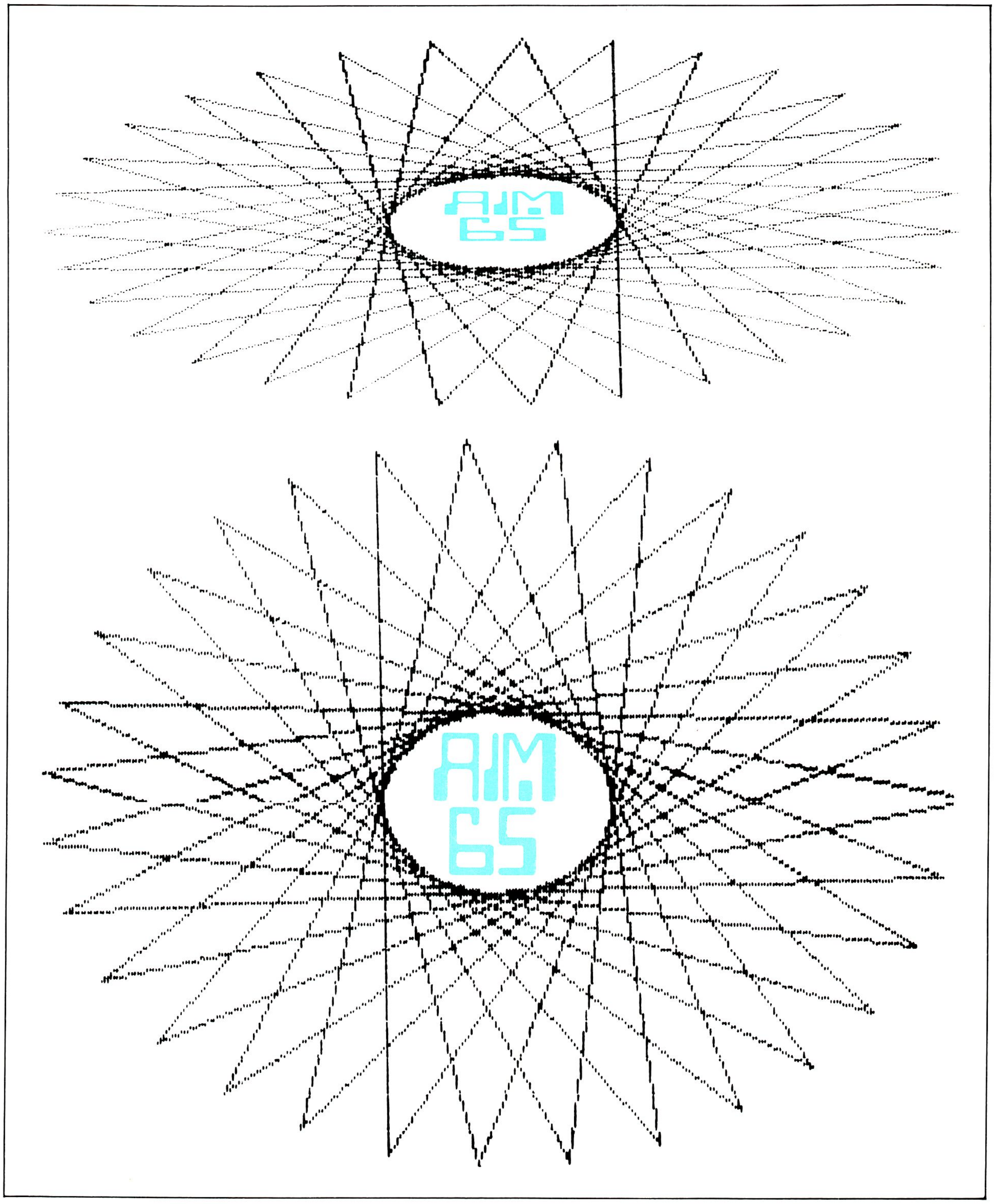
# INTERACTIVE

ISSUE NO. 2



Rockwell International

...where science gets down to business

## EDITOR'S CORNER

Your response to the questions on the subscription envelope has been gratifying. By far, most of you are interested in articles about interfacing AIM 65 to the outside world, (especially floppy disks) and finding out who makes what for the system. I'm going to do my bst to give you what you want in the way of subject matter, and hopefully you'll keep me informed if your needs should change.

### ESSENCE OF AIM (65)

A computer is a computer is a computer. That's obvious. But the fact remains that some computers can do certain things better than others. Look at people. The same person that would make a great jockey would probably make a lousy long distance runner (and vice versa).

To hear some people talk, you'd think the AIM 65 is great at everything. Well, you and I, being realists, KNOW that that's not true. The AIM 65, like any other computer, has its good points and its not-so-good points. While some of the no-so-good points can be improved upon (see the article in this issue on adding a sound channel to the AIM), I would most like to see articles that expand upon and accentuate AIM 65's strong points.

Here are some applications in which AIM 65 excels:

*low-cost, self-contained educational system.
*laboratory instrumentation monitoring and experiment control computer.
*minimum-cost software/hardware development system.
*remote communications terminal (by adding a MODEM)
*control panel and "smarts" for OEM machine or assembly-line controller
*intelligent, general-purpose calculator
*low-and medium-volume OEM products, with PROM-selected multiple "personalities"
*Any product requiring a minimal hard-copy capability

I'll bet that you can think of several more.....

### THIS ISSUE

You'll notice that we have plenty of AIM 65 graphics in this issue. This capability adds a whole new dimension to the usefulness of the machine and is quite exciting. Thanks for this ability must go first to the AIM 65 designers who used a software approach for interfacing the printer and next to the folks at Micro Technology Unlimited and Micro Mag who actually did the graphics software and made it available to the rest of the world (separately, I might add).

Eric C. Rehnke

EDITOR

## FOR YOUR INFORMATION

Here are some phone numbers that should prove useful to you:

| | |
|---|---|
| **AIM 65 APPLICATIONS** | (714) 632-0975 Use this number when you have technical questions concerning the AIM 65 system or are having difficulty getting the AIM 65 to function properly. |
| **DEVICE APPLICATIONS** | (714) 632-3860 Use this number when you have technical questions concerning individual 6500 family devices whether or not they are on the AIM 65. |
| **SERVICE INFORMATION** | 800-351-6018 Call this number when your AIM 65 is broken and needs to be repaired. |
| **LITERATURE** | (714) 632-3729 Call this number when you need literature for a certain Rockwell product or a particular application note. |
| **AIM 65 SALES INFORMATION** | 800-854-8099 (in California, call 800-422-4230) Use this number when you are wondering where you can purchase an AIM 65 or Rockwell accessory item. |
| **AIM 65 DOCUMENTATION** | (714) 632-3729 Ask to speak to the Documentation Manager if you have a question about the documentation or a problem with it. |

To keep receiving this newsletter, subscribe now! The cost is $5 for 6 issues ($8 overseas). Just fill in the attached subscription application, add your check or money order *(NO CASH OR PURCHASE ORDERS WILL BE ACCEPTED)* and mail it in using the envelope. (Payment must be in U.S. funds drawn on a U.S. bank).

All correspondence and articles should be sent to:

**NEWSLETTER EDITOR**
**ROCKWELL INTERNATIONAL**
**POB 3669, RC 55**
**ANAHEIM, CA 92803**

# COVER STORY

## AIM 65 GRAPHICS SOFTWARE

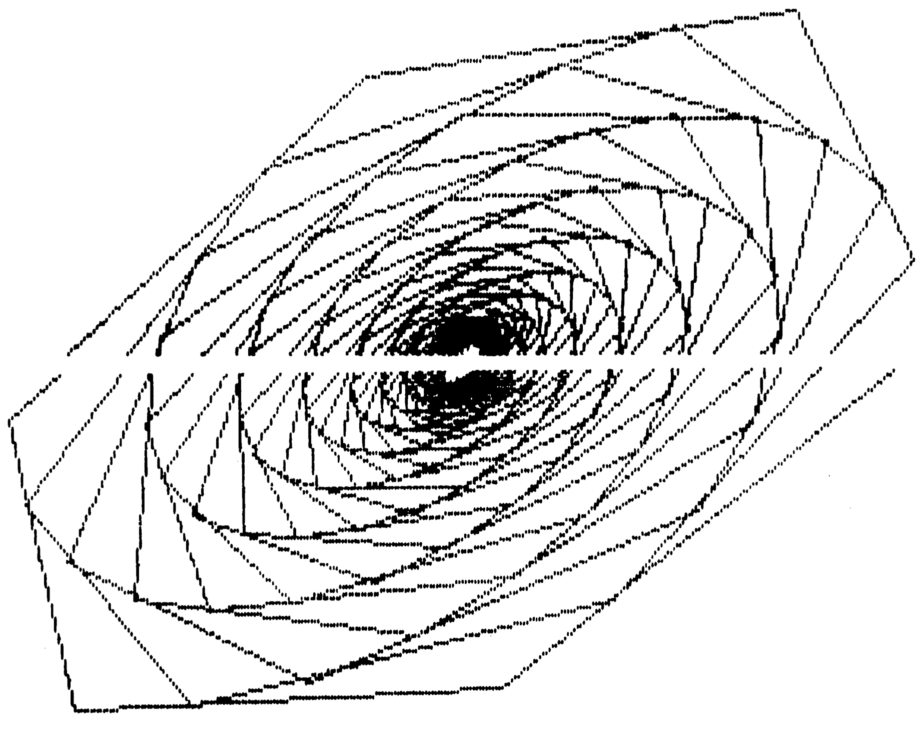
Would you believe that the graphics on the front cover (except for the lettering) were generated with an AIM 65? Well, it's true. Of course a little help was needed in the way of software since, by its lonesome, AIM 65 isn't so artistic. That help comes in the form of some creative software instruction from the folks at Micro Technology Unlimited (POB 12106, Raleigh, NC 27605 (919) 833-1458).

MTU supports AIM 65 in several ways. They manufacture hardware expansion accessories (see the list in the AIM 65 suppliers section of this issue), AND several software packages. These software packages greatly enhance the capability of the AIM 65 in several ways.

The first package is called the TEXT/GRAPHICS PRINTOUT PROGRAM FOR THE AIM 65 (K-1009-1C) and includes two programs. One of them dumps the contents of the text editing buffer out to the printer sideways. That's right, SIDEWAYS. With line lengths of to 80 characters and 10 lines per strip, AIM's printouts become much easier to read. (I just couldn't believe my eyes the first time I saw this work. It's really incredible!) I wish MTU would release the source code on this program so people could tie this into the assembler and BASIC. Now that would REALLY make AIM 65 shine!

The second part of this printout program is the one responsible for the neat designs on the cover of this issue. It's purpose is to give AIM 65 users a hard copy record (in one of two modes) of whatever is displayed on the MTU Visible Memory (320×200 bit-mapped graphics board). (This is an 8K dynamic RAM board that doubles as a video-graphics display when connected to a video monitor.) The "quick print" mode lets you print out the entire 320×200 dot image on one strip of paper while the "quality print" mode prints out the image as two strips of 320×100 each which can then be taped together for a complete, properly proportioned image. (see the cover for an example of each) Of course, the printout program doesn't really care what 8K memory location the pattern is coming from so patterns can be written into ANY memory board, or even taken from ROM, if desired. But the greatest impact and practicality will be achieved when this program is used in conjunction with the MTU graphics board.

The second package is called AIM 65 GRAPHICS/TEXT SOFTWARE (K-1008-5C) and contains such goodies as an interface program which allows graphics to be generated directly from an AIM 65 BASIC program, a program which turns the Visible Memory board into a 53 character by 22 line video display for AIM 65, a swirl pattern generator, a 320×200 Life game, a graphics subroutine library, and several BASIC demo programs thrown in for good measure.



# AIM 65 GRAPHICS

(The next two articles are being reprinted with permission from the publisher of 65XX MICRO-MAG, a German publication dedicated to 6502 based machines. 65XX MICRO-MAG is written almost entirely in German so it would be useful to have a command of the language. If not, we'll be translating some of the AIM 65 articles and reprinting them in future issues of INTERACTIVE. Thanks go to Roland Lohr (Hansdorfer Strasse 4, 2070 Ahrensburg, W. Germany)

## AIMPLOT — PLOTTING MEASUREMENT VALUES

This utility plots the results of measurements on the AIM printer at a speed of 9 dots per sec. VALDOT converts a parameter in A into a dot position (hex 00 A 63), AIGRA does the printout.

By means of the subprograms presented here, the printer of AIM 65 becomes a measurement value plotter, which outputs about 9 values per second. VALDOT converts a measurement value in the accumulator into the corresponding measurement point position. AIGRA takes care of printing out this dot. The user therefore only has to convert his measurement value into the hexidecimal value range 00-63 capable of presentation.

With regard to the way in which the printer works, one should familiarize himself with the AIM USER's GUIDE, pages 7-19 ff. There in particular one is warned against manipulating the timing of the printer. In this respect the user need have no fear, because the author was able to return to the original routines of the monitor with its time constants unchanged. With regard to commentary, reference is made for the most part to the MONITOR PROGRAM LISTING.

```
0200  2C 11 A4  AIGRA   BIT  PRIFLAG   ROUTINE CORRESPONDS
                                       TO IPST IN
0203  10 2A             BPL  OUT       $F045 FOR OUTPUT OF A
                                       LINE.
0205  20 CB F0          JSR  PINT      INITIALIZE
0208  20 66 02          JSR  NIPSU
020B  A9 C1             LDA  #$C1
020D  8D 0C A8          STA  PCR
0210  20 A0 FF          JSR  PAT23
0213  D0 08             BNE  NIP02
0215  20 A0 FF          JSR  PAT23
0218  D0 03             BNE  NIP02
021A  4C 79 F0          JMP  PRIERR
021D  20 30 02  NIP02   JSR  NPDOT
0220  20 30 02          JSR  NPDOT
0223  AD 77 A4          LDA  IDOT
0226  C9 0A             CMP  #$0A      ONLY 1 LINE
0228  90 F3             BCC  NIP02
022A  A9 E1             LDA  #$E1
022C  8D 0C A8          STA  PCR       MOTOR OFF
022F  60        OUT     RTS
0230  A9 00     NPDOT   LDA  #$00      ROUTINE CORRESPONDS
0232  BD 01 A8          STA  DRAH      TO PRNDOT IN $F087
0235  AD 0D A8  NDOT0   LDA  IFR
0238  29 02             AND  #$02
023A  F0 F9             BEQ  NDOT0
023C  AD 0C A8          LDA  PCR
023F  49 01             EOR  #$01
0241  8D 0C A8          STA  PCR
0244  EE 77 A4          INC  IDOT
0247  AD 79 A4          LDA  IOUTU
024A  0D 00 A8          ORA  DRB
024D  8D 00 A8          STA  DRB
0250  AD 78 A4          LDA  IOUTL
0253  8D 01 A8          STA  DRAH
0256  A9 A4             LDA  #$A4
0258  8D 08 A8          STA  T2L
025B  A9 06             LDA  #$06
025D  8D 09 A8          STA  T2H
0260  20 66 02          JSR  NIPSU
0263  4C BA F0          JMP  $F0BA     CARRY OUT THE REST OF
                                       ROUTINE PRNDOT
0266  A2 00     NIPSU   LDX  #$00      ROUTINE CORRESPONDS
0268  20 21 F1          JSR  INCP      TO PRNDOT IN $F0E3
026B  BD 60 A4  NIPS1   LDA  IBUFM,X
026E  CD 77 A4          CMP  IDOT
0271  D0 16             BNE  NIPS3
0273  AD 7A A4          LDA  IBITL
0276  F0 08             BEQ  NIPS2
0278  0D 78 A4          ORA  IOUTL
027B  8D 78 A4          STA  IOUTL
027E  D0 09             BNE  NIPS3
0280  AD 7B A4  NIPS2   LDA  IBITU
0283  0D 79 A4          ORA  IOUTU
0286  8D 79 A4          STA  IOUTU
0289  0E 7A A4  NIPS3   ASL  IBITL
028C  2E 7B A4          ROL  IBITU
028F  CA CA             DEX, DEX
0291  10 D8             BPL  NIPS1
0293  4C 18 F1          JMP  $F118     TO THE REMAINDER OF
                                       ROUTINE IPSU
                CALCULATE DOT POSITION FROM VALUE IN A
0296  48        VALDOT  PHA            RESCUE PARAMETER
0297  A2 00             LDX  #$00      ERASE PRINT BUFFER COM-
                                       PLETELY
0299  20 38 F0          JSR  OUTPR
029C  A2 00             LDX  #$00      X AS ADDRESSER RETRIEVE
                                       VALUE
029E  68                PLA
029F  C9 05     DIVA    CMP  #$05
02A1  90 05             BCC  FEIN      REMAINDER <5
02A3  E9 05             SBC  #$05      DIVIDE BY 5 UNTIL REMAIN-
                                       DER <5
02A5  E8                INX            ADDRESSER + 1
02A6  D0 F7     BNE     DIVA           ALWAYS JUMP
02A8  18        FEIN    CLC            ADDITION PREPARATION
02A9  2C 82 EF          BIT  #$04      OPERAND FROM FIXED
                                       VALUE STORAGE
02AC  08                PHP            RESCUE STATUS
02AD  49 03             EOR  #$03      INVERT 2 BITS
02AF  69 01             ADC  #$01      COMPUTATION IN THE
                                       4-PART COMPLEMENT
02B1  28                PLP            STATUS RETURNED
02B2  F0 02             BEQ  SPEI      SKIP
02B4  29 03             AND  #$03      IF REQUIRED , MASK 2 BITS
02B6  9D 60 A4  SPEI    STA  IBUFM,X   PRINT STORAGE
02B9  8A                TXA            IF X IS...
02BA  2C 97 F0          BIT  #$01      EVEN OR ODD DIRECT
                                       OPERAND
02BD  D0 08             BNE  ZUR       IF ODD
02BF  BD 60 A4          LDA  IBUFM,X
02C2  69 05             ADC  #$05      ADD TO DOT POSITION
02C4  9D 60 A4          STA  IBUFM,X
02C7  60        ZUR     RTS
```

**test program:**

```
A9  00              LDA  #$00      STARTING VALUE
85  00              STA  $00       COUNTER
20  96  02   T1     JSR  VALDOT    COMPUTE
20  00  02          JSR  AIGRA     PRINT
E6  00              INC  $00       COUNTER
A5  00              LDA  $00       COUNTER
C9  64              CMP  #$64      ALREADY 10
D0  F2              BNE  T1        NO
00                  BRK            END
```

The test program plots ascending measurement values from 0-99 (dec.), which are passed on to the accumulator.

## AIMGRAPH — GRAPHICS CAPABILITY FOR THE AIM PRINTER

This program lends 63 graphics characters to the AIM printer. You may even create other character fonts like Arabic or Chinese by only altering the contents of the table.

By studying the AIM MONITOR PROGRAM LISTING, it can be seen that the ROM starting with cell F2E1 is also a character generator ROM. The dot matrix is contained in 5 table sections for the columns. Here the table is controlled with the hexadecimal value of the symbol to be printed as the index. This is again almost a classical solution of how one can replace hardware by software. Our program pursues this line further and dupes the program run at the point at which the monitor comes back from the subprogram INCP. The pointer built up in $A47D and $A47E for the dot pattern to be used is manipulated to the appropriate location of our table, which starts from 0300.

By means of this method, it is obvious that any other desired symbol sets can be generated, even multiple sets in direct access. The author does not have sufficient time to play with these possibilities, and for this reason the standard graphic printout of a beautiful girl is missing. Readers will certainly take care of that promptly and exert themselves to bring games such as LIFE onto the printer.

AIMGRAPH can rely on an almost identical subroutine AIGRA such as the program AIMPLOT in this issue. Only the command for line counting is changed as follows:

```
0226  C9  5A               CMP  #$5A      FOR 90 DOTS
```

The subprogram NIPSU called up is to be replaced by the following NIPSU2. Whoever wants to operate AIMPLOT and AIMGRAPH simultaneously can query a software switch in AIGRA before the dot counting and correspondingly also in the subprograms NIPSU/NIPSU2, which are very similar to each other.

```
0266  A2  00        NIPSU2  LDX  #$00       CORRESPONDS APPROXI-
                                            MATELY TO IPSU IN $F0E3
0268  20  21  F1            JSR  INCP
026B  BD  60  A4    NIPS1   LDA  IBUFM,X
026E  29  3F                AND  #$3F       CLIP AS ADDRESSER
0270  A8                    TAY
0271  18                    CLC             ADDITION PREPARATION
0272  A9  1F                LDA  #$1F       CONVERSION TO NEW
                                            TABLE BASIS
0274  6D  7D  A4            ADC  JUMP       ADDRESS COMPUTED BY
                                            INCP
0277  85  00                STA  PNTL       MAKE $00/01 THE TABLE
                                            POINTER
0279  A9  10                LDA  #$10       DITTO FOR HIGH ADDRESS
027B  6D  7E  A4            ADC  JUM+1
027E  85  01                STA  PNTL+1
0280  B1  00                LDA  (PNTL),Y   HOLE DOT PATTERN FROM
                                            TABLE
0282  2C  7C  A4            BIT  IMASK      DOT SET
0285  F0  16                BEQ  NIPS2      ...AS IN SECTION IPSU
0287  AD  7A  A4            LDA  IBITL
028A  F0  08                BEQ  NIPS3
028C  0D  78  A4            ORA  IOUTL
028F  8D  78  A4            STA  IOUTL
0292  D0  09                BNE  NIPS2
0294  AD  7B  A4    NIPS3   LDA  IBITU
0297  0D  79  A4            ORA  IOUTU
029A  8D  79  A4            STA  IOUTU
029D  0E  7A  A4    NIPS2   ASL  IBITL
02A0  2B  7B  A4            ROL  IBITU
02A3  CA  CA                DEX, DEX
02A5  10  C4                BPL  NIPS1
02A7  4C  18  F1            JMP  $F118      TO THE REMAINDER OF
                                            ROUTINE IPSU
```

```
<M>=0300  0080 C0E0 F0F8 FC40 0C20 1010 1008 04FE   CHARACTER
< > 0310  18AA 02C6 1C00 1010 000E 1EFE 0280 1882   GENERATOR
< > 0320  0010 0004 F400 1010 0010 2810 0006 1C80   TABLE
< > 0330  1CFE FEFE FEFE 0000 0000 0EFE FE00 020E   FOR A
< > 0340  0080 C0E0 F0F8 FC40 1C20 1038 1008 0482   GRAPHICS
< > 0350  8C54 02AA 8800 1010 000E 1E80 0C80 04CC    FONT
< > 0360  0010 1CFC C000 2008 0010 1010 000E 3C60
< > 0370  4400 FEFE FEFE FE00 0000 0E3E 0200 020E   BUILT UP AND
< > 0380  C080 C0E0 F0F8 FC40 3820 10FE 1E08 0482   IN SUCCESSION
< > 0390  FEAA 0292 FE1E 1EF0 F00E 1E80 0180 FE01   AS TABLES
< > 03A0  00F0 0004 E806 C006 FEFE 7CFE 001E EE01   COL0 THRU COL4
< > 03B0  8200 00FE FEFE 00FE 0000 FE1E 02FE 02FE   MONITOR
< > 03C0  2080 C0E0 F0F8 FC40 1C20 1038 1008 0482   PROGRAM
< > 03D0  8C54 0282 8810 0000 100E 1E80 6080 04CC
< > 03E0  0010 70FC C008 0000 1000 1010 FE3E 3C0C    INVERSE
< > 03F0  4400 0000 FEFE 0000 FE00 E00E 02FE 02E0   REPRESENTATION
< > 0400  0080 C0E0 F0F8 FC40 0C20 1010 1008 04FE   POSSIBLE BY
< > 0410  18AA 0282 1C10 0000 100E 1E80 80FE 1882   EXOR-ING
< > 0420  0010 0004 F410 0000 1000 2810 FEFE 1C02   TABLE CONTENTS
< > 0430  1C00 0000 00FE 0000 00FE E006 02FE FEE0
```

As can be seen from the instruction in $026B, the program provides the information in the printer buffer starting with $A460 with a graphic meaning. It is not at all difficult to bring this information by program to that location. But the question has still not been answered as to how one goes from EDITOR directly and interactively by means of a USER OUTPUT FUNCTION to the graphic printout of the open text line. To this end suggestions are welcome.

To test out AIMGRAPH, there is the following program for printing out the first 20 ASCII symbols ($20-$33 corresponding to a gap up to 3). By changing the initial value in the accumulator, one is able to print out the entire symbol set.

```
0500     A2     LDX     #00       ADDRESSER
0502     A9     LDA     #20       ASCII = BLANK (SPACE)
0504     9D     STA     A460,X    IBUFM,X
0507     38     SEC
0508     69     ADC     #00       ADD X]
050A     E8     INX
050B     E0     CPX     #14       20 CHARACTERS
050D     D0     BNE     0504
050F     20     JSR     0200      PRINT
0512     00     BRK               BACK TO MONITOR
```

—o—

## INSIDE BASIC

**Jim Buterfield**
**Toronto**

(This article is being reprinted with permission from the publisher of TARGET, a newsletter dedicated soley to the AIM 65. Lets thank Jim Butterfield for providing the world with so much information on AIM 65 Basic! More information on Target can be gotton by writing c/o Donald Clem, RR #2, Conant Rd., Spencerville, Ohio 45887)

### Basic Token List

| Token | Operation | Address |
|---|---|---|
| 80 | END | B65E |
| 81 | FOR | B55C |
| 82 | NEXT | BB00 |
| 83 | DATA | B767 |
| 84 | INPUT | B9BC |
| 85 | DIM | BDDA |
| 86 | READ | B9F0 |
| 87 | LET | B814 |
| 88 | GOTO | B714 |
| 89 | RUN | B6EC |
| 8A | IF | B797 |
| 8B | RESTORE | B631 |
| 8C | GOSUB | B6F7 |
| 8D | RETURN | B741 |
| 8E | REM | B7AA |
| 8F | STOP | B65C |
| 90 | ON | B7BA |
| 91 | NULL | BF87 |
| 92 | WAIT | C56C |
| 93 | LOAD | E848 |
| 94 | SAVE | B69F |
| 95 | DEF | C0F1 |
| 96 | POKE | C563 |
| 97 | PRINT | B8A9 |
| 98 | CONT | B685 |
| 99 | LIST | B4BC |
| 9A | CLEAR | B481 |
| 9B | GET | B9AD |
| 9C | NEW | B465 |
| AE | SGN | C978 |
| AF | INT | CA0B |
| B0 | ABS | C997 |
| B1 | USR | 0003 |
| B2 | FRE | C0BD |
| B3 | POS | CODE |
| B4 | SQR | CC75 |
| B5 | RND | CD96 |
| B6 | LOG | C729 |
| B7 | EXP | CCF1 |
| B8 | COS | CDD2 |
| B9 | SIN | CDD9 |
| BA | TAN | CE22 |
| BB | ATN | 00BB |
| BC | PEEK | C54C |
| BD | LEN | C4BA |
| BE | STR$ | C1A3 |
| BF | VAL | C4EB |
| C0 | ASC | C4C9 |
| C1 | CHR$ | C42A |
| C2 | LEFT$ | C43E |
| C3 | RIGHT$ | C46A |
| C4 | MID$ | C475 |

### Dyadic Operation

| | |
|---|---|
| addition | C5A9 |
| subtraction | C592 |
| multiplication | C76A |
| division | C851 |
| exponentiation | CC7F |
| logical AND | BD42 |
| logical OR | BD3F |
| negation | CCB8 |
| logical NOT | BC9C |
| comparison | BD6F |

### Zero Page Usage

AIM BASIC V1.1 -

| | | |
|---|---|---|
| 0000-0002 | 0-2 | New-line jump |
| 0003-0005 | 3-5 | USR jump |
| 0006 | 6 | Search character |
| 0007 | 7 | Scan-between-quotes flag |
| 0008 | 8 | Input buffer pointer; # subscripts |
| 0009 | 9 | Default DIM flag |
| 000A | 10 | Type: FF = string, 00 = numeric |
| 000B | 11 | Type: 80 = integer, 00 = floating point |
| 000C | 12 | DATA scan flag; LIST quote flag; memory flag |
| 000D | 13 | Subscript flag; FNx flag |
| 000E | 14 | 0 = input; $40 = get; $98 = read |
| 000F | 15 | Comparison evaluation flag |
| 0010 | 16 | Input flag: suppress output if negative |
| 0011 | 17 | I/O for prompt suppress |
| 0012 | 18 | Width |
| 0013 | 19 | Input column limit |
| 0014-0015 | 20-21 | Integer address (for GOTO, etc.) |
| 0016-005D | 22-93 | Input buffer |
| 005E | 94 | Temporary string descriptor stack pointer |
| 005F-0060 | 95-96 | Last temporary string pointer |
| 0061-0069 | 97-105 | Stack of descriptors for temporary strings |
| 006A-006B | 106-107 | Pointer for number transfer |
| 006C-006D | 108-109 | Misc. number pointer |
| 006E-0072 | 110-114 | Product staging area for multiplication |
| 0073-0074 | 115-116 | Pointer: Start-of-Basic memory |
| 0075-0076 | 117-118 | Pointer: End-of-Basic, Start-of-Variables |
| 0077-0078 | 119-120 | Pointer: End-of-Variables, Start-of-Arrays |
| 0079-007A | 121-122 | Pointer: End-of-Arrays |
| 007B-007C | 123-124 | Pointer: Bottom-of-strings (moving down) |
| 007D-007E | 125-126 | Utility string pointer |
| 007F-0080 | 127-128 | Pointer: Limit of Basic Memory |
| 0081-0082 | 129-130 | Current Basic line number |
| 0083-0084 | 131-132 | Previous Basic line number |
| 0085-0086 | 133-134 | Pointer to Basic statement (for CONT) |
| 0087-0088 | 135-136 | Line number, current DATA line |
| 0089-008A | 137-138 | Pointer to current DATA item in memory |
| 008B-008C | 139-140 | Input vector |
| 008D-008E | 141-142 | Current variable name |
| 008F-0090 | 143-144 | Current variable memory address |
| 0091-0092 | 145-146 | Variable pointer for FOR/NEXT |
| 0093-0094 | 147-148 | Y-save; new-operator save; utility pointer |
| 0095 | 149 | Comparison symbol accumulator |
| 0096-0097 | 150-151 | Misc numeric work area |
| 0098-009B | 152-155 | Work area; garbage yardstick |
| 009C-009E | 156-158 | Jump vector for functions |
| 009F-00A8 | 159-168 | Misc numeric work and storage areas |
| 00A9-00AE | 169-174 | Accumulator No. 1: Exponent, 4 Mantissa, Sign |
| 00AF | 175 | Series evaluation constant pointer |
| 00B0 | 176 | Acc No. 1 high-order (overflow) word |
| 00B1-00B6 | 177-182 | Accumulator No. 2: E,M,M,M,M,S |
| 00B7 | 183 | Sign comparison, Accumulators No. 1 vs No. 2 |
| 00B8 | 184 | Acc No. 1 low-order (rounding) word |
| 00B9-00BA | 185-186 | Series pointer |
| 00BB-00BD | 187-189 | Error jump |
| 00BF-00D6 | 191-214 | Subroutine: Get Basic char; C6, C7 = Basic pointer |

## Basic Entry Points

(Note: addresses indicate where a routine is: the first address is not always the entry point.)

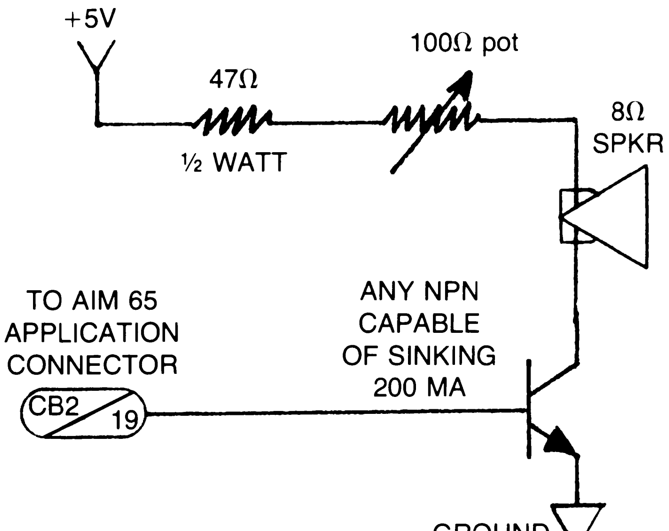
| | |
|---|---|
| B000-B002 | Cold start jump |
| B003-B005 | Warm start jump |
| B006-B009 | Vectors to subroutines; Floating to fixed, fixed to fl. |
| B00A-B043 | Action addresses for primery keywords |
| B044-B071 | Action addresses for functions |
| B072-B08F | Hierarchy and action addresses for operators |
| B090-B174 | Table of Basic keywords |
| B175-B1AB | Basic messages, mostly error messages |
| B1AC-B1D9 | Search stack for FOR or GOSUB activity |
| B1DA-B21C | Open up space in memory |
| B21D-B229 | Test: stack too deep? |
| B22A-B256 | Check available memory |
| B257-B27E | Send canned error message, then: |
| B27F-B29C | Warm start; wait for command |
| B29D-B328 | Handle new Basic line from keyboard or device |
| B329-B355 | Rebuild chaining of Basic lines in memory |
| B356-B3AD | Receive line from keyboard |
| B3AE-B435 | Change keywords to Basic tokens |
| B436-B464 | Search Basic for a given Basic line number |
| B465 | Perform NEW, then: |
| B481-B4AD | Perform CLEAR |
| B4AE-B4BB | Reset Basic execution to start-of-program |
| B4BC-B55B | Perform LIST |
| B55C-B600 | Perform FOR |
| B601-B630 | Execute Basic statement |
| B631-B63F | Perform RESTORE |
| B640-B65B | Check F1 key, and if down: |
| B65C-B684 | Perform STOP or END |
| B685-B69E | Perform CONT |
| B69F-B6AA | Perform SAVE |
| B6AB-B6B8 | Get input character |
| B6B9-B6D7 | Send formatted character to output |
| B6D8-B6E2 | Check if I/O device is Cassette, TTY, or User |
| B6E3-B6EB | Test if any key depressed |
| B6EC-B6F6 | Perform RUN |
| B6F7-B713 | Perform GOSUB |
| B714-B740 | Perform GOTO |
| B741-B766 | Perform RETURN, and then: |
| B767-B774 | Perform DATA, i.e., skip rest of statement |
| B775 | Scan for next Basic statement |
| B778-B796 | Scan for next Basic line |
| B797 | Perform IF, and perhaps: |
| B7AA-B7B9 | Perform REM, i.e., skip rest of line |
| B7BA-B7D9 | Perform ON |
| B7DA-B813 | Get fixed-point number from Basic line |
| B814-B89C | Perform LET |
| B89D-B8A8 | Enable printer on "!" character |
| B8A9-B949 | Perform PRINT |
| B94A-B966 | Print string from memory |
| B967-B987 | Print single format character (space, question mark) |
| B988-B9AC | Handle bad input data |
| B9AD-B9BB | Perform GET |
| B9BC-B9E6 | Perform INPUT |
| B9E7-B9EF | Prompt and receive input |
| B9F0-BADB | Perform READ; common routines used by INPUT and GET |
| BADC-BAFF | Messages: EXTRA IGNORED, REDO FROM START |
| BB00-BB58 | Perform NEXT |
| BB59-BB7E | Check data type, print TYPE MISMATCH |
| BB7F | Input and evaluate any expression (numeric or string) |
| BCB9 | Evaluate expression within parentheses () |
| BCBF | Check right parenthesis ) |
| BCC2 | Check left parenthesis ( |
| BCC5-BCCF | Check for comma |
| BCD0-BCD4 | Print SN (syntax) and exit |
| BCD5-BCDB | Set up function for future evaluation |
| BCDC-BCFF | Set up variable name |
| BD00-BD3E | Identify and set up function references |
| BD3F | Perform OR |
| BD42-BD6E | Perform AND |
| BD6F-BDD9 | Perform comparisons, string or numeric |
| BDDA-BDE3 | Perform DIM |
| BDE4-BE6D | Search for variable location in memory |
| BE6E-BE77 | Check if ASCII character is alphabetic |
| BE78-BEDB | Create new Basic variable |
| BEDC-BEEC | Array pointer subroutine |
| BEED-BEF0 | 32768 in floating binary |
| BEF1-BF0F | Evaluate expression for positive integer |
| BF10-C08B | Find or create array |
| C08C-C0BC | Compute array subscript size |
| C0BD | Perform FRE, including: |
| C0D1-C0DD | Convert fixed-point to floating-point |
| C0DE-C0E3 | Perform POS |
| C0E4-C0F0 | Check if direct command, print ILLEGAL DIRECT |
| C0F1-C11E | Perform DEF |
| C11F-C131 | Check FNx syntax |
| C132-C1A2 | Evaluate FNx |
| C1A3-C1B2 | Perform STR |
| C1B3-C1C4 | Calculate string vector |
| C1C5-C231 | Scan and set up string |
| C232-C263 | Subroutine to build string vector |
| C264-C2FA | Garbage collection subroutine |
| C2FB-C343 | Check for most eligible string for collection |
| C344-C37A | Collect a string |
| C37B-C3B7 | Perform string concatenation |
| C3B8-C3E0 | Build string into memory |
| C3E1-C418 | Discard unwanted string |
| C419-C429 | Clean the descriptor stack |
| C42A-C43D | Perform CHR$ |
| C43E-C469 | Perform LEFT$ |
| C46A-C474 | Perform RIGHT$ |
| C475-C49E | Perform MID$ |
| C49F-C4B9 | Pull string function parameters from stack |
| C4BA-C4BF | Perform LEN |
| C4C0-C4C8 | Move from string-mode to numeric-mode (LEN, ASC, VAL) |
| C4C9-C4D8 | Perform ASC |
| C4D9-C4EA | Input byte parameter |
| C4EB-C529 | Perform VAL |
| C52A-C535 | Get two parameters for POKE or WAIT |
| C536-C54B | Convert floating-point to fixed-point |
| C54C-C562 | Perform PEEK |
| C563-C56B | Perform POKE |
| C56C-C587 | Perform WAIT |
| C588-C58E | Add 0.5 to Accumulator No. 1 |
| C58F-C5A5 | Perform subtraction |
| C5A6-C685 | Perform addition |
| C686-C6BC | Complement Accumulator No. 1 |
| C6BD-C6C1 | Print OV (overflow) and exit |
| C6C2-C6FA | Multiply-a-byte subroutine |
| C6FB-C728 | Function constants: 1, SQR(.5), SQR(2), -0.5, etc. |
| C729 | Perform LOG |
| C76A-C797 | Perform multiplication |
| C798-C7CA | Multiply-a-bit subroutine |
| C7CB-C7F5 | Load Accumulator No. 2 from memory |
| C7F6-C812 | Test and adjust Accumulators No. 1 and No. 2 |
| C813-C820 | Handle overflow and underflow |
| C821-C837 | Multiply by 10 |
| C838-C83C | 10 in floating binary |
| C83D | Divide by 10 |
| C846 | Perform divide-by |
| C851-C8E0 | Perform divide-into |
| C8E1-C905 | Load Accumulator No. 1 from memory |
| C906-C93A | Store Accumulator No. 1 into memory |
| C93B-C94A | Copy Accumulator No. 2 into Accumulator No. 1 |
| C94B-C959 | Copy Accumulator No. 1 into Accumulator No. 2 |
| C95A-C969 | Round off Accumulator No. 1 |
| C96A-C977 | Compute SGN value of accumulator No. 1 |

(continued on next page)

## AIM-65 SOUND

Wouldn't it be nice if your computer had a means of letting you know when it needed some attention?

Well, now it can do just that with the addition of a speaker and some additional parts. No, the idea isn't new — just an adaption from the PET since it also has a 6522 VIA chip installed. And because this interface uses the CB2 line, you don't really lose too much of the system's I/O capability.


+5V
47Ω
½ WATT
100Ω pot
8Ω SPKR
TO AIM 65 APPLICATION CONNECTOR
CB2
19
ANY NPN CAPABLE OF SINKING 200 MA
GROUND

(continued from previous page)

| | |
|---|---|
| C978-C996 | Perform SGN |
| C997-C999 | Perform ABS |
| C99A-C9D9 | Compare Accumulator No. 1 to memory |
| C9DA-CA0A | Convert floating-point to fixed-point |
| CA0B-CA31 | Perform INT |
| CA32-CABC | Convert string to floating-point |
| CABD-CAF1 | Get new ASCII digit |
| CAF2-CB00 | String conversion constants: 99999999,999999999,1E+9 |
| CB01 | Print IN, followed by: |
| CB0C-CB1B | Pring Basic line number |
| CB1C-CC4B | Convert floating-point number to ASCII |
| CC4C-CC74 | Constants for numeric conversion |
| CC75 | Perform SQR |
| CC7F | Perform power function |
| CCB8-CCC2 | Perform negation |
| CCC3-CCF0 | Constants for string evaluation |
| CCF1-CD43 | Perform EXP |
| CD44-CD8D | Function series evaluation subroutines |
| CD8E-CD95 | Manipulation constants for RND |
| CD96-CDD1 | Perform RND |
| CDD2 | Perform COS |
| CDD9-CE21 | Perform SIN |
| CE22-CE4D | Perform TAN |
| CE4E-CE85 | Constants for trig: pi/2, 2*pi, .25, etc. |
| CE86-CE9D | Character subroutine, to be copied to BF to D6 |
| CE9E-CEA2 | Initialization constants |
| CEA3-CFAE | Cold start: initialize Basic, prompt, etc. |
| CFAF-CFF9 | Startup messages and prompts |
| CFFA-CFFF | Patch |

This particular circuit as well as the software presented was found in the Rockwell Hobby Club newsletter but has appeared in numerous other publications. Actually, if you're on the lazy side, you can use the battery operated speaker/amplifier from Radio Shack (about $10.95) and save yourself the trauma of building something.

The neatest thing about this method of sound generation is that once the 6522 is properly initialized, the CPU can go off and perform other tasks. NO FURTHER PROCESSOR INTERVENTION IS REQUIRED!

This is because the shift register in the VIA can be set to operate in the "free running" mode. In this mode, whatever data that is loaded into the shift register, will be continuously shifted out to the CB pin on the 6522.

Hook up the transistor amplifier (or the Radio Shack speaker/ amplifier) to AIM 65 and load in the two example sound programs or just fool around with three POKE locations in the 6522.

POKE 40971,16 (ACR) sets the 6522 chip to a "free-running" state with the shifting rate determined by T2 timer.

POKE 40970,51 (SR) loads the shift register with a "constant" that will be continuously shifted out on CB2.

POKE 40968,N (T2L) where N is a number from 1 to 255 that determines the frequency of the note by setting the time out period for T2.

Here are values for musical note equivalents. (Assuming a '51" was poked into 40970.)

HERE IS HOW TO MAKE MUSIC:
Use a subroutine for your musical sound effects. Start with

```
2000  POKE 40971,16
2010  POKE 40970,10: REM THIS IS FOR TONE--FROM 1 TO 255-VE RY MELLOW
      TO VERY SHARP.
2020  POKE 40968,115: REM THIS IS PITCH. FROM 1 TO 255-HIGH TO LOW.
2030  POKE 40971,0: REM THIS TURNS SOUND OFF.
2040  RETURN
```

To play continuously, eliminate line 2030.

Here's another one:

```
3000  POKE 40971,16
3010  POKE 40970,10
3020  FOR P = 1 TO 255
3030  POKE 40968,P
3040  NEXT P
3050  POKE 40971,0
3060  RETURN
```

Now you can start experimenting on your own with various sound effects.

You folks without BASIC should take this opportunity to convert these routines to machine language. The only possible problem area will be in the time delay loop in line 3020. You'll get the feel for how slow BASIC is when compared to machine code.

# PRODUCT SURVEY

## LET'S CLOSE THE LOOP

As a semiconductor manufacturer, we NEED your inputs. You are the marketplace, and should be the determining factor in the kinds of products we produce. If you have any ideas for things that would be useful either on a system level (modules, single-board computers, etc) or, at the component level (peripheral devices, CPUs, interface chips and the like), LET US KNOW!!!!!! Here are some questions to get you started. Please feel free to write a 10-page essay, if that's what it takes.

## SYSTEM LEVEL STUFF

As you know, we are second-sourcing the Motorola 68000 CPU. Since we may be building some sort of single-board computer with this device, it would be very helpful to know what kinds of features you would desire in such a product.

First, let's discuss a little background on the 68000 chip so you have an idea of it's place in the computing world. The 68000 is an advanced 16-bit processor with a direct addressing capability of 16 Megabytes (up to 64 Megabytes with some simple bank select logic). Actually the internal architecture of the machine works on 32-bit data but is externally limited to 16 bits because of present packaging constraints. This machine has been favorably compared with the PDP 11/34 and is really a minicomputer CPU rather than a microprocessor. Systems design will be much more complicated with the 68000 than with the 6502, for example, due to it's minicomputer-like design. You probably won't see the 68000 used in small, dedicated controller applications because of this complexity. However, for high-end microprocessor and traditional minicomputer applications, the 68000 will really shine. In fact, a network of 68000s in a multiprocessor configuration could probably move into the mainframe area of ability.

A person looking through the 68000 documentation will probably wonder why there are no op-code tables published. One reason is that by combining the 68000's 56 basic instructions, variations on these instructions and 14 addressing modes, you can come up with over 1000 instruction combinations! Another reason is that hand-assembly is next to impossible, and Motorola assumes that every serious user will be using at least an assembler to program the beast and more likely a high-level language, since that's what the machine was designed for anyway. (After attempting to hand assemble a rather short 68000 program, I fully concur with Motorola).

Now that you've had a chance to see the 68000, (at least through my eyes), you can start thinking about what kinds of things you'd like in a single-board computer designed around the 68000.

## QUESTION 1

What sort of I/O device would you desire on a 68000 single board computer? In addition to an ASCII keyboard, you have a choice between a 40 column printer/display or an interface for a user- supplied CRT and printer. Keep in mind that an on-board 40 column printer display would probably raise the price of the board between $150 and $200 so if you'd be primarily using your own CRT and printer, the increased cost of the on-board I/O would be wasted.

## QUESTION 2

Which two of the following high-level languages would you like to see available for the 68000 single-board computer: Basic, Pascal, Forth, Fortran, APL, LISP, or Cobol?

**DETACH THIS SECTION AND RETURN IT WITH YOUR COMMENTS**

## QUESTION 3

What kinds of I/O capability would be necessary for the 68000 board to meet your needs? IEEE 488? Several RS232 channels? Cassette? Floppy? Video? What? Again, keep in mind that even though we'd like to have everything, the cost will go up needlessly with things we don't really need.

## QUESTION 4

What kinds of features would you like that aren't normally included in a single-board computer?

## QUESTION 5

How much memory should be included on the main board How much ROM/PROM space? How much RAM? In the 68000, the lowest 1K bytes are dedicated to ''exception'' vectors, trap, interrupt, reset and error vectors, so we must start with that much as a base minimum.

**TAKE A FEW MINUTES**

## QUESTION 5A

For what applications would you consider using a 16-bit processor? (68000 or other machine)

## QUESTION 6

Now for some 6500-type stuff:
Assuming we were going to be designing another single-board computer based on the 6052, sort of an advanced AIM 65 type system, what would you like to see? Should an on-board printer/display be provided? Or would you rather see an I/O-independent system that could utilize an external CRT and printer? Remember the cost factor.

## QUESTION 7

Would you insist on a floppy interface, or would cassette storage be sufficient for your application? You'd be paying about $60 more for each board if the floppy interface were included.

## QUESTION 8

What types of expansion modules do you have a need for in your application? RS232, IEEE, I/O etc.

**TELL US WHAT YOU THINK**

## QUESTION 9

What would you be using an advanced 6502 system for? OEM? Software development, Hardware, development, Self-teaching, hobbyist, engineering application, or what?

## QUESTION 10

What do you feel is the minimum usable display/printer size that is practical for a low-cost development system -20, 40, 60, 80 or 120 columns?

**First Class**
Permit No. 101
El Segundo
California

**Business Reply Mail**
No Postage Stamp Necessary if Mailed in the United States

Postage will be paid by

**ROCKWELL INTERNATIONAL**
**MICROELECTRONIC DEVICES**
P.O. Box 3669
Anaheim, CA. 92803, U.S.A.

**ATTN: MARKETING SERVICES**
**D/727 RC55**

**PRODUCT SURVEY**

STAPLE HERE

## DISKS FOR THE AIM 65

Five companies have announced disk systems for AIM 65. These companies are:

**HDE Inc**
**POB 120**
**Allamuchy, NJ 07820**
**(201) 362-6574**

**COMPAS MICROSYSTEMS**
**224 S.E. 16 th St.**
**Ames, Iowa 50010**
**(515) 232-8187**

**Micro Technology Unlimited**
**POB 12106**
**Raleigh, N.C. 27605**
**(919) 833-1458**

**RNB Enterprises**
**2967 W. Fairmount Ave.**
**Phoenix, Arizona 85017**
**(602) 265-7564**

**Applied Business Computer**
**Suite G**
**707 S. State College Blvd.**
**Fullerton, CA 92631**
**(714) 871-1411**

Here are the features for each:

### HDE OMNI-65 SYSTEM

*uses the KIM-4, 44-pin expansion arrangement w/4.5''x6.0'' card

*two systems are available-a single-density/single-sided 5'' drive system (up to two drives) and a single-density/single-sided 8'' system (up to two drives)

*system is disk-based and the bootstrap program must be loaded in from cassette

*this system has the ability to save and load Basic data files (as well as program files), programs can be appended or chained from disk, disk accesses may be accomplished under Basic program control, and machine language routines can be automatically called in from disk when needing to link up with Basic through the USR function.

*able to assemble from disk only. Object code must be saved to disk manually. Can link multiple source files together from disk with special assembler directives

*schematic included in documentation

*source listing of system not available

*controller board, power supply, cables, and a single-density/single-sided mini floppy drive sell for around $800 in the U.S.

### COMPAS "DAIM" SYSTEM

*disk file compatability with the Rockwell System 65

*uses the AIM 65/SYSTEM 65 expansion motherboard

*can interface with up to two single-density/single-sided mini-floppy drives

*schematic is included

*assembly listing of system available on disk for $10.

*interfaces with the on-board AIM 65 Assembler and Basic ROM options to enable the saving and loading of source and object files (although the DAIM cannot link assembler files together from disk, COMPAS has an optional disk-based assembler ($95) that will do the job).

*able to assemble to and from disk (only one output file may be open on a single drive at one time)

*disk software is on ROM.

*controller board, power supply, single drive and cables sell for around $850 in the U.S.

### RNB VAK-7 SYSTEM FEATURES

*uses the KIM-4, 44-pin expansion arrangement w/7''x10'' card

*available only as full-size 8'' drive system with double- density capability included and double-sided drive an option.

*ROM software includes the ability to assemble from disk, and save and load Basic programs to and from disk

*drive cabinet is included

*uses DMA approach with 1K shared RAM.

*up to four double-density/double-sided drives can be handled by the controller.

*source listing not available but all routine entry points are included in documentation.

*schematic included.

*controller board, cabinet w/one 8'' double-density drive, power supply, and cable sells for around $1300 in the U.S.

### APPLIED BUSINESS COMPUTER FP-950 SYSTEM FEATURES

*uses the AIM 65/SYSTEM 65 expansion mothercoard

*can interface with up to four double-sided/double-density mini-floppy or full-size drives

*ability to save and load Basic programs to and from disk

*can assemble program to and from disk

*includes information on accessing the disk from user program control

*able to execute programs directly from disk

*has an on-board Centronics compatible printer port and printer

*schematic not available

*disk software is ROM-resident

*source listing not available (company does provide some routine entry points).

*controller board, power supply, cable, and one double-sided/double-density mini-floppy drive sells for around $850 in the U.S.

### MICRO TECHNOLOGY UNLIMITED "APEX 65" FEATURES

*uses the AIM 65 expansion bus pinout which is compatible with their own card cage.

*the controller will handle up to four Shugart compatible, 8'' double-density/double-sided drives.

*will save and load object code, Basic programs and Assembler source code.

*system is disk-based with bootstrap on ROM

*DMA type with 16K shared memory

*controller card sells for around $600 in the U.S. The user must provide the power supply, the drives, and cables.

Check with each individual vendor to see if they're delivering systems and by all means ORDER THE DOCUMENTATION to see what it's like BEFORE you order the system.

If you have one of these systems, how about writing a product review for INTERACTIVE The other readers would enjoy reading about it.

## HOW TO USE THE SPECIAL FUNCTION KEYS

Your AIM 65 is equipped with six keys which can be used for going from the monitor to your programs with a minimum of keystrokes. The first three keys are called the 'FUNCTION KEYS' and are designated F1, F2, and F3 on the right hand side of the keyboard. The operation of these keys is covered pretty well in the AIM USER GUIDE section 3-47 of the Rev 3 edition (section 3-46 of Rev 2) so I won't go into too much detail here except to point out one thing. The function keys are intended to be used in calling user-written monitor extensions. The monitor treats these functions as SUBROUTINES so an RTS is necessary at the end to allow returning to the monitor. If the keys are used to jump to a user routine which isn't meant to return to the AIM 65 then the stack will be left with some garbage on it. This garbage could fill up the whole stack if you get carried away with the function keys unless the stack is cleaned up with two PLA instructions when you enter your routine.

The three other keys (5,6 and N) would be of interest to those who are installing EPROMS in the Basic or Assembler sockets in AIM 65 and wanted to jump into them with one keystroke.

The most versatile entry is available with the Z26 ROM socket. Here you have two entry points available with one keystroke each. In the monitor mode, pressing the '5' key will transfer control to $B000. This would be the logical cold start entry point for the new software (an enhanced machine language monitor, for example). The '6' key jumps to location $B003 which could be the warm start entry point.

The 'N' key transfers control to $D000 which is the first address in the Z24 ROM socket. This key isn't as versatile as the '5' and '6' keys but can be still quite useful when non-technical persons may be operating the equipment. They can just be told to press the 'N' key after the machine is powered up instead of having to understand how to set the program counter and then start running at the address.

## WE'VE GOT OUR EARS ON

Leo Scanlon, Rockwell Documentation Manager, is eager to hear from anyone who feels he has found an error in, or has a suggestion for the AIM 65 documentation. When writing about a manual, please refer to the text by section num- ber (rather than page number) and the manual revision number.

Write to:
Documentation Manager
Rockwell International
Box 33093, RC 55
Anaheim, CA 92803

## DISASSEMBLER UTILITY

Unknown Author

(This handy little routine was submitted for publication and got inadvertently separated from the cover letter. If you know who wrote it (someone from France) please let me know so I can give the proper credits)

One thing missing on the AIM 65 is a provision for disassembling a single program line to the on-board display. If the printer is turned off, the instructions just whizz by much too quickly to read. Depressing the space bar, of course, causes the display to halt temporarily but getting good enough to halt things after just one line takes much skill.

Well, here's one solution to the problem. A short program that does the trick.

Start the program with the F3 key (assuming the proper jump location has been initialized) and the program operates much like the built-in disassembler from then on. Tape the space bar to advance to the next instruction.

```
OUTPUT=$E97A
ADDIN=$EAAE
CGPCO=$E5D7
CGPC1=$E5DD
REDOUT=$E973
READ=$E93C
CLR=$EB44
DISAS=$F46C
RCHEK=$E907
CRLF=$E9F0
*=$0112
JMP DEB
*=$EA
LENGHT *=*+1
*=$A425
SAVPC *=*+2
;
*=$0F90
.DEB LDA #$2A
JSR OUTPUT
JSR ADDIN  ;READ ADDRESS=4 DIGITS
BCS DEB
JSR CGPCO  ;PC=FIRST ADDRESS
.LECT JSR REDOUT
CMP #$20  ;SP?
BNE LECT
JSR CLR
JSR DISAS ;DISASSEMBLE ONE INSTRUCTION
LDA SAVPC
SEC
ADC LENGHT  ;ADJUST PC
STA SAVPC
BCC FIN
INC SAVPC+1
JSR RCHEK
JSR CRLF
FIN JMP LECT
.END
```

## CORRECTION FOR THE AIM 65 BASIC MANUAL

An important page was inadvertently left out of the early AIM 65 BASIC manual. This page had the information which enabled the ATN (arctangent) function to be added to BASIC. So here is that all important information.

The ATN function (see Subject 307) can be programmed in RAM using the AIM 65 Mnemonic Entry (1) and Alter Memory Locations (/) commands, as shown below. The program is written for the AIM 65 with 4K bytes of RAM. The ATN function can be relocated elsewhere in memory by changing the starting addresses of the instructions and constants, the conditional branch addresses, the vector to the constants start address and the vector to the ATN function start address.

### ATN FUNCTION CONSTANTS ENTERED BY ALTER MEMORY <M>

```
<M>   = 0F80   XX  XX  XX  XX     Constants Starting Address = 0F80₁₆
</>   = 0F80   0B  76  B3  83
</>     0F84   BD  D3  79  1E
</>     0F88   F4  A6  F5  7B
</>     0F8C   83  FC  B0  10
</>     0F90   7C  0C  1F  67
</>     0F94   CA  7C  DE  53
</>     0F98   CB  C1  7D  14
</>     0F9C   64  70  4C  7D
</>     0FA0   B7  EA  51  7A
</>     0FA4   7D  63  30  88
</>     0FA8   7E  7E  92  44
</>     0FAC   99  3A  7E  4C
</>     0FB0   CC  91  C7  7F
</>     0FB4   AA  AA  AA  13
</>     0FB8   81  00  00  00
</>     0FBC   00
```

### ATN FUNCTION INSTRUCTIONS STORED BY MNEMONIC ENTRY (1)

```
<1>
XXXX = 0FBD                               Instructions Starting Address =
                                          0FBD
0FBD    A5   LDA    AE
0FBF    48   PHA
0FC0    10   BPL    0FC5
0FC2    20   JSR    CCB8
0FC5    A5   LDA    A9
0FC7    48   PHA
0FC8    C9   CMP    #81
0FCA    90   BCC    0FD3
0FCC    A9   LDA    #FB
0FCE    A0   LDY    #C6
0FD0    20   JSR    C84E
0FD3    A9   LDA    #80                   Starting Address of Constants = 0F80
0FD5    A0   LDY    #0F
0FD7    20   JSR    CD44
0FDA    68   PLA
0FDB    C9   CMP    #81
0FDD    90   BCC    0FE6
```

(continued on next page)

(continued from previous page)

```
0FDF    A9   LDA    #4E
0FE1    A0   LDY    #CE
0FE3    20   JSR    C58F
0FE6    68   PLA
0FE7    10   BPL    0FEC
0FE9    4C   JMP    CCB8
0FEC    60   RTS
0FEC
```

## BASIC INITIALIZATION FOR ATN FUNCTION

BASIC memory must be initialized below the memory allocated to the ATN function. The ATN vector in RAM must also be changed from the address of the FC error message to the starting address of the ATN function instructions. This can be done using BASIC initialization, as follows:

```
<M>
MEMORY SIZE? 3968                     Limit BASIC to F80₁₆
WIDTH?
 3438 BYTES FREE
  AIM 65 BASIC V1.1
POKE 188, 189                         Change ATN function vector low to
                                      BD₁₆
POKE 189, 15                          Change ATN function vector high to
                                      0F₁₆
?ATN (TAN(.5))                        Test case to verify proper ATN func-
                                      tion program
.5                                    Expected answer = .5
```

## SAVING ATN OBJECT CODE ON CASSETTE

The object code for the ATN function can be saved on cassette by dumping addresses $00BB through $00BD (Jump instruction to ATN) and $0F80 through $0FEC (constants and instructions) after the function is initially loaded and verified.

The ATN function can then be loaded from cassette by executing the Monitor L command after BASIC has been initialized via the 5 command. After the ATN function has been loaded, reenter BASIC with the 6 command.

## ERROR!!! ERROR!!! ERROR!!!

There is a error in the JUMP INDIRECT instruction of ALL 6500 family CPU chips, no matter who they were made by. This fatal error occurs only when the low byte of the indirect pointer location happens to be $FF, as in JMP ($03FF). Normally, the processor should fetch the low-order address byte from location $03FF, increment the program counter to $0400 and then fetch the high-order address byte. Instead, the high-order byte of the program counter never gets increment ed and so the high-order address byte gets loaded from $0300 instead of $0400. For this reason, your program should NEVER include an instruction of the type JMP ($xxFF).

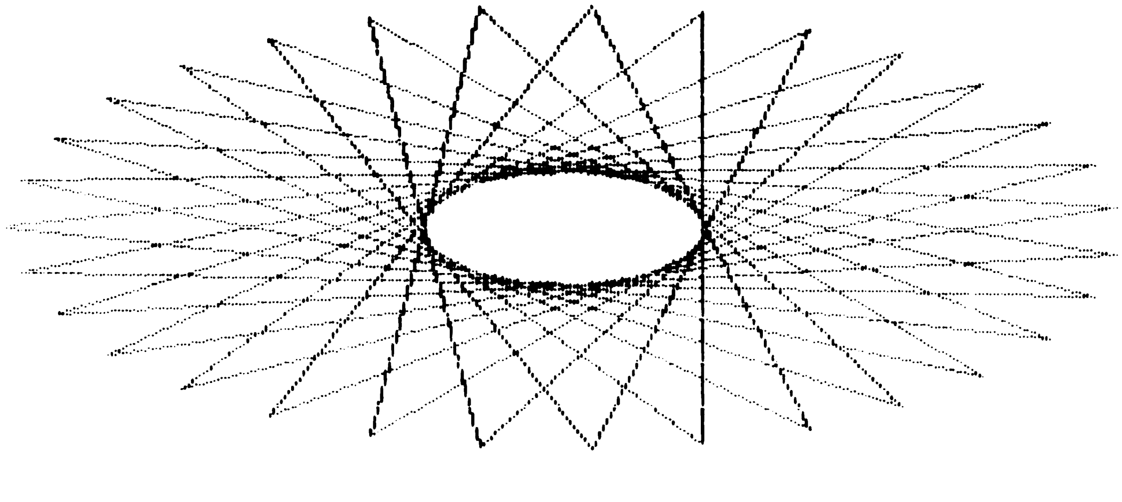
Try this example to satisfy yourself that you understand the problem: insert the following data into the AIM at the indicated memory locations.

```
0300 04
0310 6C FF 03
03FF 50 05
0450 00
0550 00
```

Execute the instruction at $0310. If the instruction worked correctly, the BRK at 0550 would have been encountered and the AIM display should be displaying 0551 xx. But, since the JMP indirect did not operate correctly, 0451 xx will be displayed since the high-order byte for the address was loaded from 0300 instead of 0400.

## CORRECTIONS CORNER

The biggest boo-boos in issue #1 were in the AIM 65 SPARE PARTS PROCUREMENT article. The proper phone number should be (714) 632-2190 for orders or inquiries. Two other major errors turned out to be that $2.00 handling fee is applicable to orders under $25.00 (not $10.00) and the reset switch really costs $2.37 (not .30). All this information is applicable only to U.S. orders.



# OFFSET LOADER FOR AIM 65

**Frank Reo**
**East Coast Tech Center**
**Rockwell International**

***(Editor's note: Since AIM 65 has no built-in capability for loading object code to a location different from where it was dumped, this program will be a godsend for some).***

## Purpose

There are many methods of using the AIM 65 to burn EPROM's. One such method is to transfer object code from the AIM 65 to the System 65 (for use by its PROM Programmer) via the TTY interface (Doc. No. R6500 N04). In order to perform this operation, it is required that object code be stored in AIM memory. In most cases (if not all cases) the object code will be assembled to operate from the address range B000, DFFF (AIM ROM sockets). If code assembled at those addresses is then loaded into the AIM, the data will go to ROM sockets and will not be stored in RAM. It now becomes desirable for a user to be able to dump object code during assembly and reload into RAM for transmission to the System 65 or simply for residence so that it can be used by any PROM burning device.

Notice that this Relocator, relocates code byte-for-byte such that the program being loaded may not necessarily execute at its relocated address.

## Description

Figure 1 is an AIM 65 disassembly of the Relocating loader program. This program is essentially a copy of the AIM monitor L-COMMAND (Pages 15 & 16, Doc. No. 29650 N36L). The first difference is in the beginning (addresses 0200 0214) where the operator defines the desired starting address of the object code. Those desired addresses are stored in locations $A41C and $A41D (ADDR & ADDR+1). The other difference is that when the absolute addresses of each block are read in they are not stored (022D & 0230).

Figure 1 shows the programs located at address $0200 thru $0265; however, the code is written such that it is relocatable. If these addresses are desired for use as storage, the program can be used to relocate itself in an area which will not be used for storage otherwise and it will execute anywhere in memory.

## Operation

This loader will work for both paper tape and audio cassette tape. Operating instructions for both modes appear below:

### Paper Tape

1. Start program = 0200
2. G.
3. TO = XXXX desired address always 4 digits
4. IN = L
5. Start paper tape reader on completion will apear in the AIM display.

### Audio Cassette Tape

1. Start Program = 0200
2. G.
3. TO = XXXX
4. IN = T FILE = (NAME) T = 1 (or 2)
5. Start tape (PLAY) on completion will appear in the AIM display.

```
0200  A0 LDY   05       ; point to MS5
0202  20 JSR   E7AF     ; disp "TO="
0205  A2 LDX   02
0207  20 JSR   E95F     ; get HI
020A  20 JSR   EA7D     ; Hex
020D  20 JSR   E95F     ; get next
0210  20 JSR   EA84     ; pack
0213  CA DEX
0214  AD STA   A41C,X   ; ADDR & ADDR+1
0217  D0 BNE   0207
0219  20 JSR   E9F0     ; crlf to display
021C  20 JSR   E848     ; where I, "IN="
021F  20 JSR   E993     ; get 1st char
0222  C9 CMP   3B       ; is it a ';'
0224  D0 BNE   021F     ; no
0226  20 JSR   EB4D     ; yes - clr chksum
0229  20 JSR   E54B     ; read record length
033C  AA TAX   ;        of bytes in X
022D  20 JSR   E54B     ; read address
0230  20 JSR   E54B     ; do not store!
0233  8A TXA            ; length to A
0234  F0 BEQ   0252     ; last
0236  20 JSR   E3FD     ; no - read data
0239  20 JSR   E413     ; store (ADDR, ADDR+1)
023C  CA DEX            ; update length
023D  D0 BNE   0236     ; done
023F  20 JSR   E3FD     ; yes - rd cksum
0242  CD CMP   A41F     ; OK
0245  D0 BNE   0263     ; no error
0247  20 JSR   E3FD     ; yes - rd cksum
024A  CD CMP   A41E     ; OK
024D  D0 BNE   0263     ; no
024F  F0 BEQ   021F     ; yes - get next record
0251  EA NOP
0252  A2 LDX   05       ; read 4 zeros
0254  20 JSR   E3FD
0257  CA DEX
0258  D0 BNE   0254
025A  20 JSR   E993     ; read last (CR)
025D  20 JSR   E520     ; set default
0260  4C JMP   E182     ; go to monitor
0263  20 JSR   E385     ; error
```

Figure 1

—o—

## FOR YOUR INFORMATION

Here's a list of all the companies that we know of who deal in accessories for the AIM 65. Rockwell makes no recommendations about these companies and only publishes this list to help our customers become aware of their existence.

### SUPPLIERS FOR AIM ACCESSORIES

ADVANCED COMPUTER PRODUCTS
1310 "B" E. Edinger
Santa Ana, CA 92705
(714) 558-8813

Power Supply
Case
ROMs, paper

APPLIED BUSINESS COMPUTERS
Suite G
707 S. State College Blvd.
Fullerton, CA 92631
(714) 871-1411

Floppy Disk System

BETA COMPUTER DEVICES
1230 W. Collins
Orange, CA 92668
(714) 633-7280

32K Dynamic RAM Board

COMPAS MICROSYSTEMS
P.O. Box 607
Ames, IA 50010
(515) 232-8187

5" Floppy Disk System
EPROM Programmer Card
RAM/EPROM Board
16K Static RAM
Assembler Software

COMPUTERIST, THE
56 Central Square
Chelmsford, MA 01824
(617) 256-3649

Card Cage/Motherboard
Memory Board
Video Board
Proto Board
Power Supply

CONDOR, INC.
4811 Calle Alto
Camarillo, CA 93010
(805) 484-2851

Power Supply

CUBIT
2267 Old Middlefield Way
Mountain View, CA 94043
(415) 962-8237

Motherboard
EPROM Programmer
8K Static RAM Board

ENCLOSURE GROUP
771 Bush St.
San Francisco, CA 94108
(415) 495-6925

Enclosures

EXCERT, INC.
P.O. Box 8600
White Bear Lake, MN 55110
(612) 426-4114

Custom AIM 65 Configurations

FORETHOUGHT PRODUCTS
87070 Dukhobar Rd.
Eugene, OR 97402
(503) 485-8575

Expansion Board Products

HDE, INC.
P.O. Box 120
Allamuchy, NJ 07820
(201) 362-6574

5" and 8" Floppy Disk Systems

8K Static RAM Boards
EPROM Board
Prototyping Card
Motherboard/Card Cage

MICROTECHNOLOGY UNLIMITED
POB 12106
Raleigh, NC 27605
(919) 833-1458

5" and 8" Floppy Disk Controller
16K Dynamic RAM Board
Dot Graphics Display Board
Card Cage/Motherboard
Prototyping Card
EPROM, I/O, EPROM Programmer Board
Graphics/Text Software Package
Power Supply
Music Board and Software

6502 PROGRAM EXCHANGE (DAVID MARSH)
2920 W. Moana Lane
Reno, NV 89509
(702) 825-8413

Microchess
Assorted Software

QUEST ELECTRONICS
2322 Walsh Avenue
Santa Clara, CA 95050
(408) 988-1640

Motherboard
Color Video Board
Parallel Board
32K Dynamic RAM Board
EPROM Programmer
Briefcase Enclosure
Power Supplies

REHNKE, ERIC C.
1067 Jadestone Lane
Corona, CA 91720

FORTH Programming Language
Math Package

RIVERSIDE ELECTRONICS
1700 Niagara St.
Buffalo, NY 14027
(716) 873-7317

Motherboard
Video Board
EPROM Programmer

CONNETICUT MICROCOMPUTER, INC.
150 Pocono Road
Brookfield, CT 06804
(203) 775-9659

A/D Modules

RNB ENTERPRISES
2967 Fairmount Ave.
Phoenix, AZ 85017
(602) 265-7564

8" Floppy Disk System
8K/16K Static RAM Boards
Motherboard/Card Cage
EPROM Programmer
EPROM Board
Prototyping Card
Extender Board
Power Supplies

SEAWELL MARKETING
P.O. Box 17170
Seattle, WA 98107
(206) 782-9480

Motherboard
16K Static
Parallel I/O

## PARITY BIT GENERATOR PROGRAM

**Mark Reardon**
**Rockwell International**

The AIM 65, and most other 6500-based systems, use a seven-bit ASCII character set, in which the high-order bit (Bit 7) is always a zero. It is possible to give this character odd parity or even parity by simply modifying this high-order bit.

The subroutine below takes an ASCII character in the Accumulator and modifies Bit 7 as appropriate to give it even parity. The same subroutine will generate odd parity if you change the LDX #08 instruction to LDX #09 and change the BPL AGAIN instruction to BNE AGAIN.

```
0000            :THIS PROCEDURE IS WRITTEN AS A
0000            :SUBROUTINE. IT USES THE X AND
0000            :A REGISTERS AND LOCATION $00.
0000            :
0000            TMP = $00
0000                    * = $200
0200  A2 08  PARITY  LDX #08          :INIT COUNTERS
0202  86 00          STX TMP
0204  CA             DEX
0205  6A     AGAIN   ROR A            :PUT 1 BIT IN C
0206  90 02          BCC NOPR         :COUNT 1'S ONLY
0208  E6 00          INC TMP
020A  CA     NOPR    DEX
020B  10 F8          BPL AGAIN
020D  66 00          ROR TMP          :PUT PARITY IN C
020F  6A             ROR A            :RESTORE A WITH PARITY
0210  60             RTS
0211                 .END
```

## BASIC BANNER PROGRAM

**G. Brinkmann**

(Editor's note; when I first got this program, I couldn't believe that this short of a program could print out banners. Punch it in and try it out for yourself

(See back page for sample)

```
10 REM "BANNER"
20 REM G. BRINKMANN
30 REM PRINTER OFF
40 POKE 42001,0
50 INPUT "TEXT";A$
60 INPUT "TIMES";C
70 REM PRINTER ON
80 POKE 42001,128
90 FOR D=1TOC
100 PRINT" ":PRINT" ":PRINT" "
110 FOR I=1TO LEN(A$)
120 REM GET CHARACTER
130 B=ASC(MID$(A$,I,1))
140 IF B>63THENB=B-64
150 REM PRINTER-TAB
160 B=B+62177     F2E1
170 FOR J=1TO5
180 REM ALL TWICE
190 FOR N=1TO2
200 REM LOAD BIT#6
210 A=64:PRINT" ";
220 REM 7 ROWS
230 FOR J1=1TO7
240 Z$=" "
250 REM BIT ON?
260 IF (PEEK(B)ANDA) THEN Z$="#"
270 PRINTZ$;:PRINTZ$;
280 REM BIT-SHIFT RIGGHT
290 A=A/2
300 NEXT J1
310 PRINT
320 NEXTN
330 REM NEXT COLUMN
340 B=B+64
350 NEXTJ
360 PRINT" ":PRINT" "
370 NEXTI
380 NEXTD
390 GOTO 40
```

INTERACTIVE

```
    # # # # # #             # # # # # #         # #             # #
    # # # # # #             # # # # # #         # #             # #
# #             # #             # #             # # # #     # # # #
# #             # #             # #             # # # #     # # # #
# #             # #             # #             # #     # #     # #
# #             # #             # #             # #     # #     # #
# # # # # # # # # #             # #             # #     # #     # #
# # # # # # # # # #             # #             # #     # #     # #
# #             # #             # #             # #             # #
# #             # #             # #             # #             # #
# #             # #             # #             # #             # #
# #             # #             # #             # #             # #
# #             # #         # # # # # #         # #             # #
# #             # #         # # # # # #         # #             # #

        # # # # # #     # # # # # # # # # #
        # # # # # #     # # # # # # # # # #
    # #                 # #
    # #                 # #
# #                     # # # # # # # #
# #                     # # # # # # # #
# # # # # # # #                         # #
# # # # # # # #                         # #
# #             # #                     # #
# #             # #                     # #
# #             # #     # #             # #
# #             # #     # #             # #
    # # # # # #             # # # # # #
    # # # # # #             # # # # # #
```

NEWSLETTER EDITOR
ROCKWELL INTERNATIONAL
P.O. Box 3669, RC55
Anaheim, CA 92803 U.S.A.