles, headers for sections, and so forth need to be centered. The Center mode calculates where to start the text so that it will be properly centered, including spliting a character space in half to get perfect centering.

A
AA
AAA

The last mode currently implemented is actually not a form of justification, but is useful. It is an enhancement in which characters may be printed **slightly bolder** than the surrounding text to make them stand out. This mode is independent of the three justification modes and can be combined with any of them.

Although the JUSTIFY routine was made for typesetting MICRO, we have found it has many other uses. Since the editing portion of the program permits you to make corrections before printing, we can type "perfect" letters.

Justification Algorithm

The justification algorithm, or rules, used is based on certain characteristics of the Diablo printer. This printer "thinks small" - it divides the line into units which are 1/120th of an inch. Each printed character is normally 10 units wide, including the space around the character, giving 12 characters per inch. In TEXT mode, there is no way to space the characters other than next to each other as in regular typing, or separated by a full space. If this was the only method of positioning characters, then the justification would consist of expanding the spaces in a line to pick up the extra units to justify a line. This is the method required for a teletype printer. It looks like this:

This is teletype mode justification.

Note that the spaces between words has been doubled in the first three positions. This is not too bad, and as long as there are not too many spaces to distribute, can be acceptable. Given the Diablo's capability of padding with as little as a space of 1/120 of an inch, much better justification be achieved. If there are only a few units to be distributed over the line, then each normal space may be stretched just a little. For example, in a line which is only one character short of full, there are only ten units of space remaining to be distributed, since each character is 10 units wide. If the line contained five normal spaces, then each space would be stretched by two units, an almost imperceptible amount.

Full justification with an extra unit.
Full justification with no extra units.

As the number of units to be distributed increases, there comes a point at which the spaces become noticeably wide. The way this can be solved on the Diablo is to distribute spaces among the characters as well as the spaces. The calculation is done as:

1. Count number of extra units.
2. If there are more units than characters and spaces, then add one or more units to each character and space.
3. If there are fewer units than characters and spaces, then test just the spaces. If there are more units than spaces, then add one or more to each space.
4. When there are finally fewer units than spaces, distribute the remaining units over the first spaces in the line.

Each character has one unit added.
Characters have not had a unit added.

Close inspection will reveal that the first line above has the individual characters spaced slightly wider than the second line. This algorithm will handle most normal lines, but if a line has too many units to fill, it will look strange.

T h i s   i s   a   v e r y   l o o s e   l i n e .

The JUSTIFY Function

JUSTIFY is written in the form of a HELP Function. HELP is a sort of high level language I have developed and is the basis of the Editor, Mailing List, and Information Retrieval packages sold by The COMPUTERIST, as well as a large number of utilities we use internally for such operations as printing labels for cassette tapes, creating copies of program tapes, and so forth. Each of the Functions is, essentially, a subroutine which is called and passed a set of parameters. If the arguments required are placed in the proper locations - 00D9, DA, and DB - and if the instruction at location 01AB is changed from JMP NXTSTP to RTS, then JUSTIFY may be called as a simple subroutine.

Operation of Justify

JSTIFY uses the pointer in CMND+03 to pick up the full address of the buffer which contains the material to be justified, and stores it in BUFFER and BUFFER+01.

CLEAR puts zero in each of the seven counters, NULLS to TEMP, and then puts a zero at the first location past the end of the buffer as defined by the start of the BUFFER and the length as defined by the parameter CMND+01. This zero guarantees a null for the end of buffer test later on.

MORE starts at the end of the buffer to pick up and test each character in order to get a count of the number of nulls, spaces, and other characters. It also tests for a Control N (0E). A Control N is used to signal that No Justification is required on the current line and control branches to NEXT.

JUSTIFY FUNCTION - 16 JAN 1978

```
                 JUSTIF ORG    $0120

                 NULLS  *      $00CC
                 SPACES *      $00CD
                 CHARS  *      $00CE
                 COFSET *      $00CF
                 SOFSET *      $00D0
                 EXCESS *      $00D1
                 TEMP   *      $00D2
                 POINT  *      $00D3
                 BUFFER *      $00D4
                 MODE   *      $00D6
                 CMND   *      $00D8
                 OUTCH  *      $1EA0
                 NXTSTP *      $0304

0120 A6 DB       JSTIFY LDXZ   CMND   +03
0122 B5 00              LDAZX  $00
0124 85 D4              STAZ   BUFFER
0126 B5 01              LDAZX  $01
0128 85 D5              STAZ   BUFFER +01

012A A2 07              LDXIM  $07
012C A9 00              LDAIM  $00
012E 95 CC       CLEAR  STAZX  NULLS
0130 CA                 DEX
0131 10 FB              BPL    CLEAR
0133 A4 D9              LDYZ   CMND   +01
0135 91 D4              STAIY  BUFFER
0137 88                 DEY

0138 B1 D4       MORE   LDAIY  BUFFER GET CHARACTER TO COUNT
013A C9 0E              CMPIM  $0E
013C F0 59              BEQ    NEXT   NO JUSTIFICATION
013E C9 20              CMPIM  $20    TEST SPACE CHARACTER OR LESS
0140 F0 1E              BEQ    SCOUNT EQUAL SPACE
0142 10 1E              BPL    CCOUNT EQUAL CHARACTER
0144 E6 CC              INCZ   NULLS  EQUAL NULL
0146 88          AGAIN  DEY           DECREMENT STRING COUNTER
0147 10 EF              BPL    MORE

0149 C8          TEST   INY
014A B1 D4              LDAIY  BUFFER
014C C9 0B              CMPIM  $0B
014E F0 16              BEQ    CENTER
0150 C6 CE              DECZ   CHARS
0152 A6 CC              LDXZ   NULLS  TEST ANY NULLS
0154 F0 41              BEQ    NEXT   NO NULLS
0156 A5 DA              LDAZ   CMND   +02
0158 CA          MULT   DEX           CALCULATE UNITS TO EXPAND
0159 F0 22              BEQ    DIVIDE GO TO DIVIDE
015B 18                 CLC
015C 65 DA              ADCZ   CMND   +02
015E D0 F8              BNE    MULT   MULT LOOP UNTIL DONE
```

```
0160 E6 CD      SCOUNT INCZ  SPACES
0162 E6 CE      CCOUNT INCZ  CHARS  BUMP SPACES AND CHAR COUNTERS
0164 D0 E0             BNE   AGAIN

0166 E6 D3      CENTER INCZ  POINT
0168 46 CC             LSRZ  NULLS
016A 90 06             BCC   SHIFT
016C A5 DA             LDAZ  CMND   +02
016E 4A                LSRA
016F 20 BE 01          JSR   OFFSET

0172 A9 20      SHIFT  LDAIM $20
0174 20 A0 1E          JSR   OUTCH
0177 C6 CC             DECZ  NULLS
0179 D0 F7             BNE   SHIFT
017B F0 1A             BEQ   NEXT

017D C5 CE      DIVIDE CMPZ  CHARS  TEST CHAR SPACING
017F 30 09             BMI   DIVDON UNITS < CHARS
0181 38                SEC          UNITS >= CHARS
0182 E5 CE             SBCZ  CHARS  HOW MANY UNITS PER CHAR
0184 E6 CF             INCZ  COFSET BUMP COUNTERS
0186 E6 D0             INCZ  SOFSET
0188 D0 F3             BNE   DIVIDE UNCOND. BRANCH

018A C5 CD      DIVDON CMPZ  SPACES REMAINDER TO SPACES
018C 30 07             BMI   SDONE
018E 38                SEC
018F E5 CD             SBCZ  SPACES
0191 E6 D0             INCZ  SOFSET
0193 D0 F5             BNE   DIVDON

0195 85 D1      SDONE  STAZ  EXCESS REMAINDER TO EXCESS

0197 A4 D3      NEXT   LDYZ  POINT  GET STRING POINTER
0199 E6 D3             INCZ  POINT  BUMP FOR NEXT TIME
019B B1 D4             LDAIY BUFFER FETCH CHARACTER
019D C9 18             CMPIM $18    BOLD?
019F F0 43             BEQ   BOLD
01A1 C9 19             CMPIM $19    NORMAL?
01A3 F0 3F             BEQ   BOLD
01A5 C9 20             CMPIM $20    TEST SPACE
01A7 F0 29             BEQ   SPACE
01A9 10 03             BPL   CHAR
01AB 4C 04 03          JMP   NXTSTP
01AE 20 A0 1E   CHAR   JSR   OUTCH
01B1 C6 CE             DECZ  CHARS  CORRECTION FOR LAST CHAR
01B3 30 E2             BMI   NEXT   LAST CHAR
01B5 A5 CF             LDAZ  COFSET FETCH OFFSET
01B7 F0 DE      NTEST  BEQ   NEXT
01B9 20 BE 01          JSR   OFFSET
01BC F0 D9             BEQ   NEXT

01BE AA         OFFSET TAX
01BF A9 10             LDAIM $10
```

```
01C1 20 A0 1E          JSR   OUTCH
01C4 A9 48      BUMP   LDAIM 'H
01C6 20 A0 1E          JSR   OUTCH
01C9 CA                DEX
01CA D0 F8             BNE   BUMP
01CC A9 1C             LDAIM $1C
01CE 20 A0 1E          JSR   OUTCH
01D1 60                RTS

01D2 20 A0 1E   SPACE  JSR   OUTCH
01D5 A5 D0             LDAZ  SOFSET FETCH SPACE OFFSET
01D7 A6 D1             LDXZ  EXCESS TEST EXTRA OUTPUT
01D9 F0 05             BEQ   NOXCES
01DB C6 D1             DECZ  EXCESS DECREMENT EXCESS
01DD 18                CLC
01DE 69 01             ADCIM $01    INCREMENT OFFSET
01E0 C9 00      NOXCES CMPIM $00
01E2 10 D3             BPL   NTEST

01E4 18         BOLD   CLC
01E5 69 1E             ADCIM $1E
01E7 AA                TAX
01E8 A9 1B             LDAIM $1B
01EA 20 A0 1E          JSR   OUTCH
01ED 8A                TXA
01EE 20 A0 1E          JSR   OUTCH
01F1 D0 A4             BNE   NEXT
```

TEST first checks to see if the Center Mode has been specified by the Control K (0B) character. It then checks to determine if there are any nulls at the end of the line. If there are no nulls then the line can be printed with no further justification required. It is already justified.

MULT multiplies the number of nulls by the character width provided by parameter CMND+02. This gives the number of units that must be distributed throughout the line to provide left and right justification.

CENTER handles the Center Mode of justification. It bumps over the Control K character and divides the nulls by two so that the nulls will be evenly divided. It tests for an odd or even number of nulls using a BCC after the LSRZ which does the divide. If there are an even number of nulls, then it branches to SHIFT. IF there are an odd number of nulls, it picks up the character width from CMND+2, divides this two to get a one-half character offset to provide more accurate centering. This is output via the OFFSET routine.

SHIFT moves the printer to the start of the centered line by outputting spaces equal to one-half the original number nulls. When finished it branches to NEXT which takes care of printing the text.

DIVIDE allocates the excess units along the line of text to produce the Full Justification. It first tests to see if it can allocate an additional unit to each individual character and space. If so, it increments both the character offset counter (COFSET) and the space offset counter (SOFSET). It then tests whether another unit can be allocated, until it finds that there are fewer units to be allocated than characters and spaces.

DIVDON takes care of any units remaining after the DIVIDE allocation. These are divided among the spaces, incrementing SOFSET until there are fewer units than spaces. The remainder, if any, is stored in EXCESS where it will be used on spaces starting at the beginning of the line.

NEXT handles the printing. It picks up and examines the next character. It branches to BOLD, SPACE, CHAR, or returns to the calling program if a null is encountered.

CHAR outputs the character using the system subroutine, in this case the KIM OUTCH subroutine. It tests for last character and puts out the character offset (COFSET) if non-zero.

OFFSET saves the offset in X, then puts the Diablo printer into PLOT mode by outputting a 10 hex. It then puts out one 'H' for each unit of offset, and finally returns the printer to TEXT mode by printing a 1C hex.

SPACE outputs a space, then combines a unit of EXCESS with the space offset and goes to NTEST to output the offset if not zero.

BOLD converts a Control X to '6' or a Control Y to '7', and then outputs the character after issuing an escape 1B hex. This sets or clears the print enhancement mode.

The DIRECT TYPESETTER

One use of JUSTIFY has been in a HELP program for direct typesetting. In this program a sheet of paper is inserted sideways in the terminal. Material is entered and edited on the left side of the page and typeset on the right side.

The CPRINT Function outputs a Control Comma (CTLCMA) 1C hex which sets the printer in TEST mode, and then issues a Carriage Return (CR) 0D hex.

The INPUT Function accepts data from the terminal, places it in the buffer defined by FILE (starts at 1780 and is 39 decimal characters long), and supports some editing features.

The next CPRINT causes the printer to TAB to the right side of the page, to the left margin of the typesetting area.

JUSTFY does the actual justification and printing. Its parameters specify that the set line has a maximum width of 39 decimal characters; that the width of each character is 10 units; and the 1E is a pointer to the start of the buffer - FFILE.

The last CPRINT sets the printer back one horizontal unit to provide a closer line spacing.

The BRANCH simply returns control to NEXT and the system is ready for the next line to be input.

DIRECT TYPESETTER - 16 Jan 1978

```
0004  0B1C010D  1  NEXT  CPRINT  CTLCMA  1    CR     TEXT MODE, CARRIAGE RETURN
0008  081C0080  2        INPUT   FILE    0    80     CLEAR AND INPUT TEXT
000C  0B090100  3        CPRINT  TAB     1    0      TAB TO TYPESET AREA
0010  01270A1E  4        JUSTFY  39.     10.  1E     39 CHAR WIDTH, 10 UNITS PER CHAR
0014  0B10014E  5        CPRINT  CTLP    1    "N     PLOT MODE, UP ONE UNIT
0018  03010000  6        BRANCH  NEXT                READY FOR NEXT LINE
001C  20008017  7  FILE  FMAP            FFILE       BUFFER AT 1780
0020  00270000  8  FMAP  00.     39.                 FIELD STARTS OFFSET 0, 39. CHAR.
```

TERMINAL INTERFACE MONITOR (TIM) FOR THE 6502 MICROPROCESSOR FAMILY

Oliver Holt
Old Nashua Road
Amherst, HN 03031

TIM is a unique monitor program for the 6500 microprocessor family. TIM is the forerunner to KIM and is still used today in many configurations--ready made and homebrew. TIM is supplied by MOS Technology on a MCS6530 multi-function chip. This chip contains ROM, RAM, an interval timer, and I/O. Using this chip, MOS Technology was able to squeeze the complete monitor function into a single IC. The 1K of ROM in the 6530 contains the monitor program; the 64 bytes of RAM are used for storage and vector interrupt addresses; the timer is used for timing the serial I/O; the 13 I/O lines are used to communicate with a serial I/O device and a parallel device. The TIM part number is MCS6530-004.

TIM has a couple of unique features not incorporated in most monitors. The first feature is the ability to reconfigure the TIM memory locations during resets. During reset all I/O lines on the 6530 are set up as inputs and look like high signals to external devices. One of these I/O lines is used with address line A15 to make A15 a "don't care" condition. 6500 type microprocessors fetch the reset vector address from FFFC and FFFD. Because A15 is a "don't care", the vector address is fetched from 7FFD instead of FFFC and FFFD. Locations 7FFC and 7FFD contain the TIM entry point for a reset condition.

Figure 1 is a block diagram of a minimum TIM-based system including the circuitry required to accomplish the reset operation. The I/O line used is PB4. This signal is inverted and NANDed with A15. During reset PB4 is high making $\overline{PB4}$ low. A low input to the NAND gate causes a high output, always enabling CS1 on the 6530. When the I/O) ports are initialized in the reset service routine, PB4 goes low making PB4 a high. Now the output of the NAND gate is $\overline{A15}$ and CS1 is only high when A15 is low. CS1 along with the other chip selects and the address lines give the 6530 a set of unique addresses below 8000 but the software is set up for the address space between 7000 and 73FF.

The other unique feature of the TIM is that the terminal interface speed is adaptive. After the system is reset, the user types a carriage return. TIM measures the terminal speed using the data stream generated by the carriage return signal. This speed information is stored and and used as the terminal speed for all following communication with the external device until the next time the system is reset.

After the reset and carriage return, TIM responds with an "*" and prints the contents of the registers, followed by an automatic carriage return and a ".". The period indicates that TIM is now ready to accept user commands. TIM commands allow displaying registers, executing programs, examining and altering memory, reading hexadecimal data from either a high speed reader or a TTY and writing either hexadecimal or BNPF data to a TTY. (BNPF is a tape format used by some of the older PROM programmers).

Using the BRK instruction the user can set up breakpoints to monitor the execution of a program. The user inserts a BRK instruction (00) where the breakpoints are required. Upon execution of BRK instruction TIM is entered and the registers are printed. The vector address for a BRK instruction is stored in RAM at FFFE and FFFF. The user may alter these locations and write his own routine for handling debug operations.

All TIM operations are performed in hex unless a BNPF tape is required. The memory is displayed in hex in groups of eight memory locations as shown:

```
.M 0000 00 01 02 03 04 05 06 07
command address data
```

TIM will respond with a period "." after each command is completed. If a user wants to modify data, he first opens memory with the "M" command and then types a colon ":" as follows: (Underlined data is what the user types).

```
.M 0000 00 01 02 03 04 05 06 07
.: 0000 00 01 25 03 99 (carriage return)
```

The carriage return terminates the operation. The 6500 registers may be examined:

```
.R 7052 31 27 F0 01 FF
    PC  P  A  X  Y  SP
```

After the registers have been opened for examination, they may be changed using the colon ":" as shown:

```
.R 7052 31 27 F0 01 FF
.: 0100 00 00 00 00 FF (carriage return)
```

The other commands for reading and punching tapes operate in a similar manner. TIM also has a switch which is set by the "H" command that specifies whether or not a high speed reader or TTY is the source of paper tape input.

TIM, like KIM, also has many useful subroutines that can be called by a users program. A set of useable subroutines to type characters, read characters, type a line feed and carriage return, type a space, and to type a byte in hex are completely documented in the TIM manual. There are other subroutines that can be used that are not documented and these include double precision addition, output a bit, input a bit, ASCII conversion, and input eight bits.

The TIM manual contains a complete software listing and a memory test program. The manual also includes example programs to aid the user in becoming familiar with the TIM commands. TIM is a very useful building block for anyone interested in building their own 6500 system. It has been used as the monitor for a number of systems available in kit and/or assembled versions. These include the CGRS Microtech 6000 system, the DATAC 1000, and others.

If you are interested in building your own homebrew system, Figure 1 is a block diagram for a basic system. TIM is available from MOS Technology representatives.

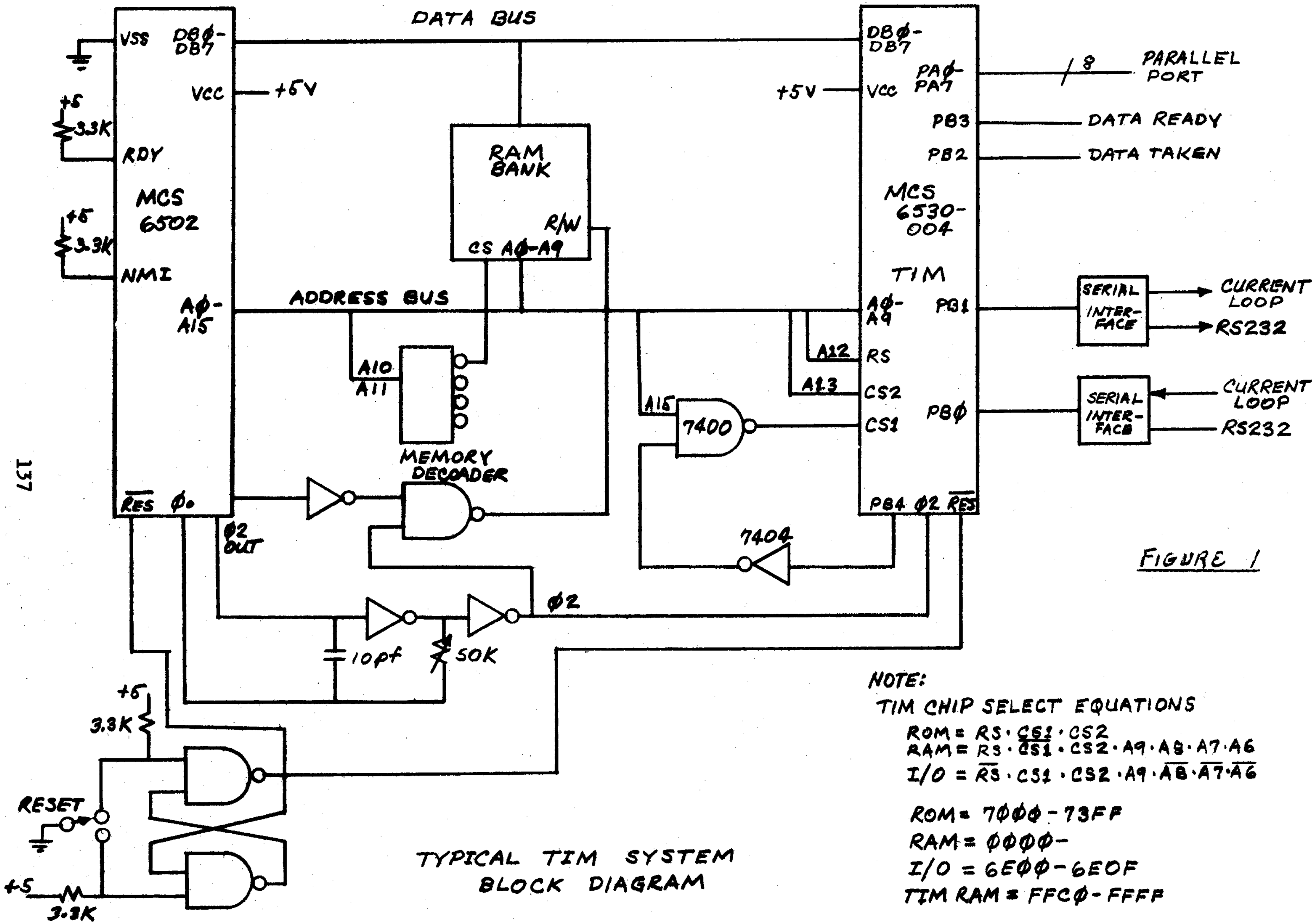


FIGURE 1

TYPICAL TIM SYSTEM BLOCK DIAGRAM

NOTE:
TIM CHIP SELECT EQUATIONS

ROM = RS · CS1 · CS2
RAM = RS · $\overline{CS1}$ · CS2 · A9 · A8 · A7 · A6
I/O = $\overline{RS}$ · CS1 · CS2 · A9 · $\overline{A8}$ · $\overline{A7}$ · $\overline{A6}$

ROM = 7ØØØ - 73FF
RAM = ØØØØ -
I/O = 6EØØ - 6EOF
TIM RAM = FFCØ - FFFF

TIM MEETS THE S100 BUS

Gary L. Tater
7925 Nottingham Way
Ellicott City, MD 21043

Hardly a computer meeting goes by without a discussion of which bus structure is best. While the S100 bus may not be optimum for the 6502 microprocessor, its use does make purchasing RAM and ROM boards easy.

With this in mind, I purchased a 6502 CPU board for the S100 bus from CGRS Microtech. This CPU board is almost a complete system with its onboard 2K RAM and 4K ROM. But in order to use my CT-64 Southwest Technical Products video terminal with this CPU, I needed an S100 terminal interface monitor (TIM) board. While CGRS markets a very nice TIM board, I elected to build a bare bones S100 TIM board which is described in this article.

In addition to serving as a serial I/O port for a terminal, TIM contains an operating system for 6500 microcomputers. The OCT-NOV issue of MICRO (page 5) contains an article on the operation of the TIM program. In summary, TIM is a read-only memory and I/O device that is self adapting to terminal speeds between 10 - 30 cps. With TIM you can display and alter CPU and memory location using a keyboard and video display; you can read and write hex formatted data from a paper tape or a cassette interface such as the Southwest Technical Products AC-30; and you have an eight bit parallel I/O port where each bit of the eight can be programmed as either input or output.

As you can see from the schematic diagram (Figure 2), only the TIM chip (6530-004) and four integrated circuits are needed, excluding voltage regulators. For the perfectionist, buffering could be added to the address lines, data lines, and parallel output port, but two CGRS Microtech systems are now successfully using this TIM design. Integrated circuits U2 and U3 are used during resets to reconfigure TIM memory locations as described in the previously referenced TIM article. The MC 1488 and MC 1489 are Motorola devices which convert TTL levels to RS 232 levels and RS 232 levels to TTL respectively.

A memory map of this TIM design is provided in Figure 1. For proper operation of a 6502 microprocessor and this TIM board, you will need both page zero and page one memory. Page one is needed by the 6502 microprocessor for its software stack. Page zero memory is used in the TIM program to store the baud rate of your terminal (locations 00EA and 00EB).

To operate a TIM based system you need only momentarily ground pin 16 of TIM (pin #75 of the S100 bus) using a switch on your front panel. After you send a carriage return to the computer, you should see a TIM message such as:

7052 30 2E FF 01 FF

This message contains first the program counter (7052), processor status register (30), accumulator (2E), X register (FF), Y register (01), and stack pointer (FF). The actual values will vary from machine to machine.

7000 - 73FF	TIM ROM
FFC0 - FFFF	TIM RAM
6E00 - 6E0F	TIM I/O
6E02	Serial Port

Figure 1
TIM Board Memory Map

If you have a problem, first check all of your wiring and the +5, +12, and -12 voltages. Then insure that your reset switch is controlling pin 16 of TIM. Next, using an oscilloscope, check for a carriage return character at pin 25 of TIM and pin 24 for the TIM message. With a good signal at pin 25 but no answer at pin 24, the last two things to check are the address lines including pin 21, PB4, and finally, check your TIM chip in a working system. The two systems built using this design on prototype boards came up immediately. Hopefully, you will have the same good fortune.

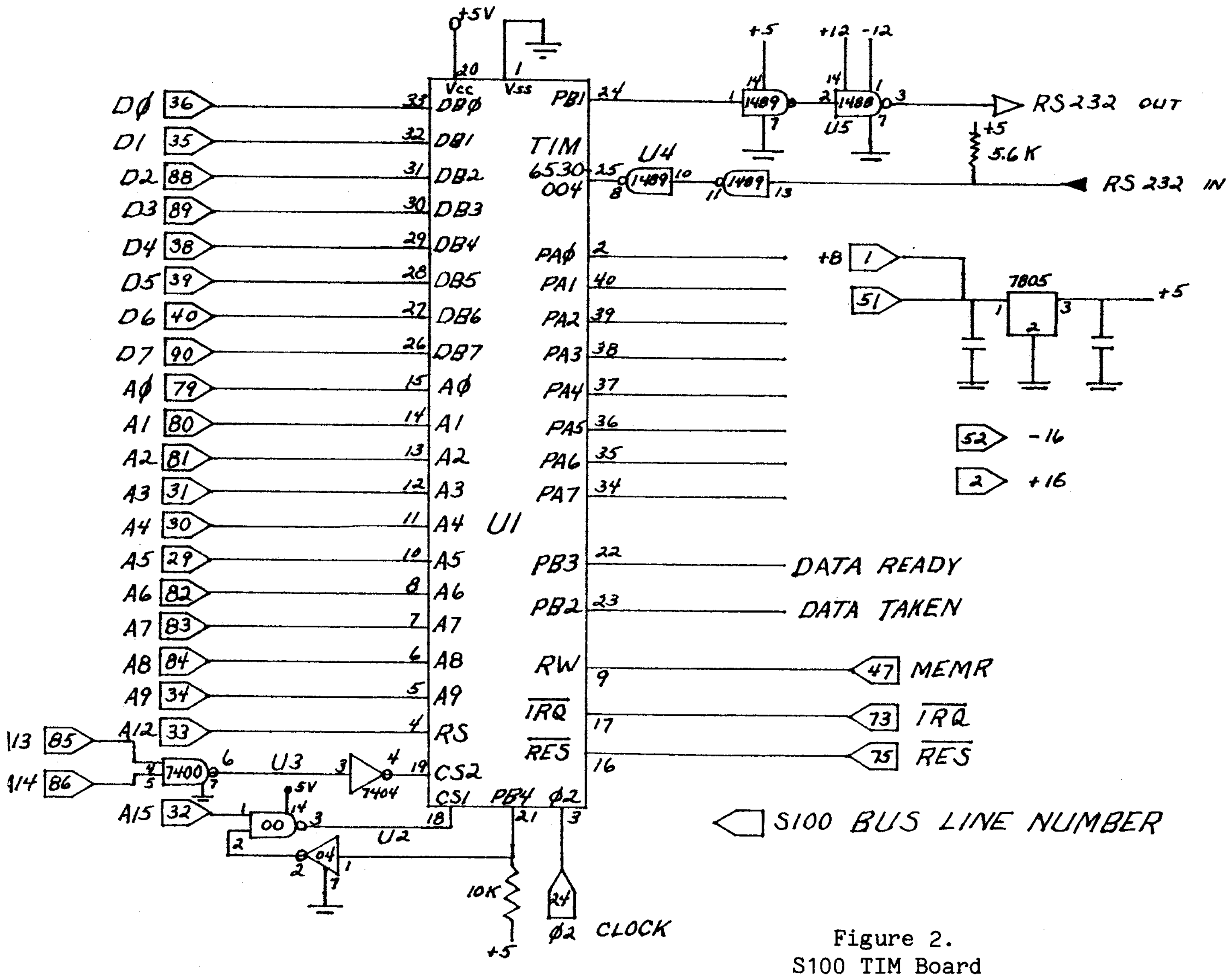


Figure 2.
S100 TIM Board

THE CHALLENGE OF THE OSI CHALLENGER

Joel Henkel
Old County Road
Hillsborough, NH 03244

One of the factors that a purchaser of a microcomputer system must consider is the degree of "do it yourself" hardware and software effort he will have to exert to get his system doing what he wants. This effort, not evident from manufacturers' literature, can be critical for user satisfaction, as became evident in our experience with the OSI Challenger. These notes evaluating the Challenger may be helpful to prospective purchasers.

In any hobby industry, user skills are assumed. This is emphasized for microcomputer firms that formerly catered to electronic kit builders. OSI is one of these, having supplied special PC board kits to hams. They follow their own packaging philosophy that differs from the "standard" S-100 bus configuration. Their brochure explains that the 100 pin S-100 connectors were rejected because the fingers were subject to poor contact. Instead, OSI uses MOLEX connectors, which make positive contact. The brochure goes on to describe the rejection of on-board voltage regulators in favor of a self contained regulated power supply.

OSI circuit boards are larger than standard S-100 bus boards. This accommodates their design philosophy of packing many optional functions into one foil pattern. For example, their 430B I/O board supports: an eight channel multiplexed eight bit analog to digital converter, two channels of eight bit digital to analog conversion and a UART controlled cassette I/O interface or an RS232/twenty mil loop I/O interface.

Our system came without keyboard or video monitor. Interfacing for these is left to the user. The computer cabinet has two holes in its rear panel for user implemented I/O cabling from individual boards. The keyboard DIP socket and video output RCA connector are available at the edge of the 440 video board. MOLEX connectors on the edge of the 430 board provides access to the various I/O options.

Hardware documentation consists of kit construction manuals for individual boards, even if the boards are purchased assembled. Various options are treated separately. Overall hardware system documentation is completely lacking. For example, nowhere is there a description of the bus convention and pinout. One must generate these from actual inspection of board foil patterns. This exercise reveals interesting peculiarities, such as bringing the NMI (non-maskable interrupt line) and IRQ (interrupt request line) onto many boards and leaving them unconnected.

The software is sophisticated. One enters the system by resetting. A prompter, D/M, comes up on the video screen. To enter the video monitor, styled after the KIM, enter M and the six hex digits appear near the top of the screen. For DOS (disc operating system), enter D and the DOS is brought up through BASIC by a bootstrap ROM. (Earlier versions required loading a short sequence of memory locations using the video monitor.) From BASIC one can enter the DOS, from which it is possible to go to various modules, such as an extended monitor, back to BASIC, or to activate a few DOS commands, such as loading and recalling disc files, executing programs, or switching floppy disc drives (for dual floppy discs). The EXTENDED BASIC by MICROSOFT has many advanced features, such as string functions, and is apparently much faster than a comparable 8080 BASIC.

Software documentation is poorly organized. Perhaps with so many possible options, the job of creating well organized system documentation was beyond OSI's capability. Our experience with software documentation availability was sobering. The system comes with all OSI software on discette. However, only a BASIC users manual is included, beyond a general system description. One has to order software user manuals separately. We waited a long five months after order for ours.

We have used two versions of the DOS, an original 1.1 version and an updated 2.0 version. One interesting change has to do with copying the DOS itself. The original version could not be copied and an explicit notice to that effect was included. An unfortunate set of circumstances could come about, however, that would wipe out track one, completely disabling any further loads of the DOS. If computer power fails (or one turns off the computer) with the disc in its drive, out goes track one! Apparently a number of users had this happen (including us). Version 2.0 has complete copying capability. According to instructions the first thing one should do is copy the DOS and store away one copy in case of wipeout.

Another change from the original version is the serial display output rate to the video monitor, which was increased from ten characters per second to several times that rate. A third change in the DOS is an augmented facility to read and write disc files.

The 440 board video display format chosen is twenty four characters per line, which is too small. One can only speculate on the reason for the short line.

Many applications could readily use a real time operating system, (RTOS). OSI does not offer a RTOS, but has advertised that one, modelled on DEC's RTS11 is in the works. When contacted recently, however, OSI reported that it has indefinitely postponed development of its RTOS in favor of development of a business system. The contemplated RTOS may explain the interrupt lines found in the foil patterns of several boards mentioned earlier, and a foil pattern option on the 470 floppy disc controller board, a real time clock in the form of a divider chain driven by the on-board crystal clock.

In summary, the OSI Challenger offers a lot of computer for the money. The tradeoff is the board orientation rather than system orientation, requiring a larger than average effort on the part of the user to bring his system up. This effort includes I/O interface cabling and "reading between the lines" in the supporting documentation.

MICRO Reviews: The First Book of KIM

This is one terrific book for anyone who has a KIM-1. It was assembled by Eric Rehnke (Publisher of "KIM-1/6502 User Notes"), Jim Butterfield ("Hypertape" and many other good utilities), and Stan Ockers (a regular "User Notes" contributor). Over half of the book is devoted to "Recreational Programs", games you can play on your basic KIM-1. The section on "Diagnostic & Utility Programs" is worth the price of the book by itself. The remainder of the book contains tutorial information on getting started with your KIM-1, expanding your system, and interfacing to the outside world. This well produced, 176 page resource is now published by Hayden Book Company and available at your computer book store for $9.00.

MICRO

ROCKWELL'S NEW R6500/1

Rockwell International
Electronic Devices Division
3310 Miraloma Avenue
P.O. Box 3669
Anaheim, CA 92803

ANAHEIM, CA., May 11, 1978 -- A single-chip NMOS microcomputer (R6500/1) operating at 2 MHz with a 1 microsecond minimum instruction execution time, has been developed by Rockwell Int'l.

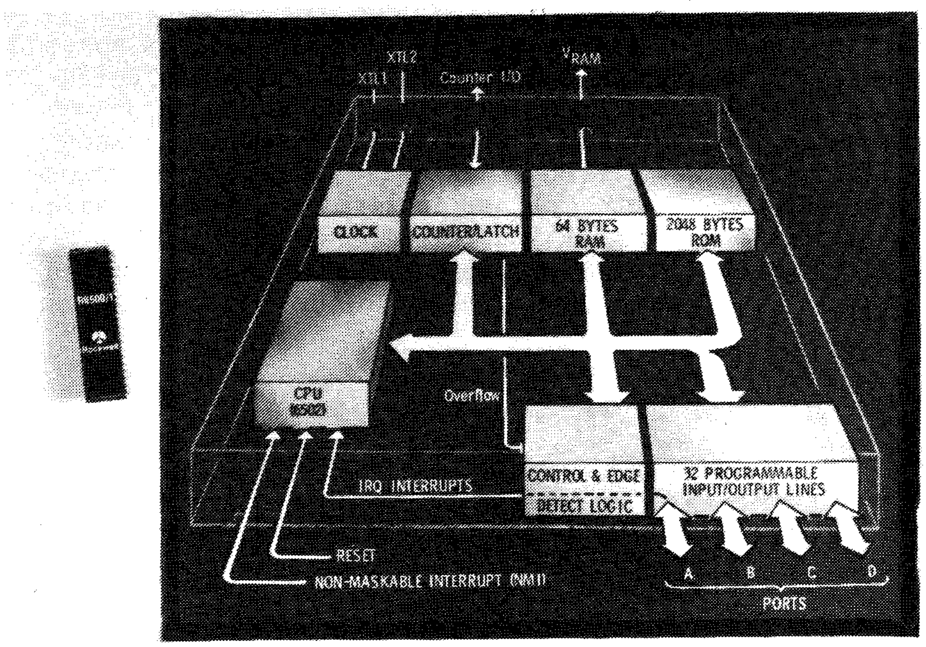
The 40-pin R6500/1 is fully software compatible with the 6500 family. It has the identical instruction set, including the 13 addressing modes, of the 6502 CPU. It operates from a single 5V power supply, and features a separate power pin which allows RAM memory to function on 10% of the operating power. On-chip features include 2K x 8 ROM, 64 x 8 RAM, 16-bit interval timer/event counter, and 32 bidirectional I/O lines. Additionally, it has maskable and non-maskable interrupts and an event-in/timer-out line.

The 32 bidirectional I/O lines are divided into four eight-bit ports (A, B, C and D). Each line can be selectively used as an input or an output. Two inputs to Port A can be used as edge sensing, software maskable, interrupt inputs -- one senses a rising edge; the other a falling edge.

Four different counter modes of operation are programmable: (1) free running with clock cycles counted for real time reference; (2) free running with output signal toggled by each counter overflow; (3) external event counter; and (4) pulse width measurement mode. A 16-bit latch automatically reinitializes the counter to a preset value. Interrupt on overflow is software maskable.

A 64-pin Emulator part, of which 40 pins are electrically identical to the standard R6500/1 part and which comes in either 1 MHz or 2 MHz versions, is available now. Rockwell expects to begin receiving codes from customers in July for production deliveries in Sept. Quantity prices for 6500/1 production devices are under $10.00 for both the 1 MHz and 2 MHz models. Single-unit prices for Emulator parts are $75.00 for the 1 MHz model and $95.00 for the 2 MHz version.

Contact: Leo Scanlon - 714/632-2321
Pattie Atteberry - 213/386-8600



ONE-CHIP SPEEDSTER -- Functional diagram of one-chip NMOS microcomputer (R6500/1) developed by Rockwell International. Fully software compatible with the 6500 family, the R6500/1 operates from a single 5V power supply at 2 MHz with a 1 microsecond minimum execution time.

ROCKWELL'S AIM IS PRETTY GOOD

Rockwell International
Microelectronic Devices
P.O.Box 3669
Anaheim, CA 92803
714/632-3729

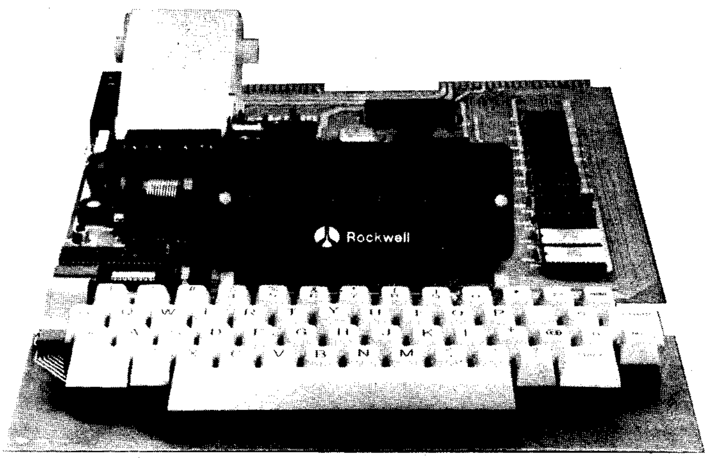
Rockwell's AIM 65 (Advanced Interface Module) gives you an assembled, versatile microcomputer system with a full-size keyboard, 20-character display and a 20-character thermal printer!

AIM 65's terminal-style ASCII keyboard has 54 keys providing 69 different alphabetic, numeric and special functions.

AIM 65's 20-character true Alphanumeric Display uses 16-segment font monolithic characters that are both unambiguous and easily readable.

AIM 65's 20-column Thermal Printer prints on low-cost heat sensitive roll paper at a fast 90 lines per minute. It produces all the standard 64 ASCII characters with a crisp-printing five-by-seven dot matrix. AIM 65's on-board printer is a unique feature for a low cost computer.

The CPU is the R6502 operating a 1 MHz. The basic system comes with 1K RAM, expandable on-baord to 4K. It includes a 4K ROM Monitor, and can be expanded on-board to 16K using 2332 ROMs or can also accept 2716 EPROMs. An R6532 RAM-Input/Output-Timer is used to support AIM 65 functions. There are also two R6522 Versatile Interface Adaptors. Each VIA has two 8-bit, bidirectional TTL ports, two 2-bit peripheral handshake control ports and two fully programmable interval timer/counters.



The built-in expansion capability includes a 44-pin Application Connector for peripheral add-ons and a 44-pin Expansion Connector with the full system bus. And, both connectors are totally KIM-1 compatible!

TTY and Audio Cassette Interfaces are part of the basic system. There is a 20 ma current loop TTY interface, just like the KIM-1, and an Audio Cassette Interface which has a KIM-1 compatible format as well as its own special binary blocked file assembler compatible format.

The DEBUG/MONITOR includes a mini-assembler and a text editor. Editing may use the keyboard, TTY, cassette, printer and display. The Monitor includes a typical set of memory display/modify commands. It also has peripheral device controllers, breakpoint capability and single step/trace modes of debugging. An 8K BASIC Interpreter will be available in ROM as an option.

AIM 65 will be available in August. It will cost $375.

```
(E)
EDITOR
FR=300   TO=1000
IN=
QWERTYUIOPASDFGHJ
JKLLZXCVBNMI
(I)
 0312          *=600
 0600   A2    LDX   #FE
 0602   E8    INX
 0603   D0    BNE   0602
 0605   EA    NOP
 0606   EA    NOP
 0607   4C    JMP   0600
 060A
(
```

SYNERTEK'S VIM-1

Synertek Incorporated
P.O. Box 552
Santa Clara, CA 95052

Synertek has announced a new 6502-based microcomputer system with the following features:

FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to use as soon as you open the box.

28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS.

EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY.

KIM-1 HARDWARE COMPATIBILITY.

The powerful 6502 8-bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.

THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user for timing loops, watchdog functions, and real-ime communication protocols.

4K BYTE ROM RESIDENT MONITOR and Operating Programs.

Single 5 Volt power capability is all that is required.

1K BYTES OF 2114 STATIC RAM on-board with sockets provided for immediate expansion to 4K bytes on-board, with total memory expansion to 65,536 bytes.

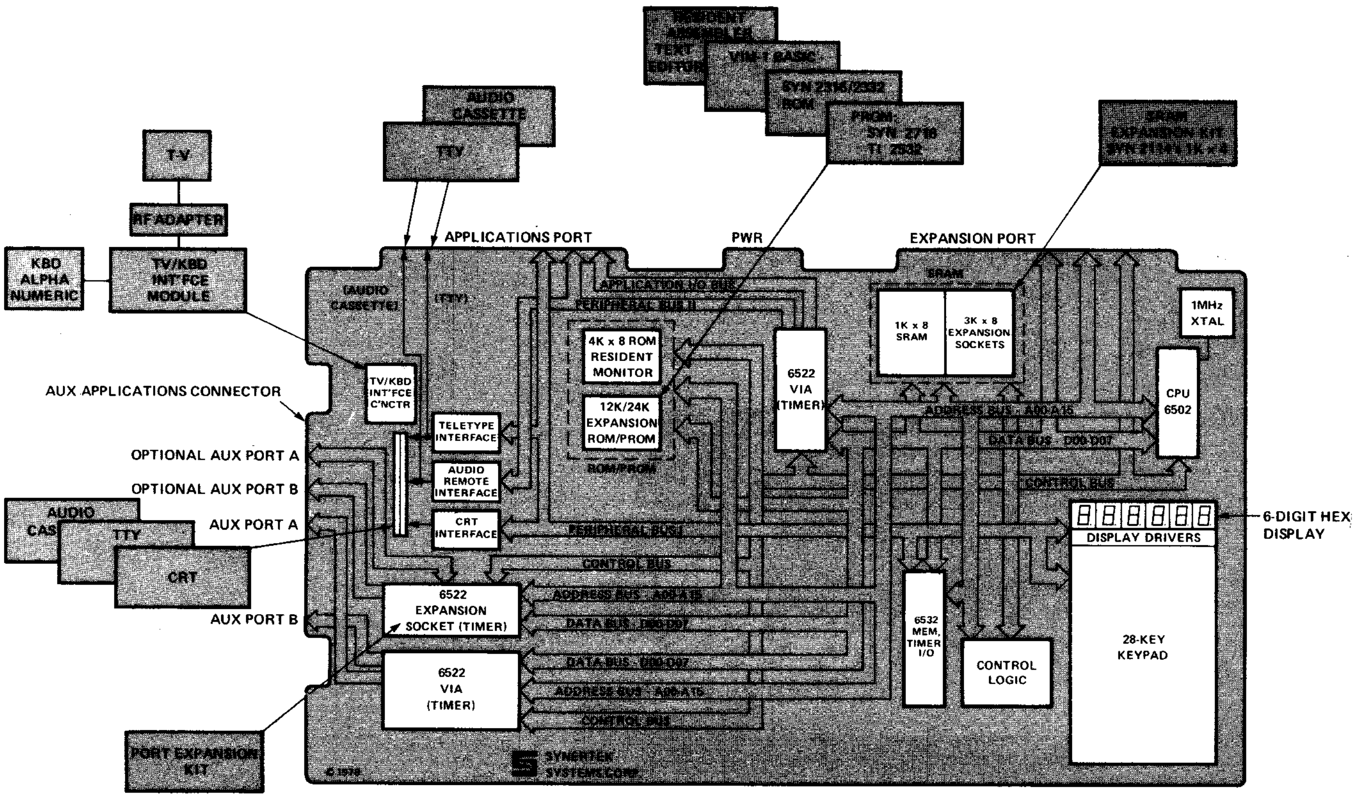
USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs.

ENHANCED SOFTWARE with simplified user interface.

STANDARD INTERFACES INCLUDE:
- Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1 compatible, Hi-speed 2400 Baud).
- Full Duplex 20mA Teletype Interface
- System Expansion Bus Interface
- TV Controller Board Interface
- CRT Compatible Interface

APPLICATION PORT: 15 Bi-directional TTL lines for user applications with expansion capability for added lines.

EXPANSION PORT FOR ADD-ON MODULES (50 I/O Lines in the basic system).



THE MICRO SOFTWARE CATALOG

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

As a service to the 6502 community, MICRO will publish a continuing catalog of software available for 6502 based systems. The source of this information will normally be the authors or distributors of the software. Since there is only a limited amount of space which can be devoted to this effort, there will be some restrictions placed on what is published. To qualify for inclusion in the catalog the software must be currently available, should have been sold (or given) to at least twenty-five customers, must be of general interest, and must be significant. "Significant" means that the program is not just a short utility which could be presented as a one-page article in a magazine, or a simple game, etc. The intent of the catalog is not to promote everyone selling everything, but rather to highlight the important software packages which do exist.

Publication of information about any software in this catalog does not imply anything about its worth, capabilities, documentation, etc. We depend on the information supplied to us. We will not knowingly include any software that is not worthy, and we reserve the right to publish additional information about these products - be it good or bad - that we receive from our readers or any other valid source.

It is easy to get your package listed. Just write to the above address and provide the information required as shown in the listings below. Please write your own "description". If we have to write the description from general information you provide, we may miss points which you think are important and emphasize things you think are trivial. Also, material which is presented in the proper form will normally get priority over other material.

Name: ASSM/TED
System: Preconfigured for TIM
Can be modified for other systems.
Memory: 4K RAM
Language: Assembler
Hardware: CRT and Keyboard, tapes and printer optional.
Description: A resident Assembler/Text Editor. Syntax very similar to MOS Technology. Produces relocatable object code on tape and can store directly executable code in memory during assembly. Programs can be assembled from memory of tape. Includes 17 operating commands and 16 pseudo ops. Editor has auto line numbering, file formating, and a manuscript feature.
Copies: Information not provided.
Price: $25.00
Includes: Hex Dump of ASSM/TED and Relocating Loader, and Operators Manual. No tape provided.
Ordering Info: Specify memory limits: 0200-1200, 0400-1400, 1000-2000, or 2000-3000. Select one.
Author: C. W. Moser
Available from:
C. W. Moser
3239 Linda Drive
Winston-Salem, NC 27106

Name: COSMAC 1802 Simulator
System: KIM-1
Memory: Less than 1K RAM
Language: Assembler
Hardware: Basic KIM-1
Description: Permits the KIM-1 to simulate the COSMAC 1802 by executing its instruction set. The simulator does this by interpretting the COSMAC instructions in a normal program sequence and making all internal COSMAC registers available for examination at any time. They may be viewed statically in a single step mode or dynamically in a trace mode. All COSMAC software features are supported with the exception of DMA.
Copies: Just released. Will be discussed in an article in Kilobaud.
Price: $10.00
Includes: KIM-1 cassette tape, user manual, and complete source listing.
Ordering Info: None required
Author: Dann McCreary
Available from:
Dann McCreary
4758 Mansfield St, #2M
San Diego, CA 92116

Name: PLEASE
System: Basic KIM-1
Memory: Basic KIM-1 memory
Language: Assembler/PLEASE
Hardware: Basic KIM-1
Description: A collection of games and demos. Includes a 24 hour clock, HiLo game, Mastermind, Shooting Stars, Drunk Test, Reaction Time Tester, Adding Machine, and more. Written in a "high-level" language - PLEASE. Permits the user to modify and create his own programs. Let's you show off your KIM-1, and teaches you how to use it.
Copies: Over 800 have been sold
Price: $15.00
Includes: Operators manual, complete source listings, PLEASE language description, with object code on Hypertape.
Ordering Info: None
Author: Robert M. Tripp
Available from:
The COMPUTERIST
P.O. Box 3
S. Chelmsford, MA 01824

Name: Micro-ADE
System: KIM-1 (easily modified for use with other 6502 based systems)
Memory: 8K RAM or 4K EPROM + 4K RAM
Language: Assembler
Hardware: Terminal - CRT or TTY, cassette units optional
Description: A combination Assembler, Editor, and Disassembler. Uses MICRO 6502 syntax. With automatic cassette controls, any length file may be edited and assembled. Object files may be automatically dumped to cassette and for short programs may be dumped to and executed from memory. Includes many useful commands for handling cassettes, moving data in memory, and so forth.
Copies: Hundreds
Price: $25.00 without source listings
$25.00 for source listings
Includes: Extensive user manual which includes source listings for the I/O to permit user modification. Object on Hypertape cassette.
Ordering Info: Specify with or without the optional source listings.
Author: Peter Jennings
Available from:
Micro-Ware Ltd.
27 Firstbrooke Road
Toronto, Ontario
Canada M4E 2L2

The COMPUTERIST
P.O. Box 3
S. Chelmsford, MA 01824

Name: The 6502 Program Exchange
System: TIM and KIM-1
Memory: Depends on Program
Language: Assmebler, BASIC, FOCAL
Hardware: Depends on Program
Description: A large collection of programs for 6502 based systems. These include utilities, games, subroutines, an assembler, editor, and a high level language: FOCAL.
Copies: Few to Many depending on the particular program.
Price: Depends on program. Many are based purely on number of pages of code. Major packages are priced separately.
Includes: Normally includes source listings, documentation, sheets of sample run, and paper tape. KIM-1 cassettes at no additional charge if user supplies cassettes.
Ordering Info: Write for catalog.
Author: Many different authors.
Available from:
The 6502 Program Exchange
2920 Moana
Reno, NV 89509

Name: Personal Savings Investment
Loan Repayment
Direct Reduction Loan Info.
System: APPLE II
Memory: At least 16K
Language: APPLESOFT BASIC
Hardware: Standard APPLE II
Description: Three separate programs. PSI - compute future value of your investments; monthly amount needed to get to a certain goal at a certain time. LP - determine monthly payments for a car, house or other type of load. DRLI - find the total interest paid and remaining balance is for a loan.
Copies: Over 25 combined
Price: $3.75 (including handling) each of the three programs.
Includes: Object on cassette tape. A listing of the program and examples of program useage.
Ordering Info: Specify which program.
Author: Les Stubbs
Available from:
Les Stubbs
23725 Oakheath Place
Harbor City, CA 90710

Name: TINY BASIC
System: KIM, TIM, Jolt, Apple I
Memory: Minimum of 3K
Language: Assembler
Hardware: User defines I/O
Description: TINY BASIC is a subset of regular BASIC, limited to 16-bit integer arithmetic [+, -, *, /, ()]. There are 26 variables (A-Z), no stirngs and no arrays. The following commands are functional: LET PRINT INPUT IF-THEN GOTO GOSUB RUN LIST CLEAR RETURN REM END. TINY BASIC does not contain any I/O instructions; three JMPs link TINY to the user's I/O routines. These are well documented in the manual.
Copies: "Several hundred 6502 version"
Price: $5.00
Includes: 26 page User Manual and a paper tape in standard hex loader format. Hex Dump may be substituted upon request for paper tape.
Ordering Info: Specify version:
TB650K (0200-0AFF) KIM, TIM, ...
TB650J (1000-18ff) Jolt
TB650T (2000-28FF) KIM with 4K RAM
Author: Tom Pittman
Available from:
ITTY BITTY COMPUTERS
P.O. Box 23189
San Jose, CA 95153

Name: HELP Mailing List Package
System: Basic KIM-1
Memory: Basic KIM-1
Language: Assembler/HELP
Hardware: Terminal, Cassettes, Relays
Description: A complete package for creating, maintaining, and printing mailing list information. A high speed cassette routine reads/writes at 800 baud (twelve times the KIM-1 rate) and can store about 900 names on one side of a 60 minute tape. Selective printing of mailing list. This package is used to maintain the MICRO mailing list This package is written in HELP, a "high-level" language which makes it easy to customize the package for your own requirements.
Copies: Over 100
Price: $15.00
Includes: An extensive user manual, a detailed discussion of the HELP language, and complete source listings. Object on Hypertape.
Ordering Info: None
Author: Robert M. Tripp
Available from:
The COMPUTERIST
P.O. Box 3
S. Chelmsford, MA 01824

Name: ASM/TED
System: KIM-1 (may be modified for use with other 6502 based systems)
Memory: 6K RAM
Language: Assembler
Hardware: TTY
Description: The text editor performs line editing in RAM and can dump/load to paper tape or audio cassette. The resident assembler is single-pass using the standard MOS Technology syntax. Source code may be paper tape or memory resident and object code is always to memory.
Copies: Information not provided.
Price: $70.00
Includes: 50 page manual, source listings, and object on KIM cassette or paper tape.
Ordering Info: Send $2.00 for current catalog of available software.
Author: Not specified
Available from:
ARESCO
450 Forest Ave., Q-203
Norristown, PA 19401

Name: MicroChess
System: Basic KIM-1
Memory: Basic KIM-1
Language: Assembler
Hardware: Basic KIM-1
Description: Plays a reasonably good game of chess on a basic KIM-1. Has programmed openings. User enters his move via the KIM keypad and the KIM Display shows the move. The computer then makes its move and displays it. Program may be set to play at different speeds: 3, 10, or 100 seconds per move average. A great way to demo your KIM.
Copies: Hundreds
Price: $10.00 without cassette
$15.00 with cassette
Includes: Operator's manual, source listings, and a detailed discussion of the operation of the program. Object on cassette tape optional.
Ordering Info: Specify tape or not.
Author: Peter Jennings
Available from:
Micro-Ware Ltd.
27 Firstbrooke Road
Toronto, Ontario
Canada, M4E 2L2

The COMPUTERIST
P.O. Box 3
S. Chelmsford, MA 01824

THE MICRO SOFTWARE CATALOG: II

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: ZZYP-PAX for PET, #1,2, and 3
System: PET
Memory: 8K RAM
Language: BASIC
Hardware: Standard PET
Description: Each of these three ZZYP-for PET includes a cassette with two games and a booklet designed to educate the beginning or intermediate level PET programmer. #1 has IRON PLANET (Rescue the Princess) and HANGMAN (Guess the secret word). Included is a 12 page booklet which not only contains game rules, but has 5 pages of useful programming techniques including: Direct Screen Access Graphics, Flashing Messages, and Programmed Delays. #2 contains BLACK BART (a mean-mouthed poker player) and BLACK BRET (for blackjack - one or two players). #3 contains BLOCK and FOOTBALL both of which allow either two-player or play-the-PET options.
Copies: Just released, 40 copies.
Price: $9.95 each
Includes: PET tape cassette, instructions and educational manual with info for program modifications.
Ordering Info: Specify ZZYP-PAX number
Author: Terry Dossey
Available from:
Many PET dealers, or,
ZZYP Data Processing
2313 Morningside Drive
Bryan, TX 77801

Name: BULLS AND BEARS (tm)
System: Apple II
Memory: 16K
Language: 16K BASIC
Hardware: Apple II
Description: A multi-player simulation of corporate finance. Involves decision-making regarding production levels, financing, dividends, buying and selling of stock, etc.
Copies: "Hundreds sold"
Price: $12.00
Includes: Game cassette and booklet.
Ordering Info: At computer stores only
Author: SPEAKEASY SOFTWARE LTD.
Box I200
Kemptville, Ontario
Canada KOG 1J0

[Dealer inquiries invited]

Name: A Variety of Programs
System: Apple II
Memory: Most 8K or less
Language: Mostly Integer BASIC
Hardware: Mostly standard Apple II
Description: A varied collection of short programs. Some utilities, some educational. Included are: ALPHA SORT MUSIC ROUTINE, STOP WATCHBASIC DUMP, MULTIPLY, ONE-ARM-BANDIT, ...
Copies: Varies, up to about 20.
Price: $7.50 to $10.00 each.
Includes: Apple II cassette and program listing.
Ordering Info: Write for catalog.
Author(s): Not specified.
Available from:
Apple PugetSound Prog. Lib. Exch.
6708 39th Avenue SW
Seattle, WA 98136

Name: HELP Information Retrieval
System: KIM-1
Memory: Basic KIM-1
Language: Assembler and HELP
Hardware: KIM-1, terminal, cassettes
Description: Permits the user to create a data base on cassette, and then perform a variety of searches on the data base. May make six simultaneous tests on FLAGS associated with the data plus one test on each of the six data fields. Permits very complex retrieval from the data base. Includes ULTRATAPE which reads/writes at 100 char/sec, 12 times the normal KIM rate.
Copies: 100+
Price: $15.00
Includes: Cassette tape, 36 page User Manual, a Source Listing book and a Functions Manual which explains the operation of the HELP language.
Ordering Info: Specify HELP Info Ret.
Author: Robert M. Tripp
Available from:
Many 6502 Dealers, or,
The COMPUTERIST, Inc.
P.O. Box 3
S. Chelmsford, MA 01824

THE MICRO SOFTWARE CATALOG: III

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: LABELER
System: TIM based or any 6502 based system
Memory: 1K
Language: Assembly
Hardware: Paper Tape Punch on TTY
Description: This program punches legible characters on a paper tape and is useful for the labeling of punched paper tapes. A 64 character sub-set of ASCII is used. There is limited editing capability on the data. There are a number of options for character size, starting address and TIM or I/O independent code.
Copies: Not Specified
Price: $4.00
Includes: Commented source listing, operating and modifying instructions, and a hex tape.
Ordering Info: Specify the following:
 Char Size 5x5 or 5x8
 Starting address 0200 or 1000
 System TIM or I/O Independent
Author: Gil House
Available from:
 Gil House
 P.O. Box 158
 Clarksburg, MD 20734

Name: HUEY
System: Any 6502 based system.
Memory: 2.5K
Language: Assembly
Hardware: ASCII I/O device.
Description: HUEY-65 is a scientific calculator program for the 6502 microprocessors. It operates from your ASCII keyboard like a calculator; will output through your routines to a TV screen or Teletype; is preprogrammed to do trig functions, natural and common logs, exponential functions and other goodies; and is programmable for many other functions (financial, accounting, mathematics, engineering, etc.) you would like to call at the press of a single key.
Copies: Not Specified.
Price: Hex Dump at any even page - $5.00
 Manual and Listings - $20.00
Ordering Info: Specify starting address.
Author: Don Rindsberg
Available from:
 The BIT Stop
 P.O. Box 973
 Mobile, AL 36601

Name: Word Processor Program
System: PET
Memory: Not Specified.
Language: Not Specified.
Hardware: RS-232 printer addressed via a CmC printer adapter.
Description: This program permits composing and printing letters, flyers, advertisements, manuscripts, articles, etc., using the Commodore PET and an RS-232 printer. Script directives include line length, left margin, centering, and skip. Edit commands allow the user to insert lines, delete lines, move lines, change strings, save onto cassette, load from cassette, move up, move down, print and type.
Copies: Not Specified.
Price: $29.50
Ordering Info: None.
Author(s): Not Specified.
Available from:
 Connecticut microComputer
 150 Pocono Road
 Brookfield, CT 06804

Name: ZIP TAPE
System: KIM-1, may be easily modified for any other 6502 system with programmable timer I/O
Memory: 3/4 page each for read and write progs.
Hardware: Simple single IC audio to logic level converter and output buffer/attenuator on 2" sq. board. Directional control, 4 connections to computer.
Description: A fast audio cassette data recording and recovery system. Programmable to 4800 baud. Loads 8K in less than 15 seconds. Follows KIM-1 protocol of open ended record length with start address, end address, and record ID specified at usual KIM locations. Load by ID, ignore ID, and relocate modes. Data recorded in binary form with 2 byte checksum error detection. Easily relocated, can either stand alone or be used as subroutines. Requires programmable timer I/O.
Copies: About 12, just introduced.
Price: $22.50 +1.00 ship & hand. $3.00 extra for KIM cassette.
Includes: Assembled and tested interface, commented listings, suggested changes to run on TIM and other systems. Cassette has software recorded at HYPERTAPE and standard KIM speeds plus 8K test recording using ZIP TAPE.
Ordering Info: With or Without tape.
Author: Lewis Edwards, Jr.
Available from:
 Lewis Edwards
 1451 Hamilton Avenue
 Trenton, NJ 08629

Name: FOCAL* (*DEC Trademark)
System: Apple II
Memory: Not Specified.
Language: Assembler
Hardware: Apple II
Description: This is an extended version of the high-level language called FOCAL. FOCAL was created for the DEC PDP-8. It is similar to BASIC. FCL65E, as this version is called, is now available for the Apple II.
Copies: Not Specified.
Price: Apple II format cassette - $25.00
 Mini-Manual - $6.00
 FCL65E User's Manual - $12.00
 Complete Source Listing - $35.00
Ordering Info: Specify parts desired.
Author(s): Not Specified.
Available from:
 The 6502 Program Exchange
 2920 Moana
 Reno, NV 89509

Name: WARLORDS
System: Apple II (PET version under devel.)
Memory: Not Specified
Language: Not Specified
Hardware: Apple II
Description: It is the Dark Ages, in the kingdom of Nerd, and all is chaos. King Melvin has died without an heir and a dire power struggle is taking place to see who will emerge as the new King. You and the other players are the WARLORDS, and you will have to decide what combination of military might and skillful diplomacy will lead you to victory.
Copies: Not Specified
Price: $12.00
Ordering Info: Specify Apple II Version
Author: Not Specified
Available from:
 Dealers who carry software from
 Speakeasy Software LTD.

Names: E/65 and A/65
System: Any 6502 based system
Memory: Not Specified
Language: Assembly
Hardware: Terminal. Cassette optional.
Description: E/65 is primarily designed to edit assembler source code. Line oriented commands specify input/out or text and find specific lines to be edited. String oriented commands allow the user to search for and optionally change a text string. Also character oriented commands and loading and dumping to bulk device. A/65 is a full two-pass assembler which conforms to MOS Technology syntax. A full range of run-time options are provided to control listing formats, printing of generated code for ASCII strings and generation of object code.
Copies: Not Specified
Price: $100 each
Includes: Object form on paper tape or KIM type cassette. Listings of source code are available for $25.00 each. Full documentation on the installation and use of each package is provided.
Author: Not Specified
Available from:

COMPAS - Computer Applications Corporation
P.O. Box 687
Ames, IA 50010

Name: Read/Write PET Memory
System: PET
Memory: 8K RAM
Language: BASIC
Hardware: Standard PET
Description: Permits user to key into memory hex codes by typing hex starting address and then typing the hex digits in sequence desired. Display memory as both hex codes and assembly language mnemonics (translates relative address into actual hex address). Stores memory on tape and loads memory from tape into any desired memory location. Executes machine-language programs.
Copies: Just released - 32 sold first day.
Price: $7.95 - postpaid
Includes: Cassette tape; complete instructions (including use of ROM subroutines to input and output memory from keyboard and to screen).
Ordering Info: From author
Author:

Don Ketchum
313 Van Ness Avenue
Upland, CA 91786

(Dealer Inquities Invited)

PROGRAMMING A MICRO-COMPUTER: 6502

by Caxton C. Foster

(Reviewed by James R. Witt, Jr.)

For those of you in the computing world who have recently purchased or constructed a microcomputer based on the 6502 microprocessor (the KIM-1 fits this description) and can't put it to reasonably practical use, then perhaps your headaches are over! Programming a Micro-Computer: 6502 by Caxton C. Foster may be exactly what you need to halt your frustrations. Foster presents the reader with a combination of reference manual for programming and an introduction to 6502 systems, specifically using the KIM-1 as a model.

The motivation behind Foster's work is practicality. Right from the beginning of the first chapter a hypothetical situation is introduced, circumstances that one might face in the course of an average day, and the microcomputer is suggested as a solution. Initially, a simple problem is introduced, a problem one would not expect a computer to solve due to its simplicity. Yet, this enables the reader to grasp the basic operation of running an uncluttered program successfully. Possible reasons as to why a certain program fails are provided to lessen confusion.

With successful completion of one program, the author wastes no time moving on to new situations. This may seem somewhat fast and confusing to those who greet micros as a totally new experience. Yet the situations do become more interesting and more challenging to solve by computer software. Such programs include: "Keybounce", "A Combination Lock", and "Digital Clock" among others. Several of these programs are completely legitimate and fully operable.

As noted before, Foster moves at a swift pace. At certain points, various instructions and KIM-1 anatomy are condensed into a mere page or two. Basic understanding of digital electronics is assumed often and may be required before fully digesting some of this material. These two minor weaknesses may tend to boggle the mind of the newcomer and hinder his comprehension of the purpose of programming and its make-up.

Suggestions: For those who are newcomers to the "sport" of computing and digital electronics, you may want to consider some other preliminary instructions BEFORE undertaking this book. If you have some sense of digital, but little knowledge of micros, you should tackle it, but should make notes of important items the first time through each chapter, and then reread the chapter to pull the odds and ends together. If you have written simple programs but have an appetite for more complex proglem-solving, then Programming A Micro-Computer: 6502 will be a definite aid and resource in satisfying your hunger.

Programming A Micro-Computer: 6502, by Caxton C. Foster, published by Addison-Wesley, 1978.

6502 INFORMATION RESOURCES

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

Did you ever wonder just what magazines were the richest sources of information on the 6502 microprocessor, 6502-based microcomputers, accessory hardware and software? For several years this writer has been assembling a bibliography 6502 references related to hobby computers and small business systems (see MICRO No's 1, 3, 4, 5, and 6). A review of the number of times various magazines are cited in the bibliography gives a rough measure of the coverage of these magazines of 6502 related subjects. Even after such a fequency chart is compiled, an accurate comparison is difficult. Some of the magazines have been published longer than others. Some periodicals have been discontinued, others have been merged with continuing publications. Some give a lot of information in the form of ads, others are devoted mostly to authored articles. Regardless of the basis of the tabulation of references, however, some publications are clearly more useful sources of information on the 6502 than others.

The accompanying list of magazines has been compiled from the bibliography. At the top of the list are several publications which specialize in 6502-related subjects. These include this publication, MICRO, as well as the KIM-1 /6502 USER NOTES. Also in this category is OHIO SCIENTIFIC'S SMALL SYSTEMS JOURNAL, a publication which covers hardware and software for the Ohio Scientific 6502-based computers. KILOBAUD, BYTE and DR. DOBB'S JOURNAL all give good coverage on the 6502 as well as other microprocessors. KILOBAUD has more hardware and constructional articles than most computer magazines. ON-LINE is devoted mainly to new product announcements and has very frequent references to 6502 related items. Following these come a group of magazines with somewhat less frequent references to the 6502. Finally toward the end of the list are those magazines with only occasional or trivial references to the 6502. An attempt has been made to give up-to-date addresses and subscription rates for the magazines cited.

MICRO
$6.00 per 6 issues
MICRO
P.O. Box 3
S. Chelmsford, MA 01824

KIM-1/6502 USER NOTES
$5.00 per 6 issues
Eric Rehnke
P.O. Box 33077
Royalton, OH 44133

OHIO SCIENTIFIC--SMALL SYSTEMS JOURNAL
$6.00 per year (6 issues)
Ohio Scientific
1333 S. Chillicothe Rd.
Aurora, OH 44202

KILOBAUD
$15.00 per year
Kilobaud Magazine
Peterborough, NH 03458

BYTE
$12.00 per year
Byte Publications, Inc.
70 Main St.
Peterborough, NH 03458

DR. DOBB'S JOURNAL
$12.00 per year (10 issues)
People's Computer Co.
Box E
1263 El Camino Real
Menlo Park, CA 94025

ON-LINE
$3.75 per year (18 issues)
D. H. Beetle
24695 Santa Cruz Hwy
Los Gatos, CA 95030

PEOPLE'S COMPUTERS (Formerly PCC)
$8.00 per year (6 issues)
People's Computer Co.
1263 El Camino Real
Box E
Menlo Park, CA 94025

INTERFACE AGE
$14.00 per year
McPheters, Wolfe & Jones
16704 Marquardt Ave.
Cerritos, CA 90701

POPULAR ELECTRONICS
$12.00 per year
Popular Electronics
One Park Ave.
New York, NY 10016

PERSONAL COMPUTING (Formerly MICROTREK)
$14.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

73 MAGAZINE
$15.00 per year
73, Inc.
Peterborough, NH

CREATIVE COMPUTING
$15.00 per year
Creative Computing
P.O. Box 789-M
Morristown, NJ 07960

SSSC INTERFACE
(Write for information)
Southern California Computer Soc.
1702 Ashland
Santa Monica, CA 90405

EDN (Electronic Design News)
$25.00 per year
(Write for subscription info)
Cahners Publishing Co.
270 St Paul St.
Denver, CO 80206

RADIO ELECTRONICS
$8.75 per year
Gernsback Publications, Inc.
200 Park Ave., South
New York, NY 10003

QST
$12.00 per year
American Radio Relay League
225 Main St.
Newington, CT 06111

IEEE Computer
(Write for subscription info)
IEEE
345 E. 47th St.
New York, NY 10017

ELECTRONICS
$14.00 per year
Electronics
McGraw Hill Bldg.
1221 Ave. of Americas
New York, NY 10020

POLYPHONY
$4.00 per year
PAIA Electronics, Inc.
1020 W. Wilshire Blvd.
Oklahoma City, OK 73116

CALCULATORS, COMPUTERS
$12.00 per year (7 issues)
Dynax
P.O. Box 310
Menlo Park, CA 94025

COMPUTER MUSIC JOURNAL
$14.00 per year (6 issues)
People's Computer Co.
Box E
1010 Doyle St.
Menlo Park, CA 94025

POPULAR COMPUTING
$18.00 per year
Popular Computing
Box 272
Calabasas, CA 91302

MINI-MICRO SYSTEMS
$18.00 per year
Modern Data Service
5 Kane Industrial Drive
Hudson, MA 01749

DIGITAL DESIGN
$20.00 per year
(Write for subscription info)
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

ELECTRONIC DESIGN
(26 issues per year)
(Write for subscription info)
Hayden Publishing Co., Inc
50 Essex St.
Rochelle Park, NJ 07662

HAM RADIO
$12.00 per year
Communications Technology
Greenville, NH 03048

COMPUTER WORLD
$12.00 per year (trade weekly)
(Write for subscription info)
Computer World
797 Washington St.
Newton, MA 02160

Editor's Note: In addition to the magazines regularly covered by the 6502 Bibliography, the following magazines may also be of interest to various 6502 readers:

PET GAZETTE
Free bi-monthly (Contributions Accepted)
Microcomputer Resource Center
1929 Northport Drive, Room 6
Madison, WI 53704

Robert Purser's REFERENCE LIST
OF COMPUTER CASSETTES
Nov 1978 $2.00/Feb 1979 $4.00
Robert Purser
P.O. Box 466
El Dorado, CA 95623

THE SOFTWARE EXCHANGE
$5.00 per year (6 issues)
The Software Exchange
P.O. Box 55056
Valencia, CA 91355

THE PAPER
$15.00 per year (10 issues)
The PAPER
P.O. Box 43
Audubon, PA 19407

PET USER NOTES
$5.00 per year (6 or more issues)
PET User Group
P.O. Box 371
Montgomeryville, PA 18936

CALL A.P.P.L.E
$10.00 per year (includes dues)
Apple Puget Sound Program Library Exchange
6708 39th Ave. SW
Seattle, WA 98136

6502 BIBLIOGRAPHY

William Dial
438 Roslyn Avenue
Akron, OH 44320

1. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "KIM-1 Microcomputer Module Users Manual (1975)"
2. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "6502 Programming Manual"
3. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "6502 Hardware Manual"
4. Fylstra, Daniel, "Son of Motorola (or the $20 CPU chip)" Byte 1 No. 2, pp. 56-62 (November 1975)
   Notes on the introduction of the MOS Technology MCS 6500 series microprocessors.
5. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "MCS 6500 Microcomputer Family Cross Assembler Manual — Preliminary" (August 1975)
6. Rehnke, Eric C., Editor, "KIM-1/6502 Users Notes", P.O. Box 33077, North Royalton, OH 44133
   Published about 6 times per year. The single most useful source of programs and miscellaneous information on the KIM-1.
7. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "Users Manual — Memory Expansion Modules KIM-2 and KIM-3" (September 1976)
8. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 Flyer ca. March 1976 — "MCS 6530 Memory, I/O, Timer, Array"
   A description of the 6530 ROM used on KIM-1.
9. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "TIM Manual" (March 1976)
10. MOS Technology, Inc., 950 Rittenhouse Rd., Norristown, PA 19401 "KIM-1 Application Note No. 2"
    Describes Interval Timer Operation. Helps to clarify the use of the timer. See also examples in the KIM monitor 6530-002 and -003.
11. Ohio Scientific Instruments, 11679 Hayden Ave., Hiram, OH 44234 "Model 300 Computer — Trainer Lab Manual"
    A series of 20 programs for instruction on the 6502 microprocessor based Model 300 Trainer. Programs are easily adapted to KIM-1 operation.
12. Ohio Scientific Instruments, 11679 Hayden St., Hiram, OH 44234 "OSI Application Note No. 1"
    Covers 6530 TIM Monitor.
13. Ohio Scientific Instruments, 11679 Hayden St., Hiram OH 44234 "Application Note No. 2"
    OSI 480 Backplane and Expansion System.
14. Ohio Scientific Instruments, 11679 Hayden St., Hiram, OH 44234 "OSI Application Note No. 5"
    Interfacing OSI Boards to other systems including KIM-1.
15. Ohio Scientific Instruments, 11679 Hayden St., Hiram, OH 44234 "OSI Model 430 Super I/O Board Instruction Manual"
16. Ohio Scientific Instruments, 11679 Hayden St., Hiram, OH 44234 "Model 420C, 4K Memory Expansion Board"
    Instruction Manual — use together with OSI Application Note No. 2 on the 480 Backplane and Application Note No. 5 on interfacing OSI boards to other systems including KIM-1.
17. ON-LINE, 24695 Santa Cruz Hwy., Los Gatos, CA 95030
    This classified ad newsletter often announces KIM-1 and 6502 software and hardware accessories. 18 issued $3.75.
18. Helmers, Carl, "There's More to Blinking Lights Than Meets the Eye" Byte 1, No. 5, pp.52-54 (January 1976)
    A program for creating patterns of flashing lights (LEDs).
19. Lloyd, Robert G., "There's More to Blinking Lights, etc." KIM-1/6502 Users Notes
    A KIM-1 version of Carl Helmers earlier program in Byte.
20. Ziegler, John, "Breakpoint Routine for 6502" Dr. Dobbs Journal 1, No. 3, pp. 17-19 (1976)
    Requires a terminal and a TIM Monitor. Upon entering, the program counter is printed, followed by the active flags, accumulator, register, Y register and stack pointer.

21. Anon., "What's New Kim-o-sabee?" Byte 1, No. 8, p. 14 (April 1976)
Brief notes on KIM-1.

22. Simpson, Richard S., "A Date with KIM" Byte 1, No. 9, pp. 8-12 (May 1976)
A description of the features of KIM-1.

23. Microcomputer Associates, 111 Main St., Los Altos, CA 94022 "Jolt Microcomputer" Radio-Electronics 47, No. 6, p.66 (June 1976)
Includes description of JOLT, based on 6502, and gives demonstration program using DEMON Monitor.

24. Travis, T.E., "KIM-1 Microcomputer Module" Microtrek, pp. 7-16 (August 1976)
Notes and programs for KIM-1 including Drunk test and several useful routines.

25. Anon., "MOS Technology — KIM MCS 6502" Interface Age 1, No. 9, pp. 12, 14 (August 1976)
An announcement of the KIM-1.

26. Rankin, Roy and Wozniak, Steve, "Floating Point Routines for the 6502" Dr. Dobbs Journal 1, No. 7, pp. 17-19 (August 1976)
Calculation from 10^{-38} to 10^{+38} with 7 significant digits.

27. Bradshaw, Jack, "Monitor for the 6502" Dr. Dobbs Journal 1, No. 7, pp. 20-21 (August 1976)
Monitor a la OSI.

28. Garetz, Mark, "Lunar Lander for the 6502" Dr. Dobbs Journal 1, No. 7, pp. 22-25 (August 1976)
A game requiring TIM Monitor and a terminal.

29. Gupta, Yogesh M., "True Confessions: How I Relate to KIM" Byte 1, No. 12, pp. 44-48 (August 1976)
A series of notes on KIM-1. Includes Clock Stretch and Random Access Memories, Bus Expansion and modification of drive capability using tristate drivers, Interrupt Prioritizing Logic and Halt Instruction.

30. Thompson, Geo. L., "KIM on, Now" Byte 1, No. 13, pp. 93-94 (September 1976)
Notes on using KIM-1.

31. Wozniak, Steve, "Mastermind: A Number Game for the 6502" DDJ 1, No. 8, pp. 26-27 (September 1976)
A number game adaptable to KIM-1 with terminal.

32. Baum, Allen and Wozniak, Stephen, "A 6502 Dissembler" Interface Age 1, No. 10, pp. 14-23 (September 1976)

33. Kjeldsen, Tony, "Next of KIM" (letter) Byte 1, No. 14, p. 136 (October 1976)

34. Pittman, Tom, "Tiny Basic for 6502" DDJ 1, No. 9, pp. 22-23 (October 1976)
Available from Itty Bitty Computers. TB650K (0200-OAFF) is for KIM and most homebrew 6502 systems with RAM in first 4K memory.

35. Anon., "Build a Simple A to D" Interface Age 1, No. 12, pp. 12-14 (November 1976)
Simple circuit, 6502 software, 16 locations. Use to interface a pot or a joystick.

36. Rankin, Roy and Wozniak, Stephen, "Floating Point Routines for 6502" Interface Age 1, No. 12, pp. 103-111 (November 1976) — See also DDJ 1, No. 7, pp. 17-19 (August 1976)
Contains good annotated listings. Loads 1DOO-1FEE.

37. Ohio Scientific Instruments, 11679 Hayden St., Hiram, OH 44234 Flyer: "OSIs New 8K Basic for 6502"
Written by Microsoft. Has automatic string space handling and runs up to 8 times faster than 8080 Basic. Cost $50.

38. Cybersystems, Inc., 4306 Governors Drive West, Huntsville, AL 35805 Flyer: "The Microcyber 1000"
A complete microcomputer system based on a repackaged KIM-1. Provides power supply, two separate ports for I/O, TTY connector, Audio Cassette connector, room for expansion board, etc.

39. Microsoftware Specialists, Inc., 2024 Washington, Commerce, TX 75428 Flyer: Assembler/Text Editor Program (4K)
Compatible with MOS Technology Assembler. Documentation and cassette $19.50.

40. United Microsystems Corp., 2601 S. State St., Ann Arbor, MI 48104 Flyer: "UMC KIM/ALPHA"
A modular system based on memory modules of 4K and 8K, full keyboards, modular backplanes. Cost ca. $700. Video monitor extra.

41. Riverside Electronic Design, Inc., 1700 Niagara St., Buffalo, NY 14207 Flyer: "KEM, KIM-1 Expansion Module and MVM-1024, Microprocessor Video Module"
Used for expansion of KIM-1 system with boards having S-1-- edge connectors ASCII keyboard interface. MVM-1024 is a video display board, scrolling, edit functions, 16 rows of characters, blinking cursor, etc.

42. 6502 Program Exchange, 2920 Moana, Reno, NV 89509 Flyer of April 1977 lists 7 programs for KIM-1 with 1 to 4K of memory and keyboard facilities.

43. DATA1-K Assembler Flyer November 10, 1976
A resident assembler for KIM-1. Model DATA1-K is compatible with MOS Technology Cross Assembler Language.

44. Pollock, James W., "1000 WPM Morse Code Typer" 73 Mag. No. 196, pp. 100-103 (January 1977)
Use of KIM-1 for sending code at 9-1000 WPM from a keyboard.

45. Robbins, Carl M., "The Microprocessor and Repeater Control" QST 61, No. 1, pp. 30-34 (January 1977)
KIM-1 control of repeater functions.

46. Inman, Don, "Data Handler Users Manual" Peoples Computer Co. 5, No. 4, pp. 10, 11, 13 (January-February 1977) Peoples Computer Co. 5, No. 5, pp. 16, 17, 18 (March-April 1977) Peoples Computer Co. 5, No. 6, pp. 52, 53, 55 (May-June 1977)
A how-to course in 6502 programming.

47. Miller, Lindsay, "Found, A Use for Your Computer" Kilobaud, Issue No. 2, p. 80 (February 1977)
A clock program for the KIM-1.

48. Gordon, H.T., "Stringout Mods" DDJ 2, No. 2, p. 8 (February 1977)
A 6502 program applicable to KIM-1 to relocate blocks of instructions in RAMs.

49. Fugitt, Lemuel A., "A 6502 Op Code Table" Byte 2, No. 3, p. 36 (March 1977)
See also Allen, Syd, "6502 Op Code Table" in KIM-1/6502 Users Notes, Issue 4, p. 9 (March 1977).

50. Kushnier, R., "A Partial KIM-1 Bibliography" KIM-1/6502 Users Notes No. 4, p. 7 (March 1977)

51. Cushman, Robert H., "Bare-bones Development Systems Make Good Learning Tools" EDN 22, No. 6 (March 20, 1977) See also 22, No. 8, pp. 104-111 (April 20, 1977) 22, No. 4, pp. 89-92 (February 20, 1977) 22, No. 10, pp. 84-90 (May 20, 1977) 22, No. 12, pp. 79-84 (June 20, 1977)
Use of KIM-1 in a music program is detailed in April 1977 issue.

52. Salter, Richard J. and Burham, Ralph W., "Navigation with Mini-0" Byte 2, No. 4, pp. 100-109 (April 1977); See also Byte 2, No. 2, p. 62 (February 1977) and Byte 2, No. 3, p. 70 (March 1977)
Several articles in a series on the Omega Navigation System and the Mini-0 Receiver driven by a KIM-1 processor. Developed at the Ohio University Avionics Engineering Center.

53. Haas, Bob, "KIM-1 Memory Expansion" Kilobaud, No. 4, pp. 74-76 (April 1977)
Adding the S.D. Sales 4K Low Power RAM board to KIM-1.

54. Sherman, Ralph, "A 650X Program Relocater" DDJ 2, No. 4, pp. 30-31 (April 1977)

55. Ockers, Stan, "TV Sketch Program" DDJ 2, No. 4, pp. 32-33 (April 1977)
A program for use with KIM-1 equipped with a Southwest Tech Prod Co. Graphics Board GT 6144.

56. Jennings, Peter R., "Microchess" DDJ 2, No. 4, p. 33 (April 1977)
Description of chess playing program. Cost $10.

57. Wear, Tom, 380 Belaire, Punta Gorda, FL 33950, Private Communication April 20, 1977
Information on bringing up new memory boards with KIM-1, including OSI 420 and OSI 480 backplane. Includes a very nifty memory test routine for checking the operation of memory boards. See also KIM-1/6502 Users Notes No. 5, p. 4 (May 1977).

58. Loffbourrow, Tod Interface Age (April 1977)
All about a robot named Mike -- based on KIM-1.

59. Tripp, Robert M., "PLEASE" Flyer: The Computerist, May 1977
Fun and Games with KIM-1. A cassette in Supertape for fast loading into KIM-1. Has a number of interesting programs including clock, timer, billboard, travelling display, drunk test, Hi-Lo number game, etc. Available for $10 from Robert M. Tripp, P.O. Box 3, S. Chelmsford, MA 01824.

60. Jennings, Peter, "Microchess" The Computerist Flyer, May 1977
Play Chess with KIM-1 with no additional peripherals or memory. Availble from Robert M. Tripp for $15.

61. The Computerist, P.O. Box 3, S. Chelmsford, MA 01824 Flyer May 2, 1977
Offers power supplies for KIM-1, also a relay kit to permit KIM to control two reed relays for two cassette recorders.

62. Computer Shop, 288 Norfolk St., Cambridge, MA 02139 Adv. in The Computerist, p. 18, May 1977
Offers a 4K RAM board for $74.50 that can be used with KIM-1.

63. Riverside Electronics, 1700 Niagara St., Buffalo, NY 14207 Advertisement. See KIM-1/6502 Users Notes No. 5 (May 1977)
Offers 5 Application Notes for $1 on the use of their MVM-1024 and KEM expansion boards. Ask for MVM-1, 2, 3, 4, 5 and KIM-1.

64. Aresco, 314 Second Ave., Hahhon Hts., NJ 08035 See KIM-1/6502 Users Notes, Issue No. 5, p. 1 (May 1977)
Lists several programs available for KIM, TIM, etc.

65. Simpson, Rick, "Come Fly with KIM" Byte 2, No. 6, pp. 76-80 (June 1977)
Load 12K of memory in two minutes with a "Fly Reader" for paper tape.

66. Lancaster, Don, "A TVT for your KIM" Kilobaud, No. 6, pp. 50-63 (June 1977)
TVT-6L is a low cost method of providing a TV monitor for KIM-1. Uses minimum new hardware but depends on a software program in KIM-1 memory for handling characters. Uses a low cost TV (Pansonic T-126A) for monitor.

67. Lancaster, Don, "Build the TVT-6" Popular Electronics 12, No. 1, pp. 47-52
A low cost direct video display based on KIM-1 software and a minimum of added hardware. Slightly different than the TVT-6L.

68. Pickles and Trout, P.O. Box 2270, Goleta, CA 93018 "TV Mod Kit"
Detailed instructions and kit of parts for conversion of a low cost ($80 approx.) Hitachi SX Chassis (Model P-04, P-08, PA-8, etc.) for a TV Monitor.

69. Forethought Products, P.O. Box 386, Coburg, or "KIMSI" The Computerist, p. 8 (June 1977)
KIMSI is a motherboard/Interface that allows KIM-1 to be interfaced to an S-100 bus; 8 slots.

70. MOS Technology/Commodore "PET" The Computerist, p. 17 (June 1977)
An announcement on PET, a new 6502 system with video monitor, ASCII keyboard, 12 K ROM including 8 K Basic and 4 K RAM, audio cassette; price ca. $4.95, available September 1977.

71. Grater, Robert, "Giving KIM Some Fancy Jewels" Byte 2, No.7, pp. 126-127 (July 1977)
Adding a remote LED display for the KIM-1.

72. Runyan, Grant, "The Great TV to CRT Monitor Conversion" Kilobaud, No. 7, pp. 30-31 (July 1977)
Although not specific to KIM-1, this article is useful in adapting a monitor to KIM. Uses inexpensive 12" Hitachi Model P-04, P-08, PA-4, PA-8. See also Sams Photofact Folder 1 Set 1601 or Folder 3 Set 1501.

73. Simpson, Richard, "KIM Forum" Kilobaud, No. 7, pp. 4, 19, 86 (July 1977)
KIM-5 will be a ROM expansion board with up to 8 MCS 6504 (2K x 8) mask programmed ROMs. One ROM is KIMath, a set of subroutines for doing floating point arithmetic. Cost $50. Programming Manual for KIMath purchased separately is $15. Also a resident assembler and text editor are available as a set of 3 ROMs.

74. Tripp, Robert M., "The 6502 World" The Computerist, p. 16 (July 1977)
MOS Technology may offer a 16K RAM board for KIM-1. New KIM repair facility is KIM Diagnostig Center 2967 W. Fairmont Ave. Phoenix, AZ 85017 Tele. 602-248-0769
4K Ram for KIM-1 assembled and tested for only $129 available from Tripp.

75. Tripp, Robert M., "HELP Relay Package" The Computerist Flyer July 5, 1977
Components for relay control of 2 cassette recorders. Includes control program subroutine.

76. Tripp, Robert M., "4K RAM Board" The Computerist Flyer July 5, 1977
4K for KIM-1, socketed chips, 5.25" x 9.25" board, can separately address each 1K. Cost $129 assembled.

77. Tripp, Robert M., "Digital to Analog Converter" The Computerist Flyer July 1977
Micro Technology Unlimited DAC board with audio output to drive 8/16 ohm speakers. Can play 4 part harmony with only KIM-1. Includes cassette tape program for tunes.

78. Tripp, Robert M., "Mod to Improve the PLEASE Clock" The Computerist July 5, 1977

79. Boyle, Peter, "The Gory Details of Cassette Storage" Kilobaud, No. 3, pp. 116-119 (March 1977)
Comments on audio cassette problems. States that KIM runs at 133 baud but is capable of 1200 baud.

80. Johnson Computer, P.O. Box 523, Medina, OH 44256 "Basic for KIM-1" μP No. 4 (June 1977)
Resides in 2K at address 2000. Available in paper tape $5.

81. Johnson Computer, P.O. Box 523, Medina, OH 44256 "Harness Eliminator" μP No. 4 (June 1977)
Minimize wiring in connecting KIM 2 or 3 to KIM-1 with a rigid coupling.

82. Johnson Computer, P.O. Box 523, Medina, OH 44256 "KIM-1 Resident Assembler/Text Editor Model DATA1-K" μP No. 4 (June 1977)
Use with MOS Tech Cross Assembler Manual.

83. Johnson Computer, P.O. Box 523, Medina, OH 44256 "KIMath — Floating Point Math Package" μP No. 4 (June 1977)
Rom is $50. Documentation alone $15. Available from Johnson Computer.

84. Tripp, Robert M., "Is the KIM-1 for Every-1" Kilobaud, No. 8, pp. 56-57 (August 1977)
General description of KIM-1.

85. Fish, Larry, "Troubleshoot Your Software" Kilobaud, No. 8, pp. 112-113 (August 1977)
A trace program for 6502.

86. Severson, Gerald D., "Plaudits for MOS Technology" DDJ 1, No. 6, pp. 5 (June-July 1976)
Note on good service from MOS technology on the 6502.

87. Western Data Systems, 3650 Charles St., No. Z, Santa Clara, CA 95050 "Western Data's 6502-Based Data Handler" DDJ 1, No. 6, p. 43 (June-July 1976)
A $170 kit with hex keyboard, LED binary readout, 1 K ram, capability of addressing 65K, uses 100 pin tustate bus and is compatible with a long list of Altair peripherals, 100 pin connector.

88. Espinosa, Chris, "A String Output Subroutine for the 6502" DDJ 1, No. 8, p. 33 (September 1976)
This routine saves pointers, loops, etc. in outputting the string.

89. Meier, Marcel, "6502 String Output, Revisited" DDJ 1, No. 10, p. 50 (November 1976)
Further mod of Espinosa's earlier routine.

90. Anon., "That didn't Take Long at All" Byte 1, No. 5, p. 74 (January 1976)
Note on 6502 product introduction and the JOLT computer kit.

91. Anon., "Control Logic for Microprocessor Enables Single Step" Electronic Design, p. 78 (October 11, 1976)
Uses 6502 system.

92. Anon., "6502 Disassembler" Interface Age, p. 14 (September 1976)

93. Butterfield, Jim, "KIM Goes to the Moon" Byte 2, No. 4, pp. 8-9, 132 (April 1977)
A lunar lander program; see also same program in KIM-1/6502 users notes.

94. Hybrid Technologies, P.O. Box 163, Burnham, PA 17009 "Ad for KIM-1 Peripherals" Byte 2, No. 8, p. 157 (August 1977)
2K/8K ROM based, EProm Programmer, 2K/4K/8K Ram boards, assembler board, TV Interface board, relay board, mother boards.

95. Simpson, Richard, "Circular Ad for 6502 Software" Aresco, 314 Second Avenue, Haddon Heights, NJ 08035, July 26, 1977
Describes FOCAL, a 4K language similar to BASIC, and a 2.5K resident assembler suitable for all 6502-based micro systems.

96. Commodore International, Ltd., 901 California Avenue, Palo Alto, CA 94304 Tele. (415) 326-4000 "The PET!" Peoples Computers 6, No. 1, p. 59 (July-August 1977)
An announcement of the PET computer based on 6502. Available early September 1977 for $595.

97. Crow, Darrell — Microcomputer Associates, 2589 Scott Blvd., Santa Clara, CA 95050 Tele. (408) 247-8940 "6502 Assembler, Tinz Basic on ROM's" Peoples Computers 6, No. 1, p. 60 (July-August 1977)
RAP is a 1.75K Resident Assembler Program on two 2K ROM's together with 2.2K Tinz Basic, pin compatible with 2708-type PRoms — price $200.

98. Inman, Don, "The Data Handler Users Manual Part 4" Peoples Computers 6, No. 1, pp. 42-46 (July-August 1977) (Cont. from Item 46)
Covers indexed addressing.

99. Anon., "User Group Being Formed for Commodore PET 2001 Computer" ON LINE 2, No. 10, p. 11 (August 3, 1977)
Membership is $5 including User Notes. Contact Gene Beals, P.O. Box 371, Montgomeryville, PA 18936.

100. Anon., "6502 Assembler/Text-Editor for KIM-1 and TIM" ON LINE 2, No. 10, p. 10 (August 3, 1977)
Resides in 2K, requires Teletype or CRT and cassette recorder. $29.95. M.S.S., Inc., 1911 Meadow Lane, Arlington, TE 76010

101. Anon., "MICRO-ADE" ON LINE 2, No. 10, p. 6 (August 3, 1977)
New Product Announcement by MICRO-WARE Ltd., 27 Firstbrooke Rd., Toronto, Canada, M4E2L2 Micro-Ade, a 4K package is a compact development tool for all 6502 users including KIM-1. User manual, hex dump, object program on paper tape or KIM cassette is $25. Complete annotated source listing is available for another $25.

102. Ohio Scientific Instruments, 11679 Hayden, Hiram, OH 44234
OSI Small Systems Journal (first regular July 1977) is a new publication, $6 for six issues, devoted to 6502 and OSI users.

103. Deckant, Gary, "Understanding and Using the 6502 Assembler" OSI Small Systems Journal 1, No. 1, p. 8 (July 1977)
Explains use of assembler program.

104. Anon., "1K Corner" OSI Small Systems Journal 1, No. 1, p. 8 (July 1977)
The game of 23NIMB for OSI 65V systems. Requires terminal. Resides 0200-0332.

105. Cheiky, Mike and Meier, Marcel, "The Auto-Load Cassette System" OSI Small System Journal 1, No. 1, pp. 9-14, (July 1977)
For OSI 65V system.

106. Anon., "The 6502 Disassembler — From Object to Source End" OSI Small System Journal 1, No. 1, pp. 14-15 (July 1977)
A disassembler is a program which attempts to convert machine code back into assembler source. Obviously it cannot reconstruct comments or labels. Points out other pitfalls in using disassemblers.

107. Pyramid Data Systems, 6 Terrace Ave., New Egypt, NJ 08533 Ref :KIM-1/650X, Users Notes No. 6, p. 1 (July 1977)
XIM is an extended I/O monitor package for Kim, residing in about 1K memory. Adds 17 commands to terminal equipped Kim. Has 45-page user manual. Cost $12.00 for manual and KIM cassette.

108. ORB, P.O. Box 311, Argonne, Ill., 60439, "The First Book of KIM" Ref: KIM-1/650X, Users Notes No. 6, p.1 (July 1977)
Ockers, Rehnke and Butterfield have collaborated in a 180-page new book to guide beginners and others in the use and enjoyment of KIM-1. Cost $9.50 including postage.

109. Aresco, 314 Second Ave., Haddon Hts., NJ 08035, "Comprehensive 650X Assembler/Text Editor" Ref: KIM-1/650X Users Notes No. 6, p. 4 (July 1977)
Designed for use with KIM-1 but can be used with other 650X systems such as TIM, Apple, Baby, OSI, etc. — Occupies 6K, available on KIM cassette or KIM-TIM paper tape. Cost $60.00.

110. Bates, Dan, Rt 7, Box 310, Claremore, Okla, 74017, "Small Microcomputer Board using 6505. Ref: KIM-1/650X, Users Notes No. 6, p. 9 (July 1977)
Bates has developed a board around the 28 pin 6505 and sells the 6" x 4" PC board for $15.00 including schematic and assembly instructions. Can handle ASCII to Baudot, microcontrolled repeater/autopitch, etc.

111. Laabs, John, "Build a $20 EPROM Programmer" Kilobaud No. 9, pp. 70-77, (Sept 1977)
KIM-1 is used to run software and some external hardware to program the 5204 4K EPROM.

112. Ohio Scientific Instruments, Hiram, Ohio, 44234, "A Computer that Thinks in BASIC" Kilobaud No. 9, p. 10, (Sept 1977)
Announcement of OSI's Model 500 CPU board built on 6502. Complete with 8K Basic in ROM for $298.

113. Clarke, Sheila, "A PET for Every Home" Kilobaud No. 9, pp. 40-42, (Sept 1977)
A look at the Commodore PET 2001 based on the 6502. About $600 includes Video terminal keyboard, 12K, (8K Basic in ROM and 4K operating system).

114. American Institute for Professional Education, Carnegie Bldg., Hillcrest Road, Madison, NJ, 07940, "Microprocessing Fundamentals" Circular Advertisement — approx. Aug. 15, 1977.
Dr. Joseph B. Ross, Purdue Univ. and Dr. Garnett Hill, Oklahoma State Univ. will present a course in Fall of 1977 at several locations. Course is based on KIM-1 hardware together with instruction in Digital Devices, Programing Fundamentals, Advanced Programing, Peripherals, 1/0 addressing, applications, etc. Cost about $600 including a KIM-1 to keep after the course.

115. Gregson, Wilfred J. II,"RTTY with the KIM" 73 Magazine 9, No. 204, p. 110-112 (Sept 1977)
A clever program for using KIM-1 and the 6-digit LED display as a readout for a RTTY signal. Simply feed the audio from a receiver into the tape input of KIM-1 and read the message as it flows across the display (about 45.5 baud, 60 wpm). Can also handle other ratio to 100 baud). Can also use KIM-1 as a display only, operating from an already available terminal unit.

116. Synertek 3050 Coronado Drive, Santa Clara, CA 95051 Misc. Data Sheets received by mail.
Describes second source of 6502 and associated microprocessor chips by Synertek. SY6502 is updated to include ROR function. Other chips include SY6530, SY6522 (PIA), SY6532, SY6520, etc.

117. Rockwell International, 3310 Miraloma Ave., P.O. Box 3669, Anaheim, CA 92803 Data Sheets D39 thru D44 received by mail.
Describes Rockville R6502 microprocessor and other second source Microprocessor accessory chips including R650X, R651X, R6520 (PIA) R6530 (ROM, RAM, 1/0, Timer) R6532, etc. R6502 also available in 2 MHz option. R6502 has the updated ROR function.

118. Bumgarner, John O., "A-KIM-1 Sidereal/Solar Clock" Interface Age 2, No. 9, p. 36-37 (Aug 1977)

119. Atkins, R. Travis, "A New Dress for KIM" Byte 2, No. 9, p. 26-27 (Sept 1977)
Describes mounting the KIM-1 in a briefcase together with power supply, prototype boards, etc.

120. Chamberlin, Hal, "A Sampling of Techniques for Computer Performance of Music" Byte 2, No. 9, p. 62-83 (Sept 1977)
General Discussion of Music Generation plus detailed information on application to KIM-1 and a description of the hardware and software for a D/A music board and software package being marketed by Micro Technology Unlimited, 29 Mead St., Manchester, NJ 03104. PC board alone is $6.00, assembled and tested D/A board $35.00, software package on KIM cassette is $13.00 additional.

121. Beals, Gene, P.O. Box 371, Montgomeryville, PA 18936, "User Group for the Commodore PET 2001 Computer" Ref: On Line 2, No. 11, p. 2 (Aug 24, 1977)
Yearly membership $5.00 brings Users Notes publication.

122. Cater, J., 11620 Whisper Trail, San Antonio TX 78230, "Run OSI 6502 8K Basic on your TIM or JOLT" On Line 2, No. 11, p. 3 (Aug 24, 1977)
Cost $4.00 for annotated source and object code of patches for TIM or JOLT."

123. Firth, Mike, 104 N. St. Mary, Dallas, Texas 75214, "Large Type Summary of Command Coder for 6502 plus addresses." On Line 2, No. 11, p. 8 (Aug 24, 1977) Cost: $0.13 stamp plus SASE.

124. House, Gil, P.O. Box 158, Clarksburg, Md., 20734, "6502 Legible Tape Labeler." On Line 2, No. 11, p. 9 (Aug 24, 1977)
A program for TIM (JOLT DEMON), Hex tape and documentation $4.00

125. Kushe, Willi, "KIM-1 Breakpoint Routines Plain and Fancy" DDJ 2, No. 6, pp. 25-26 (June-July 1977)
A modified routine using KIM-1 Monitor allows size to be cut to only 124 Bytes.

126. F and D Associates, Box 183, New Plymouth, OH 45654 On Line 2, No. 9, p. 9 (July 13, 1977)
New product Announcement: Video Display Board compatible with 6502. Two pages 16 lines x 64 characters, scrolling, screen erase. Bare Board $29 incl. software and documentation.

127. Staff Article "PET Computer" Peoples Computers 6, No. 2, p. 22-27 (Sept-Oct. 1977)
Chuck Peddle, father of the PET is interviewed. Interesting comments on the marketing of this 6502 based microcomputer and accessories.

128. Inman, Don, "The Data Handler Users Manual: Part 5" Peoples Computers 6, No. 2, pp. 50-55 (Sept-Oct. 1977)
Covers Session VII — Writing Programs

6502 BIBLIOGRAPHY

PART II

William Dial
438 Roslyn Avenue
Akron, OH 44320

129. Torzewski, Joe, "Apple I Library" On_Line 2 No. 12 p. 11 (Sept 14, 1977)
Apple I owners interested in a library for software and hardware should contact Joe Torzewski, 51625 Chestnut Rd., Granger IN 46530.

130. House, Gil, P.O. Box 158, Clarksburg, MO 20734, "6502 Tape Labeler" On_Line 2 No. 12 p. 11 (Sept 14, 1977)
Man readable 6502 legible tape labeler for TIM, JOLT, DEMON

131. Cater, J., 11620 Whisper Trail, San Antonio, TX 78230, "Run OSI 6502 8K BASIC on your Tim or Jolt" On_Line 2 No. 11, p. 13 (Sept 14, 1977)
Full info and patches to run this super fast BASIC.

132. Staff Article "The PET Computer" Personal Computing 1 No. 5, pp 30-40 (Sept-Oct, 1977)
Interviews with Chuck Peddle of Commodore and with other microcomputer experts.

133. Lancaster, Don, "Hex-to-ASCII Converter for your TVT-6" Popular Electronics 12 No. 4, pp 49-52 (Oct. 1977)
Simple module produces op-code display for entire computer. Describes a board to be connected between the TVT-6 and the KIM-1 microcomputer.

134. Microcomputer Associates Inc., 2368-C Walsh Ave., Santa Clara, CA 95050 Popular Electronics 12 No. 4, p. 100 (Oct, 1977)
New Product Announcement: A 6502 RAP, resident assembler program and TINY BASIC of ROM. Cost $200.

135. CGRS Microtech, P.O. Box 368, Southampton PA 18966, On_Line 2 No. 13, p 2 (Oct 5, 1977)
New Product Announcement: EXOS and DATE are two new 6502 software packages. EXOS is an Extended Operating System featuring a number of useful commands and DATE is a disassembler, assembler, trace and debug editor. Available on four programmed 2708 EPROMS or on TIM format paper tape. Programs are each $150 or $295 for both, on EPROMS.

136. Pyramid Data Systems, 6 Terrace Ave., New Egypt, NJ 08533, On_Line 2 No 13, p 6 (Oct 5, 1977)
New Product Announcement: XIM is a 1K software package for KIM that adds 17 commands to the KIM Monitor, including a Breakpoint routine. Cassette and 45 page manual is $12 ppd., paper tape is $10.

137. K L Power Supplies, P.O. Box 86, Montgomeryville, PA 18936, On_Line 2, No. 13, p. 11 (Oct 5, 1977)
New Product Announcement: Model 512 Power Supply is for the KIM with enough capacity for an extra 8K and other accessories.

138. Matthews, K., "6502 Forum" Kilobaud No. 10, p 11, (Oct. 1977)
Mentions E.C.D. Micromind II based on the 6512 A (related to 6502).

139. Rugg, Tom and Feldman, Phil, "BASIC Timing Comparisons" Kilobaud No. 10 p. 20 (Oct, 1977)
Compares over 30 different hobby computer systems on seven different Benchmark programs in BASIC. Fastest was OSI 8K BASIC using 6502 in a Challenger running at 2 MHz. Actually a late entry which was still a little faster was the HeathKit H-11 with a special Extended Instruction Set and a Floating Instruction Set which are to be offered as accessories for the H-11.

140. Overstreet, Jim, "Try Your KIM-1 on RTTY" 73 Magazine No. 205 pp 88-91 (Oct, 1977)
Has a Baudot Receive Program that takes the output from an FSK converter and runs a video terminal with the KIM board. A CW transmit program is also given in the article.

141. Schawlow, Arthur L., "Search Subroutine for the 6502 Disassembler", Interface Age 2 No. 1, p 146 (Oct, 1977)
A description, listing and sample run of an object code search subroutine for use with the 6502 Disassembler published in the September 1976 issue of Interface Age.

142. Simonton, John S., Jr., "What the Computer does ... an Introduction.", Polyphony 3, No. 1, pp 5-7, 28 (July, 1977)
PAIA Electronics will shortly have a complete KIM-1 package showing how to interface with their 8700 Computer Controller based on a 6503 processor. A large selection of programs for KIM is promised.

143. Simpson, Rick, "KIM Forum", Kilobaud No. 11, pp 16-17, 48 (Nov, 1977)
Caxton Foster of the Computer Sciences Dept. of the University of Massachusetts is the author of a college text on microprocessors and all programming examples use KIM-1. Also R.W. Burhans, E.E. Dept. of Ohio University has some informative comments on the adjustment of the PLL pot VR-1.

144. Butterfield, Jim, "Hyper about Slow Load Times", Kilobaud No 11, pp 66-69 (Nov, 1977)
Butterfield explains the development of his HYPERTAPE (Supertape) program for loading or dumping to a KIM audio cassette at 50 bytes per second, six times the normal KIM-1 rate.

145. Blankenship, John, "Expand Your KIM", Kilobaud No 11, pp 84-87 (Nov 1977)
The first of several articles on expanding KIM to use the S-100 bus to give 13K memory, Cromenco Dazzler, a printer and keyboard, joysticks,etc.

146. Johnson Computer, P.O. Box 523, Medina, OH 44256, On Line 2, No 14, p 7 (October 26, 1977)
New Product Announcement: KIM-1 8K Basic by Microsoft is available in either a 6-digit or 9-digit precision version which includes full printout of error messages. Prices are $97.50 and $129.00.

147. Rockwell International, P.O. Box 3669, Anaheim, CA 92803, Product Bulletin
Rockwell now has available a number of 6500 family microprocessor chips including r6502, r6505 and others. They also are promoting SYSTEM 65, a floppy disc based powerful development system.

148. Sneed, James R., "Adding an Interrupt Driven Real Time Clock", Byte 2 No. 11, pp 72-74 (Nov, 1977)
An external board drives interrupts at 15 Hz which is used to calculate time for use by the computer.

149. Brader, David, "A 6502 Personal System Design: KOMPUUTAR", Byte 2 No. 11 pp 94-137 (Nov, 1977)
A very detailed constructional article.

150. NCE/CompuMart, 1250 N. Main St., Ann Arbor, MI 48104, Byte 2 No. 11, p 140, (Nov, 1977)
New Product Announcement: A number of Accessories for the KIM-1 including backplane/S-100 adapter, 8K Seals memory, Poly Video terminal interface, Itty Bitty Tiny BASIC, Matrox Video RAMS, Graphics and Alpha-Numerics Boards.

151. The Enclosures Group, 55 Stevenson St., San Francisco, CA 94105, Byte 2 No. 11, p 234 (Nov, 1977)
New Product Announcement: Offers an enclosure to dress up the KIM-1.

152. Apple Computer Inc., 20863 Stevens Creek Blvd., Cupertino, CA 95014, Byte 2, No. 11, p 252 (Nov, 1977)
Apple II is a new entry in the home computer market. At $1298 it offers 6K Basic in ROM video graphics in 15 colors, 4K of programmable memory in RAM, a 2K monitor, cassette interface, floating point package, etc.

153 Anon., "Get the most out of Basic", OSI Small Systems Journal 1, No. 2, pp 4-7 (Sept, 1977)
Note on Basic in general and the OS-65D System.

154. Smith, Gary A., "Contributed Program", OSI Small Systems Journal 1, No. 2, p. 12 (Sept, 1977)
Program displays the memory address and the data contained in HEX.

155. Anon., "OSI 6502 Cycle Time Test", OSI Small Systems Journal 1, No. 2, pp 12-13 (Sept., 1977)
Measures the cycle time using a stop watch and program to record the number of whole cycles.

156. Anon., "Memory Test", OSI Small Systems Journal 1, No. 2,pp15-17(Sept 1977)
A memory test for video and serial-based computers using the 6502.

157. Anon., "1K Corner: Close the Window", OSI Small Systems Journal 1, No. 2, p 18 (Sept., 1977)
Close the Window is a dice game designed to be played on the OSI 65V Computers.

158. The COMPUTERIST, P.O. Box 3, South Chelmsford, MA 01824
MICRO is a new bimonthly publication specializing in information related to 6502 processor based systems.

159. Salzsieder, Byron, "Cheap Memory for the KIM-1", MICRO No. 1 pp 3-4, Oct.-Nov., 1977)
You can add a Veras Systems 4K Byte memory board to your KIM-1 at half the price of the KIM-2.

160. Holt, Oliver, "Terminal Interface Monitor (TIM) for the 6502", MICRO No. 1, pp 1-7 (Oct.-Nov., 1977)
TIM is available on a MOS Technology ROM 6530.

161. Anon., "We're No. 1", MICRO, No. 1, p 6 (Oct-Nov, 1977)
An editorial points out that over 12,000 KIM-1 units are in the field and a thousand more each month are being ordered. Apple I and Apple II systems, plus the OSI units, Jolts, Data Handlers, and other 6502 based systems, plus the huge number of PETs and Microminds that have been ordered, plus a lot of home-brew systems, it all adds up to a lot of 6502 systems. Also Atari has purchased one and one-half million 650X chips for their game units.

162. Ferruzzi, Arthur, "Inside the Apple II", MICRO, No. 1, pp 9-10 (Oct-Nov 1977)
A detailed description of the Apple II.

163. Ferruzzi, Arthur, "Rockwell International and the 6502", MICRO, No. 1, p 10, (Oct.-Nov., 1977)
Rockwell is now second sourcing the entire 6502 product line. They have also developed SYSTEM 65, a fancy development system with dual mini floppies, 16K static RAM, text editor, assembler and debug monitor on ROM, serial and parallel interfaces for terminal and printer, hardware breakpoint, etc.

164. Floto, Charles, "The PET's IEEE-488 Bus: Blessing or Curse?", MICRO, No. 1, p 11 (Oct.-Nov, 1977)
Discussion of this feature mentions a rumor that Pickles and Trout may offer a 488 adapter for their new S-100 I/O board, as well as an I/O board for the 488 bus.

165. Anon., "6502 Related Companies", MICRO, No. 1, p12 (Oct.-Nov., 1977)
Lists 28 companies serving 6502 processors.

166. Tripp, Robert M., "Hypertape and Ultratape", MICRO, No. 1, pp13-16, (Oct.-Nov., 1977)
Ultratape runs at 12 times the normal KIM-1 speed, but requires special programs for both loading and dumping.

167. Rowe, Mike, "KIM-Based Degree Day Dispatcher", MICRO, No. 1, pp 17-18, (Oct.-Nov., 1977)
Hundley Controls of Hanover, Mass. is building a number of different KIM-based systems to be used by fuel oil dealers to perform a variety of functions such as meter ticket reading, basic accounting, calculating degree-days by measuring temperature and determining when oil deliveries are to be made, etc.

168. Tripp, Robert M., "Computer Controlled Relays", MICRO, No. 1 p 19 (Oct.-Nov., 1977)
Relays can be used for control of audio assettes, and a variety of other functions. A 7404 Hex Inverter is used to buffer the signals from the KIM's 6530 Port B I/O lines.

169. Dial, William R., "6502 Bibliography", MICRO, No. 1, pp 21-27, (Oct. - Nov., 1977)
128 references to 6502 related articles, programs, etc.

170. Camus, Armand L., "Making Music with the KIM-1", MICRO, No. 2, pp 3-7, (Dec. 1977-Jan. 1978)
How to write music for a DAC such as that recently described by Chamberlain in Byte Magazine, Sept. 1977.

171. Floto, Charles, "Meet the PET", MICRO, No. 2, pp 9-10 (Dec 1977-Jan 1978)
An owners view of the PET 2001.

172. Dejong, Marvin L., "Digital-Analog and Analog-Digital Conversion Using the KIM-1", MICRO, No 2, pp 11-15, (Dec. 1977-Jan 1978)
Experiments with a KIM-1 controlled DAC/ADC.

173 Wallace, Bob, "The PET Vs. the TRS-80", MICRO No. 2, pp 17-18, (Dec. 1977 - Jan 1978)
A feature-by-feature comparison.

174. Schwartz, Marc, "Ludwig von Apple II", MICRO, No. 2, p 19 (Dec 77-Jan 78).
How to write music for the Apple II.

175. Anon., "MICROBES - Tiny Bugs in Previous MICRO", MICRO, No. 2, p 22, (Dec. 1977 - Jan. 1978).
Some corrections for HYPERTAPE and ULTRATAPE and Computer Controlled Relays.

176. Henkel, Joel, "The Challenge of the OSI Challenger", MICRO, No. 2, pp 23-24 (Dec 1977 - Jan 1978)
An owners impressions of the OSI Challenger.

177. MOS Technology, "Improving Keyboard Reliability", MICRO, No. 2, p 25, (Dec 1977 - Jan 1978)
A hardware modification for your KIM-1 to improve action of the "9, D, or C" keys. Based on an Application Note by MOS Technology.

178. Dial, William, "Important Addresses of KIM-1 and Monitor", MICRO, No. 2, pp 27-30, (Dec 1977 - Jan 1978)
A programmers reference card for the KIM-1.

179. Computer Shop, 288 Norfolk St., Cambridge, MA 02139, MICRO, No. 2, p 26, (Dec. 1977 - Jan. 1978)
Advertisement for CS 100 Video Terminal Board for KIM. Includes portable cabinet for the KIM with space for cassette recorder, ASCII keyboard, power supply, extra memory boards, 3-slot motherboard, TIM kit, etc.

6502 BIBLIOGRAPHY
PART III

William Dial
438 Roslyn Avenue
Akron, OH 44320

180. Gordon, H.T., "Decoding 650X Opcodes", Dr. Dobbs Journal 2, No. 7, pp 20-22 (Aug. 1977)
Subroutines that can be used with KIM.

181. Butterfield, Jim F., "A High-Speed Memory Test Program for the 6502" DDJ 2, No. 7, p 23 (Aug. 1977)
A memory test program written for the KIM system.

182. Anon. "Ohio Scientific's New Disc Operating System", DDJ 2, No. 7, p 32 (Aug. 1977)
The OS-65D is a complete operating system for all disc based OSI computer systems. Includes DOS, 8K Basic, Assembler, Editor, Extended Debugger and a Disassembler.

183. Anon.., "OSI offers Computer that thinks in Basic for $298", DDJ 2, No. 7, p 39 (Aug. 1977)
OSI's new Model 500 CPU board can be used as a stand-alone computer or as the PCU in a larger system.

184. Moser, Carl W., 3239 Linda Dr., Winston-Salem, NC 27106, DDJ 2, No. 8, p 28 (Sept. 1977)
Announcement of New Product: $25 for 6502 Editor and Assembler Hex Listing and Manual. Configured for TIM Systems.

185. Anon.,"1K Corner", OSI Small Systems Journal 1, No. 4, p 3 (Oct. 1977)
Hex address and offset calculator program resides at ODDE to OEE4.

186. Anon., "Now You Can Play Star Wars", OSI Small Systems Journal 1, No. 4, pp 11-13, (Oct. 1977)
Star Wars program by Robert L. Coppedge requires 8K Basic, OSI 440 Video Board and at least 4K of RAM.

187. Anon., "Conventional Typewriter", OSI Small Systems Journal 1, No. 4 pp 8-9 (Oct. 1977)
Gary Smith's program for using the OSI-65V when interfaced to a printer to be used as a conventional typewriter and also modify the text for a data file.

188. Gordon, H.T., "OPLEGL Correction and a 6502 Scanning-Debugger", DDJ 2, No. 9, pp 42-44 (Oct. 1977)
Gordon offers a corrected version of his 650X subroutine, OPLEGL, and gives a new byte-count subroutine, NUMBYT. A new scanning-debugger, SIMBUG, is submitted.

189. Swope, J., "6502 Goodies", DDJ 2, No. 9, Issue 19, p 45 (Oct. 1977)
Swope, President of CGRS Microtech, PO Box 368, Southampton, PA 18966, announces that his company has finished a 6502 computer board for the S100 bus.

190. Wozniak, Stephen, "Sweet 16: The 6502 Dream Machine", Byte 2, No. 11, pp 150-159 (Nov. 1977)
Sweet 16 is a 16 bit "metaprocessor" in software, intended as a 6502 enhancement package, not a stand-alone processor.

191. Shattuck, Bob and Schmidt, Bill, "Receive CW with a KIM-1", 73 Magazine, No. 206, pp 100-104 (Nov. 1977)
A program for receiving CW with optional TTY or KIM display.

192. Johnson, Donald J., "KIM-1 Sidereal/Solar Clock Correction", Interface Age 2, No 12, p 9 (Nov. 1977)

A correction in the listing given in the August issue of Interface Age permits 24-hour operation.

193. KL Power Supplies, PO Box 86, Montgomeryville, PA 18936, Interface Age 2, N No. 12, p 140 (Nov. 1977)
The Model 512, 4.5 amp. power supply is designed for KIM-1.

194. Micro Technology Unlimited, Box 4596, Manchester, NH 03108, Interface Age 2, No. 12, p 140 (Nov. 1977)
The MTU Model K-1000 power supply is designed to power the KIM-1.

195. Wasson, Philip A., "Trace", KIM-1/6502 User Notes, Issue 7/8, pp 2-3 (Sept & Nov 1977)
With this program and about $2.00 worth of hardware you can see displayed on an oscilloscope screen, all of the registers in the 6502 and three consecutive memory locations.

196. Ohsiek, Charles C., "ID on Audio Cassette for SUPERTAPE", User Notes, Issue 7/8, p 4 (Sept & Nov 1977)
Program allows writing an ID on the audio cassette tape prefixing the data SUPERTAPE writes out.

197. Hawkins, George W., "2-Task Alternating Scheduler Routine", User Notes, Issue 7/8, p 5 (Sept & Nov 1977)
Program allows two programs to be run together in the KIM-1.

198. Gordon, Hal, "A Catalog of KIM-1 ROM Bytes", User Notes, Issue 7/8, p 5, (Sept. & Nov. 1977)
A table of the location of ROM bytes.

199. Anway, Allen, "Program BRANCH", User Notes, Issue 7/8, p 6 (Sept & Nov 1977)
With this program you can go through your program, find the Branch instructions and force the branch to see where you will end up.

200. Pollock, Jim, "KIM-1 to S-100 Bus Adapter", User Notes, Issue 7/8, p 7 (Sept. & Nov. 1977)
This adapter allows KIM-1 to be used with S-100 boards such as the $125 8K RAM board of Ithaca Audio.

201. Heinz, Harvey, "A Simple Music Program for KIM", User Notes, Issue 7/8, pp 8-9 (Sept. & Nov. 1977)
This is an excellent tutorial program with basic level explanations.

202. Hapgood, Will, "An A/D Converter", User Notes, Issue 7/8, pp 10-11, (Sept. & Nov. 1977)
A circuit for making very accurate A/D conversions using a Motorola dual-slope conversion chip, MC 1405 or 1505.

203. Butterfield, Jim, "KIM Blackjack", User Notes, Issue 7/8, pp 11-13, (Sept. & Nov. 1977)
Game uses the KIM display to good advantage in this program.

204. Strandtoft, B., "KIM-1 Resident Programs and Subroutines", User Notes, Issue 7/8, p 14 (Sept. & Nov. 1977)
A list of KIM Monitor routines with brief explanations.

205. Goenner, Markus P., "TTY Rapid Load", User Notes, Issue 7/8, p 15, (Sept. & Nov. 1977)
Program starts at 0000 and is fully relocatable.

206. Parson, Charles H., "Read Temperature Once per Minute", User Notes, Issue 7/8, pp 16-18, (Sept. & Nov. 1977)
Program for temperature control systems.

207. Oliver, John and Hall, Williamson, "A KIM-1 Binary Dump and Load Routine", User Notes, Issue 7/8, pp 19-20, (Sept. & Nov. 1977)
SUPERDUMP/SUPERLOAD allows the use of the KIM-1 Cassette tape interface to read and write data blocks under program control. 1K bytes are dumped or loaded in less than 12 seconds.

208. The COMPUTERIST, PO Box 3, S Chelmsford, MA 01824, "MEMORY PLUS for KIM-1", New Product Announcement, MICRO, No. 2, p 2 (Dec 1977-Jan 1978)
New board for fitting directly beneath the KIM-1 has 8K RAM, 8K EPROM MOS Technology Versatile Interface Adapter, EPROM programmer, On Board Voltage Regulators; fully assembled and tested $245; Intel 2716 2K EPROMS extra $50 each.

209. Cole, Phyllis, "PET Update", Peoples Computers 6, No. 3, pp 6-7 (Nov-Dec1977)
Several rumors on the PET are answered.

210. Cole, Phyllis, "Our PET's First Steps", Peoples Computers 6, No. 3, pp 8-10, (Nov-Dec 1977)
An account of bringing a PET on stream in spite of a few initial bugs and limited documentation at the time.

211. Inman, Don, "The Data Handler Users Manual: Part 6", Peoples Computers 6, No. 3, pp 11-15, 44 (Nov-Dec1977)
The latest contribution in this series covers multiplication and division programs.

212. The 6502 Program Exchange, 2920 Moana, Reno, NV 89509, "Software Announcement:, On Line 2, No. 15, p 7 (Nov. 16, 1977)
Recent software includes an extended version of FOCAL, a 4K resident assembler and an efficient Mini-Editor.

213. MSS, Inc., "65XX Programs Available", New product announcement, On Line 2, No. 17, p 2 (Dec. 30, 1977)
Programs available include Disassembler, Loader, Punch, Dump, Memory Editor, Life Game, File Commands, Assembler/Text Editor, etc., MSS, Inc., 3201 East Pioneer Parkway, Suite 40, Arlington, Texas 76010.

214. Rychlewski, Walter J., III, "PET Demonstration Tape", On Line 2, No. 17, p 7, (Dec. 30, 1977), New Product Announcement.
Ten BASIC programs demonstrate most of the features of the PET; includes graphics and real time clock; $10 cassette. 603 Spruce St., Liberty, MO 64068.

215. Purser, Robert Elliott, PO Box 446, El Dorado, CA 95623, On Line 2, No. 17, p 9 (Dec. 30, 1977), New Product Announcement.
PET layout sheet with SASE, free.

216. Anon, "1K Corner: Cassette Loader and Memory Block Transfer", OSI Small Systems Journal 1, No. 5, p 3 (Nov. 1977)
With this program the user may record his own programs via the 430B Super I/O Board in a format that is recognizable to the auto-load function in the 65V Monitor PROM.

217. Anon, "Two New Software Packages", OSI Small Systems Journal 1, No. 5, pp 4-7 (Nov. 1977)
OSI has released two major new Disc software packages, Word Processor and 9-Digit BASIC which run under OS-65D version 2.0

218. Anon, "Two New Video Games", OSI Small Systems Journal 1, No. 5, pp 8-12 (Nov. 1977)
SAM (Surface-to-Air Missile) and BOMBER require OSI 8K BASIC, OSI 440 Video Board, terminal and Keyboard, and at least 4K of RAM.

219. Pfeiffer, Erich A., "Seasons Greetings", OSI Small Systems Journal 1, No. 5, p 12 (Nov. 1977)
Program using PEEK and POKE instruction to present a video message.

220. Anon, "ASCII Files under OS-65D", OSI Small Systems Journal 1, No. 5, pp 13-15 (Nov. 1977)
Auxilliary assistance program for a file system.

221. Anon, "BASIC in ROMS", New Product Announcement, OSI Small Systems Journal, 1, No. 5, p 15 (Nov. 1977)
The BASIC in ROM set No 65AB including 4 ROMS, one EPROM for the 6502 system; Another version 65VB for 440 Video system also available. Either version is $99.

222. Struve, Bill, "A $19 Music Interface", Byte 2, No. 12, pp 48-69, 170-171 (Dec. 1977)
Some theory and a KIM-1 interface for computer/music addicts.

223. Gordon, H.T., "The XF and X7 Instructions of the MOS Technology 6502", Byte Magazine 2, No. 12, p 72 (Dec. 1977)
A look at some of the unlisted instructions available in the 6502.

224. Forethought Products, PO Box 386, Coburg, OR 97401, Kilobaud, No. 12, p 15 (Dec. 1977), New Product Announcement.
A new board that makes S-100 (Altair/Imsai) type boards compatible with KIM. Motherboard has 8 slots, and does not alter the operation of KIM in any way.

225. Lancaster, Don, "TVT Hardware Design", Kilobaud, No. 12, pp 30-34 (Dec1977)
Part 1; instruction decoder and scan. Taken from Lancaster's new book, "The Cheap Video Cookbook" on the TVT-6L.

226. Blankenship, John, "Expand Your KIM!", Kilobaud, No 12, pp 36-42 (Dec 1977)
Part 2 discusses cabinet, nuts and bolts, in this series.

227. Byrd, David A., "TVT-6 Display Uncrowding", Popular Electronics 12, No. 6, p 6 (Dec. 1977)
Gives a technique for correction of a crowding of the display in Lancaster's TVT-6 Video Display.

228. Pittelkau, Clifton W., "The Bionic Clock!", 73 Magazine, No. 208, pp 102-105 (Jan. 1978)\
Software to add a real time clock to your KIM.

229. Eaton, John, "Growing with KIM", Kilobaud, No. 13, pp 36-39 (Jan. 1978)
Expansion PC Board provides compatibility with S-100 bus.

230. Chamberlin, Hal, "Software Keyboard Interface", Kilobaud, No. 13, pp 98-105 (Jan. 1978)
Shows how with a minimum of hardware and minimum cost.

231. Kraul, Douglas R., "Designing Multichannel Analog Interfaces", Byte 2, No. 2, pp 18-23 (June, 1977)
Hardware and software for an 8-channel analog I/O.

232. Fylstra, Dan, "Interfacing the IBM Selectric Keyboard Printer-Teaching KIM to Type", Byte 2, No. 6, pp 46-52, 133-139 (June 1977)
Hardware and software for hooking KIM up to a Selectric.

233. Jobs, Steven, "Interfacing the Apple Computer", Interface Age 1, No. 11, pp 65-66 (Oct. 1976)
Interfacing with a printer.

234. Wozniak, Steve and Baum, Allen, "A 6502 Disassembler from Apple", DDJ 1, No. 8, pp 22-25 (Sept. 1976)
Displays single or sequential 6502 instructions in mnemonic form.

235. Grater, Robert, "A Teletype Alternative", Kilobaud, No. 1, pp 114-116 (Jan77)
Convert parallel input TVT to serial operation, for KIM.

236. Anon."Errata to Zieglers 6502 Bug Program", DDJ 1, No. 8, p 33 (Sept. 1976)
Corrections for the listing given earlier in DDJ 1, No. 3.

237. Parks, Don, "Adding PLOP to your System", Kilobaud, No. 5, p 98 (May 1977)
A 6502 noisemaker for computer games.

238. Rankin, Roy, "Errata for Rankin's 6502 Floating Point Routines", DDJ 1, No. 10, p 57 (Nov/Dec, 1976)
Correction of a bug found in his earlier routine published in DDJ 1, No.7.

239. Lancaster, Don, "Build the TVT-6, Part II", Popular Electronics 12, No. 2 pp 49-55 (August, 1977)
System debugging, software, and how to interface to KIM and other systems.

240. The Data Mart, 914 East Waverly Drive, Arlington Heights, IL 60004, New Product Announcement, "Real Time Clock", On Line 2, No. 18, p 11 (Jan 18,1978)
Real Time Clock and Calendar for 6502. Assembled $95.

241. Optimal Technology, Inc., Blue Wood 127, Earlysville, VA 22936, Hardware Announcement: PROM Programmer, On Line 2, No. 18, p 11 (Jan 18, 1978)
Programmer for KIM programs both the 2708 and 2716 EPROMS. Runs on all 650X systems.

242. Trageser, Jim, "TVT-6L Correction", Kilobaud, No. 12, p 123 (Dec. 1977)
Corrections for the June 1977 article by Lancaster.

243. Meyers, Michael J., "Dedicated Controllers - There is Money to be Made", Kilobaud, No. 10, pp 84-92 (Oct. 1977)
Hobbyists should take advantages of opportunities to make money with their KIM or other micro.

244. Burhams, R.W., "Consider a MITE Printer", Kilobaud, No. 11, pp 38-42, (Nov. 1977)
At $276, the Mite Expandor is an alternative to the ASR-33 TTY.

245. Penhollow, Bert G.H.. "Binary to BCD Conversion for Microprocessors", Electronic Design, p 212 (Oct. 11, 1977)
Packs the units and tens into one byte.

246. Chamberlain, Hal, "Computer Bits: Computer Music Part II", Popular Electronics 10, No. 4, pp 88-91 (Oct. 1977)
A description of music techniques which have been implemented on the KIM-1 DAC board. Also discusses generation of Touch Tone codes.

247. Chamberlain, Hal, "Computer Bits: Computer Music Part I", Popular Electronics 10, No. 3, pp 116-119 (Sept. 1977)
Timed loop techniques for computer music programs.

248. Anon., "74 Megabyte Disc Review", OSI Small Systems Journal 1, No. 6, pp pp 2-6 (Dec. 1977)
OSI offers the 74 megabyte CD-74 disc drive for small computers. Has four aluminum disc platters about 12" diameter. $6000. 6502 Related.

249. Anon., "Article Sponsorship Program", OSI Small Systems Journal 1, No. 6, p 7 (Dec. 1977)
OSI will pay for and provide technical assistance for articles on OSI equipment or programs to be published in computer magazines. 6502 Related.

250. Anon., "1K Corner", OSI Small Systems Journal 1, No. 6, p 7 (Dec 1977)
Short Program for PRIME NUMBER GENERATOR.

251. Owens, Gerald, "Shoot the Gluck", OSI Small Systems Journal 1, No. 6 pp 8-10 (Dec. 1977)
A game for the 12K Challenger with video.

252. Anon., "Floppy Disk Users Group", OSI Small Systems Journal 1, No. 6 p 10 (Dec. 1977)
OSI has formed a users group to redistribute user-contributed software on diskettes. The first group of 6502 machine code programs (12 listings) is now available.

253. Anon., "Terminal/Cassette DOS Input Routine", OSI Small Systems Journal 1, No. 6 pp 11-12 (Dec. 1977)
Program for reloading or transferring program source code.

254. Anon., "New Diskette Software packages", OSI Small Systems Journal 1, No. 6, p 12, (Dec. 1977)
Work Processor WP-1 and WP-1A is a complete word processor. OS-65D Version 2.0 with Nine-digit BASIC. Disk-Test provides a quick functional check of the 6502 computer system.

255. Anon., "Bank Accounts", OSI Small Systems Journal 1, No. 6, pp13-15(Dec 1977)
Two practical programs: CHECKBOOK ACCOUNT and SAVINGS ACCOUNT.

256. Fylstra, Dan, "SWEETS for KIM", Byte 3, No. 2, pp 62-77 (Feb. 1978)
SWEETS, a Simple Way to Enter, Edit and Test Software, is a small text editor and assembler which operates on hexadecimal code and which is designed to fit in the KIM-1's 1K byte small memory while leaving room for the user's programs.

257. Feagans, John, "A Slightly Sour SWEET 16", Byte 3, No. 2, p 93 (Feb. 1978)
Correction of a slight bug in the Wozniak article in Byte, Nov. 1977.

258. Leasia, John D., "Random Errors", Byte 3, No. 2, p 93 (Feb. 1978)
Correction of an error in the pseudorandom number generator shown earlier in Byte, Nov. 1977, p 218.

259. Kathryn Atwood Enterprises, P.O. Box 5203, Orange, CA 92667, Byte 3, No. 2, p 187 (Feb. 1978), New Product Announcement
4K RAM board, KIM interface and Mother Board.

260. Electronics Warehouse Inc., 1603 Aviation Blvd., Redondo Beach CA 90278, New Product Announcement.
Apple II I/O Board Kit plugs into slot of Apple II Mother Board.

261. Pittelkau, Clifton W., "KIM-1 Can Do It!", 73 Magazine, No. 209, pp 68-71 (Feb. 1978)
Adapting a KIM-1 to function as a versatile RTTY terminal at nominal cost.

262. O'Reilly, Francis J., "Looking for a Micro?", 73 Magazine, No. 209, pp 76-77, (Feb. 1978)
Pro's and Con's of the KIM-1 as a micro.

263. Bridge, Theodore E., "A KIM-1 Disassembler", DDJ 2, No. 10, Issue 20, pp 12-13 (Nov.-Dec. 1977)
A modification of Wozniak's earlier 6502 disassembler.

264. Eaton, John, "MATHPAC: A Kimath Supplement", DDJ 2, No. 10, Issue 20, pp 15-21 (Nov.-Dec. 1977)
MATHPAC is designed to increase the power of a 6502 system. It takes the power of KIMATH and gives it to the user. The user's I/O ASCII device turns the system into a scientific calculator.

265. Osborne, Adam, "War of the Processors", SCCS Interface 1, No. 6, pp 14-17, (May, 1976)
Traces evolution of major microprocessors, including 6502 and compares their computing power.

266. Anon., "KIM-1, A complete Microcomputer System for $245", SCCS Interface 1, No. 6, pp 44-45 (May, 1976)
A new products announcement for KIM-1.

267. Teener, Mike, "Bits and Byters", SCCS Interface 1, No. 6, p 58 (May, 1976)
Historical note recaps Motorola's suit against MOS Technology over the 6502's predecessor.

268. MOS Technology, Inc., 950 Rittenhouse Road, Norristown, PA 19401, KIM Application Note #107702, "S-100 to KIM-4 Bus Adapter",
Mechanical details of a simple adapter that will plug into the KIM-4 Mother Board and which will accept certain compatible S-100 boards such as the Kent-Moore No. 60083 video display board or the Kent-Moore No. 60082 4K static RAM board.

269. MOS Technology, Inc., 950 Rittenhouse Road, Norristown, PA 19401, KIM Application Note #111477, "Using KIM as a Dedicated Controller"
The KIM itself can be used as a very low cost controller with the addition of a PROM, a power-on-reset modification, and some additional circuitry to transfer control to the added PROM upon power-up.

270. MOS Technology, Inc., 950 Rittenhouse Road, Norristown PA 19401, KIM Application Note #117701, "Digital-Analog and Analog-Digital Conversion Using the KIM-1"
This is essentially the same as Reference #172 on DeJong's article in MICRO No. 2. Uses a 1408 D/A converter with KIM together with hardware and software for D-A and A-D as well as software to store the A/D converter output and recall converted data, emulating a storage oscilloscope.

271. MOS Technology, Inc., 950 Rittenhouse Road, Norristown, PA 19401, KIM Application Note #771121, "Software Routines for TVT"
Machine Language program to use with external keyboard.

272. Optimal Technology, Inc., Blue Wood 127, Earlysville,VA 22936, On Line 3, No. 1, p 1 (Feb. 8, 1978). New Product Announcement.
2708/16 EPROM PROGRAMMER for KIM-1. Requires 1-1/2 I/O Ports. Assembled and tested $59.95. Kit $49.95.

273. Purser, POB 466, El Dorado, CA 95623, On Line 3, No.1, p 3 (Feb. 8, 1978)
Free Guidelines for writing programs for the TRS-80 and PET and then selling them to Radio Shack and Commodore. Send SASE.

274. Personal Software, P.O. Box 136-03, Cambridge MA 02138
On-Line 3 No 1 pg 4 (Feb. 8, 1987) New Product Announcement.
Four full length games on cassette for PET or TRS-80.
POKER, ONE QUEEN, KINGDOM, MATADOR; $9.95 for all four. STIMULATING SIMULATIONS by Dr. C.W. Engel, and additional entertainment personal finance/investment, and other systems programs including a 6502 Assembler in BASIC.

275. 6502 Program Exchange, 2920 Moana, Reno, NV 89509, Kilobaud, p 7 (Mar. 1978)
Announcement of new 6502 Software including an extended version of FOCAL called FCL 65E (6.5K). Also a Mini-Manual to get you started on TIM or KIM systems.

276. Eaton, John, "Corrections", Kilobaud, No. 15, p 12 (March, 1978)
Note on the availability of drilled PC boards for Eatons' KIM expansion article in January 1978 Kilobaud.

277. Scogin, Tom, "AppleSOFT Benchmarks: Fast!", Kilobaud, No. 15, p 12 (Mar 78)
Gives times for seven benchmark programs using Apple-II Integer and Apple-II AppleSOFT versions of BASIC.

278. Blankenship, John, "Expand Your KIM!", Part 4., Kilobaud, No. 15, pp 84-88 (March, 1978)
Part four of this series uses a $10 circuit board with a SWTP keyboard and a PR-40 printer as a miniature teletype.

279. Zaks, Rodney, "Micro History", Personal Computing 2, No. 2, pp 31-35, (Feb., 1978)
History of microprocessors. Has a very small paragraph on the MOS Technology 650X family.

280. DeJong, Marvin L., "Employing the KIM-1 Microcomputer as a Timer and Data Logging Module", MICRO No. 3, pp 3-7 (Feb. - Mar., 1978)
System for logging the time of up to 75 events to the nearest 100 microseconds or to other time increments, and later displaying these times on the KIM-1 display.

281. Carpenter, C.R., "Machine Language used in 'Ludwig von Apple II'", MICRO, No. 3, p 8 (Feb. - Mar. 1978)
Notes on an assembled version of the machine language used by Schwartz, MICRO, No. 2, p 19 in his music program.

282. Carpenter, C.R., "Printing with the Apple II", MICRO, No. 3, pp13-16, (Feb-Mar, 1978)
Hard-copy output from the Apple II using a TELPAR thermal printer, a simple one-transistor adapter circuit and a machine language printing routine.

283. Foreman, Evan H., P.O. Drawer F, Mobile, AL 36601, "The PET Shop", MICRO, No. 3, p 10 (Feb. - Mar. 1978)
Foreman offers to trade five game programs for the PET on a one-for-one basis.

284. Floto, Charles, "The PET VET Tackles Data Files", MICRO, No. 3, pp 9-10, (Feb. - Mar. 1978)
Discusses problems some have encountered in recording data files on tape and reading the information back in. Floto, in his capacity as the PET VET, offers his services on problems met with specific applications of PET.

285. Tater, Gary L., "Hold That Data", MICRO, No. 3, p 11 (Feb. - Mar. 1978)
Program to stop data on the video terminal by pressing a key. Handy for examining data during a disassembly or a long directory program.

286. Tripp, Robert M., "Typesetting on a 6502 System", MICRO, No. 3, pp 19-24, (Feb.-Mar. 1978)
A program for "justification" of copy to be printed.

287. Tater, Gary L., "TIM Meets the S100 Bus", MICRO, No. 3, pp 25-26 (Feb.-Mar. 1978)
A bare-bones TIM S100 board to use with a terminal such as the CT-64 from SWTP.

288. Holt, Rod, "The Apple II Power Supply Revisited", MICRO, No. 3, p 28 (Feb.-Mar. 1978)
It is pointed out that the Apple II power supply, although small in physical size, is a switching type which runs cool and is sufficient to run an Apple II with several extra cards plugged into the system.

289. Anon, "Microbes-Tiny Bugs in Previous Micros", MICRO, No. 3, p 28 (Feb-Mar)
Corrections for Ultratape, MICRO No. 1, p 13; Making Music with the KIM, MICRO No. 2, p 7; and Important Addresses of KIM-1, MICRO No. 2, p 30.

290. Husbands, Charles R., "A Simple Frequency Counter Using the KIM-1", MICRO No. 3, pp 29-32 (Feb.-Mar. 1978)
The use of KIM-1 as a counter operating over the range of 500 Hz to above 15KHz.

291. Dial, William, "6502 Bibliography-Part II", MICRO, No. 3, pp 33-36 (Feb-Mar)
The second segment of this bibliography covers references 129 to 179 of the rapidly growing 6502 literature.

292. DeJong, Marvin L., "Lighting the KIM-1 Display", MICRO, No. 3, Back Cover.
Information on how to use the KIM-1 seven-segment display.

293. Anon, "Software Sources: 6502 Executive for KIM-1", Popular Electronics 13 No. 3, p 98 (March 1978)
Adaptable to any 6502 system, this Executive is designed for KIM-1 with 4K or more and TTY or TVT interface. $25 for listing. From Innovative Software, Inc., 3007 Casa Bonita Dr., Bonita, CA 92002.

294. Pollock, James W., "Microprocessors: A Microprocessor controlled CW Keyboard" Ham Radio 11, No. 1, pp 81-87 (Jan. 1978)
A preprogrammed microcomputer is designed to function as a Morse Code keyboard. Uses a MOS Technology MCS6504 which is a software compatible cousin to the 6502.

295. Connecticut Microcomputer, 150 Pocono Rd., Brookfield, CT 06804, New Product Announcement, "RS-232 Adapter for KIM", DDJ 3, No. 21, p 3 (Jan '78)
The ADApter converts KIM's 20 ma. current loop port to an RS-232 port without affecting the baud rate. $24.50

296. Schick, Paul, "Unsupported OPCODE Pitfalls", DDJ 3, No. 21, p 3 (Jan 1978)
Comments on the earlier article on 650X Opcodes: DDJ, Aug 1977.

297. Moser, Carl, "Memory Test for 6502", DDJ 3, No. 21, pp 4-5, (Jan 1978)
A program which tests RAM memory in a 6502 based system. I/O is arranged for 6502 TIM based system but can be easily changed.

298. Smith, Stephen P., "Challenging Challenger's ROMS", DDJ 3, No. 21, p 6 (Jan)
Using the PREK function of the OSI Microsoft BASIC, a disassembler to convert stored bytes in the PROMs or ROMs has been devised.

299. Computers One, PO Box 7148, Honolulu, HI 96821, New Product Announcement, On Line 3, No. 2, p 4 (March 1, 1978)
Pre-recorded programs for PET. "HUSTLERS" includes a number of business oriented programs for checking accounts, rent accounts, legal dairy and trust accounts.

300. Lufkin, C.R., 315 Dominion Dr., Newport News, VA 23602, On-Line 3, No 2, p 5 (March 1, 1978)
FITABP is Federal Income Tax Program for PET owners with 8K. Prints out form 1040 Schedule A and B.

6502 BIBLIOGRAPHY
PART IV

William Dial
438 Roslyn Avenue
Akron, OH 44320

301. Michels, Richard E. "How to Buy an Apartment Building", Interface Age 3, No. 1, pp 94-99 (Jan 1978)
A 6502 FOCAL based system for handling the many factors involved via a computer decision making program.

302. Woods, Larry "How Are You Feeling Today?" Kilobaud No.14,pp24-30(Feb 1978)
Biorhythms with your KIM are displayed on the KIM readout.

303. Craig, John "Editor's Remarks" Kilobaud No. 14 p 22 (Feb 1978)
In a discussion of Microsoft Level II BASIC it is pointed out that Microsoft BASIC is being used on Altair 6800 and 8080, TRS-80, and 6502 based systems OSI, PET, KIM and Apple (floating-point version).

304. Bishop, Robert J. "Star Wars" Kilobaud No. 14 pp52-56 (Feb. 1978)
An Apple-II graphics game based on the 6502.

305. Blankenship, John "Expand Your KIM! Part 3" Kilobaud pp68-71 (Feb. 1978)
This installment covers bus control board and memory.

306. Burhans, R.W. "How Much Memory for a KIM?" Kilobaud p 118 (Feb. 1978)
Decoding the KIM for 28K.

307. Pearce, Craig A., p.6 suggestions for running graphics on the PET.
Julin, George, pp6-7, letter on PET graphics.
Stuck, H.L. p 7, more on the PET.
Above three are letters in Peoples Computers 6, No. 4, (Jan-Feb. 1978)

308. Wells, Edna H. "Program Abstract" Peoples Computers p 7 (Jan-Feb. 1978)
Program for the Commodore PET with 8K BASIC, entitled Graphics-to ASCII Utility--ASCIIGRAPH.

309. Cole, Phyllis "SPOT-The Society of PET Owners and Trainers", Peoples Computers No. 4, pp 16-19 (Jan-Feb 1978)
Notes for the Users of the PET.

310. Inman, Don "The Data Handler User's Manual: Conclusion" Peoples Computers No. 4 pp24-31 (Jan-Feb 1978)
The final installment of this series covers simple and inexpensive output devices.

311. Inman, Don "The First Book of KIM, Peoples Computers No. 4 p34 (Jan-Feb1978)
A good review of this excellent book.

312. Braun, Ludwig "Magic for Educators--Microcomputers" Personal Computing, 2 No. 1, pp 30-40 (Jan. 1978)
Discussion of micros includes the 6502 based Apple II and the PET.

313. Helmers, Carl "An Apple to Byte", BYTE 3, No. 3, p. 18-46 (Mar. 1978)
A user's reactions to the Apple II, with an example of a simple "color sketchboard" application.

314. Fylstra, Dan "User's Report: The PET 2001", BYTE,pp114-127 (Mar. 1978)
A fairly comprehensive report on the PET.

315. Brader, David, "KOMPUUTAR Updates", BYTE pp131-132 (Mar. 1978)
In a letter Brader responds to some inquiries on his KOMPUUTAR system based on 6502 which was published in BYTE, Nov. 1977.

316. Jennings, Peter R., "Microchess 1.5 versus Dark Horse", BYTE 3, No. 3 pp 166-167 (March 1978)
Microchess 1.5 is Jenning's new extended version of the original Microchess. It occupies 2.5K and runs on KIM-1 with expanded memory. It is still being developed but in a test game with Dark Horse, a 24K program written in Fortran IV, the new version did very well indeed.

317. Rindsberg, Don "Here's HUEY!...super calculator for the 6502", Kilobaud, No. 12, pp 94-99 (December 1977)
The calculating power of FORTRAN with trig functions, natural and common logs, exponential functions, all in 2.5K.

318. Finkel, LeRoy "Every Home (School) Should Have a PET" Calculators/Computers, page 83 (October 1977)

319. Anon, "12-Test Benchmark Study Results Show How Microprocessors Stack Up (8080, 6800, 6502)", EDN, page 19 (November 20, 1977)

320. Gordon, H.T. "Decoding Efficiency and Speed", DDJ 3, Issue 2, No. 22, pp 5-7 (Feb., 1978)
Pros and cons of table look-up in 650X microprocessors.

321. Green, Wayne "Publishers Remarks", Kilobaud, No. 16, p 4 (April 1978)
In a column on microprocessors, Green indicates that MOS Technology has a SuperKIM being readied and also that books on expanding the KIM system are coming out.

322. Carpenter, Chuck "Letters: KIM-1, ACT-1; The Scene", Kilobaud p 18 (Apr 1978)
A generally favorable report of one user's experience in interfacing and using ACT-1 with the KIM-1.

323. Braun, Ludwig, "PET Problems", Personal Computing No. 3, pp 5-6 (March 1978)
Some observations by a PET owner.

324. Lasher, Dana "The Exterminator--for Buggy KIMs" 73 Magazine (April, 1978)
Hardware and Software for a debugging facility.

325. Eaton, John "Now Anyone Can Afford a Keyboard" 73 Magazine (April, 1978)
A melding of a surplus keyboard, KIM and software.

326. Foster, Caxton C. "Programming a Microcomputer: 6502" Addison-Wesley Publishing Company, Reading, Mass. 1978
Caxton C. Foster of the University of Massachusetts, Amherst, has put together a very helpful book on programming the 6502 using KIM-1 as a lab tool.

327. Barden, William, Jr. "Computer Corner - 6502" Radio-Electronics (May 1978)
An in-depth look at the widely used 6502 microprocessor.

328. Wozniak, Steve "Renumbering and Appending Basic Programs on the Apple-II Computer" DDJ Issue 3 (March 1978)
Comments and techniques for joining two BASIC programs into a single larger one.

329. Eaton, John "A KIM Binary Calculator" DDJ Issue 3 (March 1978)
An easier way to solve binary math programs.

330. Wells, Ralph "PET's First Report Card" Kilobaud pp 22-30 (May 1978)
An objective evaluation of PET serial No. 171.

331. Blankenship, John "Expand you KIM" Kilobaud, pp 60-63 (May 1978)
Part 5: A/D interfacing for joysticks. Four channels.

332. Holland, Hugh C. "KIM Notes" BYTE 3 No. 4, p 163 (April 1978)
Correction for Hal Chamberlin's Four Part Harmony Program published in September 1977 BYTE.

333. Anon., "Byte's Bits", BYTE 3, No. 4, p 166 (April 1978)
Notes on picking the right color television for an Apple.

334. KIM-1 User Notes, Issue 9/10, (January - March 1978)
Rehnke, Eric "Have you been on the Bus" page 1.
Kushnier, Ron "Space War Phaser Sound" page 2.
Butterfield, Jim "Skeet Shoot" page 2.
Edwards, Lew "KIM D-BUG" page 3.
Flacco, Roy "Graphics Interface" page 4.
Wood, James "RPN Calculator Interface to KIM" page 6.
Bennett, Timothy "KUN Index by Subject, Issues 1 to 6" page 12.
Niessen, Ron "On Verifying Programs in RAM" page 12.
Pottinger, Hardy "Greeting Card Generator" page 13.

6502 BIBLIOGRAPHY
PART V

William Dial
438 Roslyn Avenue
Akron, OH 44320

335. Smith, Stephen P. "6502 Disassembler Fix", DDJ 3, No. 23, Issue 3, Pg 3 (March 1978)
ROR and ROL instructions were omitted in the previously published disassembler - DDJ 3, Issue 1. This offers a simple fix.

336. KIM-1 User Notes, Issue 9/10, (January - March 1978)
Butterfield, Jim "Dicey" page 17. A program to roll up to six dice.
Butterfield, Jim "Teaser" page 17. Jumbo version of Bob Albrecht's "Shooting Stars".
Lewart, Cass "Correction for Lancaster's TVT" page 20.
Oliver, John P. "Comments and Corrections for SUPERDUMP/LOAD" pg 21.

337. Quosig, Karl and Susan "Input/Output", Personal Computing 2, No. 4, pg 8 (April 1978).
Comments on PET problems.

338. Bishop, Robert J. "Rocket Pilot", Kilobaud No. 13, pg 90 (Jan. 1978)
And interactive game for the Apple II.

339. OSI-Small Systems Journal 2, No. 1 (January-February 1978)
Anon. "What's a USR Function". Via the USR function, one can have a 6502 BASIC program which works in conjunction with one or several machine code programs.
Anon. "Quickie". A 6502 BASIC program for converting decimal to binary numbers.
Glasser, Daniel "Chessboard". Program in 6502 BASIC for a computer chessboard which moves pieces and displays the new board. Not a chess program.
Anon. "DOS CNTRL". A BASIC program to perform transfers to or from OSI's new hard disk drive.
Anon. "Track Zero Writer". A Machine language program to modify track zero.
Anon. "9 Digit BASIC". A concise method for modifying OSI 9 Digit BASIC for an end-user 9 Digit BASIC.
Anon. "OS-65U Performs". A description of a new system said to be a new standard for microcomputer operating systems.
Anon. "500/510 Breakpoint Utilities". A breakpoint program.
Anon. "510 Tracer". A tracer program which prints a disassemble of the next instruction to be executed.

340. Bishop, Robert J. "Fiendish New QUBIC Program", 73 Magazine, No. 209, pg 78 (Feb 1978).
An attempt at producing an improved version of the original Qubic program.

341. Rosner, Richard "Daddy, Is It The PET?", ROM 1, No. 9, pg 26 (Mar/April 1978)
Description of many features and operations of the PET, including many "how to" instructions.

342. Bishop, Robert J. "LOGAN - A Logic Circuit Analysis Program", Interface Age 2, No. 6, pg 128 (May 1977). An Apple I BASIC program for analyzing networks of logic gates.

343. Bishop, Robert J. "Apple Star Trek", Interface Age 2, No. 6, pg 132 (May 1977).
Star Trek written in Apple I BASIC.

344. Chamberlin, Hal "Microcomputer Input/Output", Popular Electronics 13, No. 5, pg 86 (May 1978).
Comments on the KIM's memory-mapped I/O system.

345. Peoples Computers 6, No. 6 (May/June 1978)
Johnson, Ralph "Letters". The University of California at San Diego plans a Pascal system for the 6502.
Cole, Phyllis "Apple II". A review of this 6502 based micro.
Voros, Todd L. "Sketchcode". A technique to minimize errors and simplify the process of debugging. Listed in 6502 assembly code.
Offen, Dave "Kaleidoscope". A continuously running graphics program for the PET.
Hofheintz, M. C. "Tiny GRAPHICS". A short graphics program for the PET.

346. Gordon, H. T. "Editha", DDJ 3, Issue 5, No. 25, pg 34 (May 1978). A revision of the Fylstra KIM-1 Editor program "SWEETS" published in BYTE.

347. Tullock, Michael "PET Files", Personal Computing 2, No. 5, pg 20 (May 1978). Things your user's manual never told you about PET. How to use files.

348. O'Reilly, Francis J. "Instruction Search", Byte 3, No. 5, pg 153 (May 1978). Discussion of 6502 op code 27 and the search for other as yet undefined instructions.

349. Carpenter, Charles R. "Tiny BASIC Shortcuts", Kilobaud, Issue 18, pg 42 (June 1978). Suggests methods to expand the capabilities of Tom Pittman's Tiny BASIC for the 6502.

350. O'Haver, T. C. "More Music for the 6502", Byte 3, No. 6, pg 140 (June 1978). A music composition and generation program.

351. O'Haver, T. C. "Audio Processing with a Microcomputer", Byte 3, No. 6, pg 166 (June 1978). Adding a virtual tape loop. Uses a 6502 processor.

352. Eaton, John "Low Cost Keyboard - II", 73 Magazine, No 213, pg 100 (June 1978). Part II of an article on the low-cost keyboard. Software is designed around the 6502.

353. Swindle, David "A Sensible Expansion: Atwood Memory for your KIM", Kilobaud, Issue 19, pg 60 (July 1978). Description of a low cost method to add memory to KIM.

354. MICRO, Issue 4 (April/May 1978)

Carpenter, C. R. "Variables Chart". Chart to layout and keep track of string and numerical variables for Apple II Applesoft BASIC.

Floto, Charles "The PET Vet Examines Some BASIC Idiosyncrasies". Includes suggestions and modifications for a Mailing List Program by Richard Rosner.

DeJong, Marvin L. "A Complete Morse Code Send/Receive Program for the KIM-1". Converts ASCII from a keyboard to a Morse code digital signal and also converts a Morse code digital signal to an ASCII code for display on a video system.

O'Brien "PET Software from Commodore". New selected Application notes from Commodore.

Floto, Charles "Early PET-Compatible Products". A review of several new accessories for the PET.

Rowe, Mike "The MICRO Software Catalog". A continuing catalog of software available for 6502 based systems.

Carpenter, C. R. "Apple II Printing Update". Updated information and modifications of the system described previously in MICRO No. 3.

Chamberlin, Hal "Standard 6502 Assembly Syntax?". A plea for standardization.

Rowe, Mike "A Worm in the Apple". Discussion of some problems encountered in interfacing the Apple to other devices such as the 6820 PIA.

Jenkins, Gerald C. "A KIM Beeper". A short blast or two of audio for load errors, end-of-line, etc.

Auricchio, Rick "An Apple II Programmer's Guide". Some of the previously undisclosed details of the Apple Monitor.

355. O'Connor, Clint "Book Review: Programming a Microcomputer: 6502", Kilobaud, Issue 20, pg 8 (August 1978). A very favorable review of Caxton C. Foster's book.

356. Grossman, Rick "KIM Plus Chess Equals Microchess", Kilobaud, Issue 20, pg 74 (August 1978). A challenging game of Chess can be played in KIM's 1K of memroy using MicroChess by Peter Jennings.

357. Palenik, Les "FINANC - A Home/Small-Business Financial Package", Kilobaud, Issue 20, pg 84 (August 1978). Programs include Calculations on investments, Depreciation, Loans, etc.

358. Braun, Ludwig "Commodore PET", Creative Computing 4, No. 4, pg 24 (July/August 1978)

359. Creative Computing 4, No. 4 (July/August 1978).

Braun, Ludwig "Commodore Pet". An equipment profile which stresses the value of the PET as a teaching machine.

North, Steve "Apple II Computer". An equipment profile points out that the Apple is not a machine for the classroom or for the S-100 hardware buff but is one of the most versatile micros on the market.

Dawkins, Gary D. "High-Resolution Graphics for the Apple II". Allows user to draw a shape in high-resolution graphics mode from the keyboard.

Ahl, David H. "Atari Video Computer System". An equipment profile of a 6505 based programmable game system.

360. MICRO, Issue 5 (June/July 1978)

Covitz, Frank H. "Life for your PET". LIFE written in machine language for the PET.

Rockwell International ""Rockwell's New R6500/1". The 6500/1 is a single chip NMOS microcomputer, 1 or 2 MHz, fully compatible with the 6500 family.

De Jong, Marvin L. "6502 Interfacing for Beginners: Address Decoding I". The first installment in a continuing series.

Rowe, Mike "Half a Worm in the Apple". More on the controversy on interfacing the Apple to PIA's. See also EDN May 20, 1978.

Sander-Cederlof, Bob "A Slow List for Apple BASIC". Program slows down the list process so it can be more easily reviewed.

Rowe, Mike "The Micro Software Catalog: II". The second part of this continuing series.

Synertek Inc. "Synertek's VIM-1". A good description of the many features of the 6502 based VIM-1. Similar to and compatible with KIM-1 with some new features.

Suitor, Richard F. "Applayer Music Interpreter". A music interpreter written in 6502 assembly language for the Apple, but can be used on other 6502 systems.

Dial, William "6502 Bibliography - Part IV". The fourth part of the continuing bibliography of the 6502 literature (of which this is the fifth part!).

Williams, J. C. "A Block Hex Dump and Character Map Utility Program for the KIM-1". A fully relocatable utility program which will dump a specified block of memory from a KIM to a terminal in several formats.

Rockwell International "Rockwell's AIM is Pretty Good". Rockwell's AIM 65 is an assembled versatile microcomputer system on one board plus keyboard. It has a 20-character display and a 20-character thermal printer, 4K ROM monitor, 1K RAM expandable on board to 4K. Application and Expansion connectors are fully KIM-1 compatible. TTY and Audio Cassette, DEBUG/MONITOR/ ROM or EPROM on board up to 16K. 8K BASIC will be available in ROM.

Carpenter, Chuck "Apple II Accessories and Software". Items reviewed include a renumber and append program, a serial interface board, a MODEM, Applesoft II, and the "APPLE II BASIC Programming Manual.

McCann, Michael J. "A BASIC 6502 Disassembler for Apple and PET". Accepts machine language -object code- and produces a symbolic representation that resembles an assembly listing. Originally written in Commodore BASIC, it will work with Applesoft BASIC as well.

LDA	AD	LDAIM	A9	LDAZ	A5	LDAIX	A1	LDAIY	B1	LDAZX	B5	LDAX	BD	LDAY	B9	N Z
STA	8D			STAZ	85	STAIX	81	STAIY	91	STAZX	95	STAX	9D	STAY	99	
ADC	6D	ADCIM	69	ADCZ	65	ADCIX	61	ADCIY	71	ADCZX	75	ADCX	7D	ADCY	79	N Z C V
SBC	ED	SBCIM	E9	SBCZ	E5	SBCIX	E1	SBCIY	F1	SBCZX	F5	SBCX	FD	SBCY	F9	N Z C V
AND	2D	ANDIM	29	ANDZ	25	ANDIX	21	ANDIY	31	ANDZX	35	ANDX	3D	ANDY	39	N Z
EOR	4D	EORIM	49	EORZ	45	EORIX	41	EORIY	51	EORZX	55	EORX	5D	EORY	59	N Z
ORA	0D	ORAIM	09	ORAZ	05	ORAIX	01	ORAIY	11	ORAZX	15	ORAX	1D	ORAY	19	N Z
CMP	CD	CMPIM	C9	CMPZ	C5	CMPIX	C1	CMPIY	D1	CMPZX	D5	CMPX	DD	CMPY	D9	N Z C
ASL	0E	ASLA	0A	ASLZ	06					ASLZX	16	ASLX	1E			N Z C
LSR	4E	LSRA	4A	LSRZ	46					LSRZX	56	LSRX	5E			N Z C
ROL	2E	ROLA	2A	ROLZ	26					ROLZX	36	ROLX	3E			N Z C
ROR	6E	RORA	6A	RORZ	66					RORZX	76	RORX	7E			N Z C
DEC	CE			DECZ	C6					DECZX	D6	DECX	DE			N Z
INC	EE			INCZ	E6					INCZX	F6	INCX	FE			N Z
BIT	2C			BITZ	24											7 Z 6
LDX	AE	LDXIM	A2	LDXZ	A6					LDXZY	B6			LDXY	BE	N Z
STX	8E			STXZ	86					STXZY	96					
CPX	EC	CPXIM	E0	CPXZ	E4											N Z C
DEX	CA	INX	E8													N Z
LDY	AC	LDYIM	A0	LDYZ	A4					LDYZX	B4	LDYX	BC			N Z
STY	8C			STYZ	84					STYZX	94					
CPY	EC	CPYIM	C0	CPYZ	C4											N Z C
DEY	88	INY	C8													N Z

BPL	10	BMI	30	BVC	50	BVS	70	BCC	90	BCS	B0	BNE	D0	BEQ F0
CLC	18	SEC	38	CLI	58	SEI	78			CLV	B8	CLD	D8	SED F8
JMP	4C	JMPI	6C	JSR	20	RTS	60	RTI	40	BRK	00	NOP	EA	
TAX	AA	TXA	8A	TAY	A8	TYA	98	TSX	BA	TXS	9A			N Z
PHA	48	PLA	68	PHP	08	PLP	28 (Flags Restored)							N Z

I = Indirect
IM = Immediate
X = absolute indexed by X
IX = Indexed indirect by X
ZX = Zero page indexed by X
A = Accumulator
Z = Zero page
Y = absolute indexed by Y
IY = Indirect indexed by Y
ZY = Zero page indexed by Y
No extension for Relative, Implied or Absolute addressing modes.

LEAST SIGNIFICANT DIGIT

MOST SIGNIFICANT DIGIT	0	1	2	4	5	6	8	9	A	C	D	E
0	BRK	ORAIX			ORAZ	ASLZ	PHP	ORAIM	ASLA		ORA	ASL
1	BPL	ORAIY			ORAZX	ASLZX	CLC	ORAY			ORAX	ASLX
2	JSR	ANDIX		BITZ	ANDZ	ROLZ	PLP	ANDIM	ROLA	BIT	AND	ROL
3	BMI	ANDIY			ANDZX	ROLZX	SEC	ANDY			ANDX	ROLX
4	RTI	EORIX			EORZ	LSRZ	PHA	EORIM	LSRA	JMP	EOR	LSR
5	BVC	EORIY			EORZX	LSRZX	CLI	EORY			EORX	LSRX
6	RTS	ADCIX			ADCZ	RORZ	PLA	ADCIM	RORA	JMPI	ADC	ROR
7	BVS	ADCIY			ADCZX	RORZX	SEI	ADCY			ADCX	RORX
8		STAIX		STYZ	STAZ	STXZ	DEY		TXA	STY	STA	STX
9	BCC	STAIY		STYZX	STAZX	STXZY	TYA	STAY	TXS		STAX	
A	LDYIM	LDAIX	LDXIM	LDYZ	LDAZ	LDXZ	TAY	LDAIM	TAX	LDY	LDA	LDX
B	BCS	LDAIY		LDYZX	LDAZX	LDXZY	CLV	LDAY	TSX	LDYX	LDAX	LDXY
C	CPYIM	CMPIX		CPYZ	CMPZ	DECZ	INY	CMPIM	DEX	CPY	CMP	DEC
D	BNE	CMPIY			CMPZX	DECZX	CLD	CMPY			CMPX	DECX
E	CPXIM	SBCIX		CPXZ	SBCZ	INCZ	INX	SBCIM	NOP	CPX	SBC	INC
F	BEQ	SBCIY			SBCZX	INCZX	SED	SBCY			SBCX	INCX

ASCII CONVERSION TABLE

HEX	BITS	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0 0 0 0	NUL	DLE	SPACE	0	@	P	`	p
1	0 0 0 1	SOH	DC1	!	1	A	Q	a	q
2	0 0 1 0	STX	DC2	"	2	B	R	b	r
3	0 0 1 1	ETX	DC3	#	3	C	S	c	s
4	0 1 0 0	EOT	DC4	$	4	D	T	d	t
5	0 1 0 1	ENQ	NAK	%	5	E	U	e	u
6	0 1 1 0	ACK	SYN	&	6	F	V	f	v
7	0 1 1 1	BEL	ETB	'	7	G	W	g	w
8	1 0 0 0	BS	CAN	(	8	H	X	h	x
9	1 0 0 1	HT	EM	)	9	I	Y	i	y
A	1 0 1 0	LF	SUB	*	:	J	Z	j	z
B	1 0 1 1	VT	ESC	+	;	K	[	k	{
C	1 1 0 0	FF	FS	,	<	L	\	l	¦
D	1 1 0 1	CR	GS	-	=	M	]	m	}
E	1 1 1 0	SO	RS	.	>	N	^	n	~
F	1 1 1 1	SI	US	/	?	O	_	o	DEL

HEXIDECIMAL CONVERSION TABLE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440