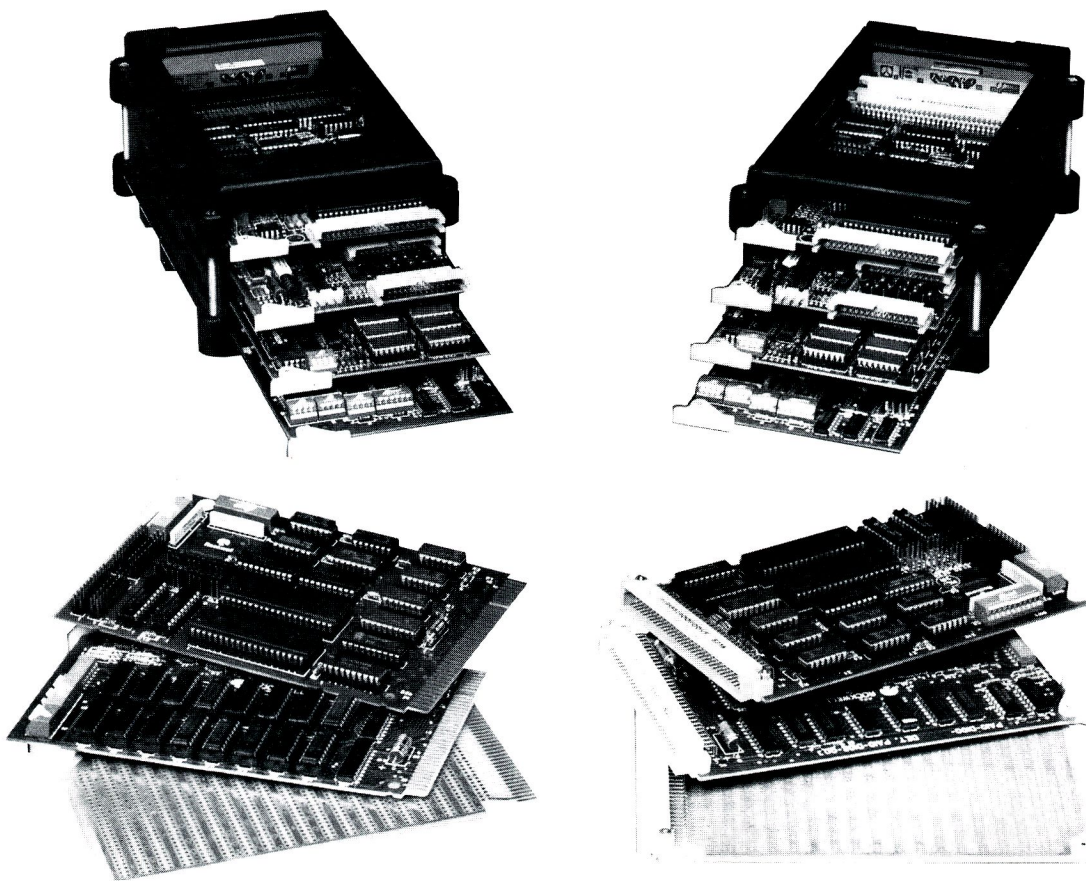


INTERACTIVE

ISSUE NO. 3

MICROFLEX 65

... lets you expand your AIM 65 or
build a standalone microcomputer system.



Rockwell International

...where science gets down to business

EDITORS CORNER

I'd like to devote an entire issue on the application of AIM 65 to Computer Aided Design (CAD).

I remember working out design problems with a calculator and thinking how much more efficient it was than the slide rule method (at that time, it was almost against the rules to have a calculator in school). I also remember having to work out the same equation over and over, changing one parameter each time, until it came out right. Those types of repetitive tasks are ideal for the computer to work on.

There must be quite a large number of design problems where parameters must be changed and solutions checked. One area that immediately comes to mind is in active filter design. Plenty of equations to work out and parameters to change here.

I'm sure that a number of you are using a BASIC equipped AIM 65 for CAD. How 'bout sharing some of those programs with the rest of us???

SUBMITTING ARTICLES

Please try to type your article double spaced. If you can't get to a typewriter, then print neatly. Don't use the editor on your AIM 65 because it is upper case only and will drive a typist to drink. Programs should be submitted on AIM 65 cassette as a BASIC or assembly language text file so the program can be assembled on a machine with a wider carriage printer for increased readability. Use a tape gap of about \$20 to compensate for any differences in equipment. Your tape will be returned to you from an appreciative editor.

PUBLISHED PROGRAMS

Several of you have mentioned that you are having problems getting the AIMPLLOT program from issue # 2 to run correctly. Besides the corrections to AIMPLLOT that are mentioned in this issue, I don't know where the problem is as of yet, but should have it figured out by the next issue. If you can't wait, send me a self addressed stamped envelope and I'll send you the fix when I get it. I will, if at all possible, try to run the programs that are published in Interactive and ask that all programs be submitted on cassette in source form. (The only program in this issue that I haven't tried is the one in the BASIC USR HELPER article.)



Editor

COPYRIGHT 1980 ROCKWELL INTERNATIONAL CORPORATION

Rockwell does not assume any liability arising out of the application or use of any products, circuit, or software described herein, neither does it convey any license under its patent rights nor the patent rights of others. Rockwell further reserves the right to make changes in any products herein without notice.

FOR YOUR INFORMATION

AIM 65/ MICROPRODUCTS APPLICATIONS ENGINEER

(714 632-0975 Use this number when you have technical questions concerning the AIM 65 system or are having difficulty interfacing to the AIM 65.

DEVICE APPLICATIONS ENGINEER

(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 repair. Their address is:

AIM 65 REPAIR
Rockwell International
6 Butterfield Trail Dr.
El Paso, TX 79924

LITERATURE & DISTRIBUTOR/DEALER INFORMATION

(714) 632-3729, 800-854-8099 (in California call 800-422-4230)- Call one of these numbers when you need literature for a certain product, information on your nearest Rockwell dealer/distributor or to request a particular application note.

SALES INFORMATION

(714) 632-3698-Call this number when you need price information for AIM 65 or Microflex 65 accessories or other Rockwell products.

SPARE PARTS

(714) 632-2190-Call this number when you want to order spare parts for your AIM 65. (The minimum cash order is \$10.)

To keep receiving this newsletter, subscribe now! The cost is \$5 for 6 issues (\$8 overseas). (**NO CASH OR PURCHASE ORDERS WILL BE ACCEPTED**) (Payment must be in U.S. funds drawn on a U.S. bank).

All subscription correspondence and articles should be sent to:

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

MICROFLEX 65 ADD-ON FAMILY NOW AVAILABLE

The first eight members of the new Microflex 65 products have been introduced by Rockwell.

The units include a module adapter for single cards, a buffer module that adapts AIM 65 to multiple-card motherboards, a 4-slot piggyback module stack, a prototyping module, an extender card for troubleshooting, an 8K static RAM card, a 16K PROM/ROM card and a two-port asynchronous communications interface adapter.

The Microflex 65 bus offers memory addressing up to 128K bytes, high immunity to electrical noise, and growth provisions for user functions.

The RM65-7101 Single-Card Adapter connects any Microflex 65 module to the AIM 65 microcomputer's system expansion connector.

The RM65-7104 Adapter/Buffer Module interfaces the AIM 65 to any Microflex 65 motherboard and will drive up to 15 modules. The RM65-7004 4-Slot Piggyback Module Stack (PMS) is the first available card cage and motherboard assembly in the Microflex 65 family. The compact PMS form factor allows low-profile packaging of a Microflex 65/AIM 65 system in a desktop or terminal style enclosure. The PMS is \$150 in single quantities.

The RM65-7201 Design Prototyping Module allows Microflex 65 users to develop their own custom circuits. Power and return lines are pre-routed and plated-through holes allow manual or automatic wire wrapping of components and sockets. The RM65-7211 Extender Card provides easy access to circuitry for probing by extending a card from a cage at enclosure, simplifying signal tracing and troubleshooting.

The RM65-3108 8K Static RAM Module uses R2114 devices arranged in two 4K memory blocks. Module features include address assignment, write protect and bank select and enable. The RM65-3216 16K PROM/ROM Module has eight 24-pin sockets, allowing installation of standard 2K, 4K or 8K ROM or PROM devices.

The RM65-5451 ACIA Module interfaces two independent, asynchronous serial I/O channels. Each may operate as a data terminal or a data set. Channel 1 provides RS-232C and 20 ma current loop interfaces and Channel 2 is an RS-232C port. Program-selectable features on each channel include word length, number of stop bits, parity, internal/external receiver clock source, and 15 data rates from 50 to 19,200 baud. On-board DC/DC converter allows +5V only operation.

The Rockwell Microflex 65 product line expands the capabilities of the AIM 65 microcomputer in Industrial, OEM, Educational and Product Development applications. The Microflex 65 family is available in edge connector or Eurocard versions.

MORE MICROFLEX 65 "ON THE WAY"

Three additional Microflex 65 family boards (the SBC, a 32K dynamic RAM, and a GPIO module) are slated for delivery in December 1980.

The SBC (Single Board Computer) features an R6502 CPU, sockets for up to 16K of PROM/ROM, 2K of 2114 RAM, and an R6522 VIA. The SBC is ideal for applications which are form-factor sensitive because of its rather compact board size (about 4" x 6.5") and also for applications requiring several additional boards because of its compatibility with all the other Microflex 65 cards.

The 32K dynamic RAM is addressable in 4K sections and features a scheme of refreshing that is complete transparent to the rest of the system. Write protect and bank select switches are included for increased versatility. An on-board DC-DC converter furnishes the necessary -5 volts so only +5 and +12 are required from the Microflex 65 bus.

The GPIO (General Purpose Input/Output) module contains two R6522 VIA devices which provide four 8-bit I/O ports and eight control lines. The 8-bit ports are fully buffered as are the control lines.

The data direction of each I/O port can be under either manual or software control.

These boards are available in either edge connector or Eurocard versions.

For further information contact your local Rockwell sales office.

PL/65 NOW AVAILABLE

PL/65, an intermediate level system-implementation language, is now available for the AIM 65.

PL/65 is designed to improve the productivity of the programmer and to increase program readability. Control statements such as conditional execution (IF-THEN-ELSE), conditional looping (FOR-TO-BY), coupled with a simplified block capability, support structured program design techniques.

The PL/65 compiler generates R6500 assembly language source code. In addition, PL/65 allows assembly language instructions to be incorporated in-line in portions of programs where timing or code optimization requirements are critical. The result is a system implementation language which has the power and flexibility of assembly language and the structuring potential of a high-level language.

The AIM 65 PL/65 compiler is contained in two 4k byte ROMs which plug directly into the AIM 65 BASIC sockets. For further information, contact Electronic Devices Division, Rockwell International, P.O. Box 3669, Anaheim, CA 92803, (714) 632-3729 or your local Rockwell sales office.

SOLVING SIMULTANEOUS EQUATIONS USING BASIC

George Sellers

(ED. NOTE: The first time I entered this program into my system, I tried to make it "look nice" by inserting spaces between commands and operands. The program wouldn't run until I eliminated all unnecessary spaces (it ran out of room). So type the program in EXACTLY as shown.)

Here is a BASIC program you might find of interest for solving simultaneous equations with up to 20 equations and 20 unknowns. It is directly transcribed from the FORTRAN program in reference (1) and is based on what is called the Gauss-Jordan Method with maximum pivot feature.

The program just fits into 4K of RAM on the AIM 65. The following table shows run times for various numbers of unknowns:

NUMBER OF EQUATIONS	RUN TIME IN SECONDS
3	1.325
5	3.015
10	14.325
20	92.485

The input is organized with the coefficients of each equation being a row of matrix "A" which is called the coefficient matrix. The right side of the equations are organized into a column which is called the constant matrix "B." The solutions are also organized into a column and this column is called the solution matrix "X."

Thus $A \cdot X = B$ in matrix algebra.

The data are entered into the program by way of the prompts for each column. The coefficient matrix can then be printed out to verify the accuracy of the input and corrections can be made if necessary. Finally, the constant matrix can be input and after a short time, the solution matrix is printed.

If matrix notation is not familiar to you, I suggest you check an advanced algebra text.

This technique forms the basis of many important types of problems i.e. network theory, regression analysis, linear programming (see BYTE magazine for most recent method of trying to solve large systems of linear equations) and economics (see Sept '80 Scientific American pg 207) to name a few.

1) Golden, James T., FORTRAN IV PROGRAMMING AND COMPUTING, Prentice Hall, Englewood, N.J. 1965

```

0 REMSIMULTANEOUS EQUATIONS
10 DIMA(20,21),X(20),LC(20),CK(20)
15 INPUT"ENTER N" ;NM
20 FORJ=1TONM;FORI=1TONM
40 PRINT"COL " ;J;"ROW " ;I;
50 INPUTA(I,J);NEXTI;NEXTJ
60 INPUT"CHECK INPUT " ;AN; IFAN#="NO"GOTO90
70 FORI=1TONM;PRINT;PRINT"ROW " ;I" " ;
75 FORJ=1TONM;PRINTA(I,J);
79 NEXTJ;NEXTI;PRINT
80 INPUT"CHANGE INPUT " ;AN; IFAN#="NO"GOTO90
85 INPUT"ROW, COL, & NEW VALUE " ;I,J,A(I,J);GOTO80
90 FORI=1TONM;PRINT"B(" ;I;" ) " ;
100 INPUTA(I,NM+1);NEXTI
110 FORI=1TONM;CK(I)=0;NEXTI
130 NP=NM+1
140 FORI=1TONM
150 IP=I+1
160 AX=0

```

```

170 FORK=1TONM
180 IF(AX-ABS(A(K,I)))>=0G0T0215
190 IFCK(K)>0G0T0215
200 LC(I)=K
210 AX=ABS(A(K,I))
215 NEXTK
220 IFABS(AX)<=10E-6G0T0500
230 CK(L)=1.0
240 L=LC(I)
250 CK(L)=1.0
260 FORJ=1TONM
270 IF(L-J)=0G0T0300
280 F=-A(J,I)/A(L,I)
290 FORK=1PTONF
295 A(J,K)=A(J,K)+F*A(L,K);NEXT K
300 NEXTJ
310 NEXTI
315 PRINT"SOLUTION IS"
320 FORI=1TONM
330 L=LC(I)
340 X(I)=A(L,NM+1)/A(L,I)
350 PRINT"X(";I;")"X(I)
360 NEXTI
370 END
500 PRINT"AX < 10E-6";END

```

```

RUN
ENTER N? 5
COL 1 ROW 1 ? 5
COL 1 ROW 2 ? 3
COL 1 ROW 3 ? 1
COL 1 ROW 4 ? 2
COL 1 ROW 5 ? 4
COL 2 ROW 1 ? -1
COL 2 ROW 2 ? 3
COL 2 ROW 3 ? -2
COL 2 ROW 4 ? 1
COL 2 ROW 5 ? 6
COL 3 ROW 1 ? 1
COL 3 ROW 2 ? -2
COL 3 ROW 3 ? 4
COL 3 ROW 4 ? -1
COL 3 ROW 5 ? -5
COL 4 ROW 1 ? 1
COL 4 ROW 2 ? -6
COL 4 ROW 3 ? 2
COL 4 ROW 4 ? -1
COL 4 ROW 5 ? -3
COL 5 ROW 1 ? 2
COL 5 ROW 2 ? 3
COL 5 ROW 3 ? -1
COL 5 ROW 4 ? 2
COL 5 ROW 5 ? 3

CHECK INPUT? YES
ROW 1 5 -1 1 1
2
ROW 2 3 3 -2 -6
3
ROW 3 1 -2 4 2
-1
ROW 4 2 1 -1 -1
2
ROW 5 4 6 -5 -3
3

CHANGE INPUT? NO
B( 1 )? 4
B( 2 )? -6
B( 3 )? 4
B( 4 )? -2
B( 5 )? 1

SOLUTION IS
X( 1 ) 2
X( 2 ) .3333333334
X( 3 ) -.4999999999
X( 4 ) .8333333335
X( 5 ) -3

```

LEARN TO TOUCH TYPE

(ED. NOTE: I LIKE this program! Talk about CAI (Computer Aided Instruction). The sound output from the last issue could easily be adapted into TOUCH to signal the operator he made a boo-boo.)

Mel Evans

1027 Redeemer

Ann Arbor, MI 48103

If you use your AIM 65 keyboard much at all, you can increase both the speed and the accuracy of your input by learning to touch-type. All it takes is the right kind of practice, and after you've got it, you'll be able to input as fast as you can read the characters, and with almost no typos. And here's the best part: you don't have to go to school to get "the right kind of practice." With the TOUCH program listed in Fig. 1, your AIM 65 can give it to you whenever, and as long as, you feel like practicing.

TOUCH is a modification for AIM 65 of a BASIC program written by Art Armstrong ("Thirty Days to a Faster Input," BYTE, Dec. 79, p.250). If you try TOUCH and decide you really want to use it, read Armstrong's article first: it is full of good tips on how to speed up the learning process.

A sample run is shown in Fig. 2. The program first asks for a set of characters to be used in the practice session. Type in any sequence of printing characters, ending with RETURN. (Start small, with ASDFG.) The program prints the selected character set, and then asks for the length and number of "words" to be used in the practice session. It then presents the first "word": a string of characters randomly chosen from the practice set.

Put your fingers on the "home" keys (ASDF left, JKL, right, as shown in the BYTE article). Your goal is to type the word without peeking at the keyboard, but peek if you must at first. After typing a character, return to the "home" keys for reference. As Armstrong says, "The important thing is to always use the same finger for each key. Otherwise the process cannot become automatic."

The teaching technique is "operant conditioning": the instant you press a wrong key, the printer tells you about it, and you start over with a new

word. After the last word, the program prints your score and a list of the characters you missed (and number of misses each). You can use this list to determine which characters to emphasize in the next session, and you can emphasize them as shown in the second session in Fig. 2. Notice that the H key was typed five times into the practice set. This makes H occur five times as often as the other characters in the resulting words.

With printer ON, you get a full record of each session, as in Fig. 2. With printer OFF, the display is the same, but only mistakes, score, and error list are printed. (This is done by using PRINT for display-only and PRINT! for display-and-print.)

Most of the differences between TOUCH and Armstrong's program are just conversion to the AIM 65 dialect of BASIC, but there are a few functional changes. The printout option mentioned above is one. I didn't incorporate his "echo" feature: it would use a lot of paper on the AIM 65 printer, and like he says, it's better practice not using it. Another change: after each session, TOUCH asks you if you want another session with the same practice characters. To quit, on SAME KEYS AGAIN? type N, and on WHICH KEYS? hit F1. Now you're back in BASIC entry mode.

For reading single keys, TOUCH uses the GET instruction, which is mentioned, but not fully explained, in the AIM 65 BASIC manual. GET reads the keyboard and returns with a character. If no key is down, it returns with the null character. If a key is down, it returns with that single character. If you GET again while the key is still down, it does not return until key up, and then returns with the null character.

This makes GET easy to use for entering strings. Observe lines 15-50 in the listing (Fig. 1). Until you press a key, line 20 repeatedly gets the null character and line 30 repeatedly adds it to C\$ (which of course does not change C\$). When you finally hit a key, it gets added to C\$ just once: GET then waits until you release the key.

But watch out! Observe line 310. It GETs B\$; but then, if it is the null character, GETs again. Otherwise it would proceed to line 350, find that B\$ (the null character) didn't match the word character, and give you an error. Try deleting that IF, and you will get a score of 0% before you can reach the first key!

The program as listed runs in about 1350 bytes of RAM. If you omit REMs and spaces, it will probably run on a 1K AIM 65. Now then. Don't just sit there. Read Armstrong's article, and then get busy!

```

2 REM LET AIM TEACH YOU TOUCH-TYPING
4 REM ORIG. BY ART ARMSTRONG (BYTE, DEC 1979 PAGE 250)
6 REM AIM MOD BY MEL EVANS (5/11/80)
8 REM
10 PRINT"WHICH KEYS?"
12 REM BUILD KEY STRING
15 C$=""
20 GET A$
22 REM EXIT ON CR
25 IF A$=CHR$(13) THEN 60

```

```

30 C#=C#+A#
50 GOTO 20
55 REM STRING BUILT SO PRINT IT
60 PRINTC#;PRINT" "
70 L=LEN(C#);DIM A(L)
80 INPUT "WORD SIZE";WL
90 INPUT "HOW MANY WORDS";NT
95 REM CLEAR SCORE COUNTERS
100 NR=0;NF=0
105 REM CLEAR ERROR COUNT
110 FOR I=1 TO L:A(I)=0:NEXT
200 FOR T=1 TO NT
205 REM BUILD WORD # T
210 NF=NF+WL
220 A#=""
230 FOR I=1 TO WL
235 REM SELECT RANDOM CHARACTER
240 P=INT(L*RND(1)+1)
250 A#=A#+MID$(C#,P,1)
260 NEXT I
265 REM PRINT WORD
270 PRINT A#
300 FOR I=1 TO WL
305 REM READ KEY & CHECK FOR MATCH
310 GET B#;IF B#="" THEN 310
350 IF B#<>MID$(A#,I,1) THEN 500
360 NR=NR+1
370 NEXT I
380 NEXT T
390 REM COMPUTE & PRINT SCORE
400 PRINT" ";PRINT!"SCORE:";INT(100*NR/NF);"% "
402 IF NR=NF THEN 414
405 PRINT!"ERRORS:";FOR I=1 TO L;IF A(I)=0 THEN 410
407 PRINT!MID$(C#,I,1);A(I)
410 NEXT I
414 PRINT" "
416 INPUT "SAME KEYS AGAIN";A#
420 IF LEFT$(A#,1)="Y" THEN 100
430 RUN
495 REM UPDATE ERROR COUNT
500 FOR J=1 TO L
510 IF MID$(C#,J,1)<>MID$(A#,I,1) THEN NEXT;GOTO 520
515 A(J)=A(J)+1
520 PRINT!"*** ERROR ON ";MID$(A#,I,1)
525 REM WAIT A BIT & THEN RETURN
530 FOR I=1 TO 300:NEXT
540 GOTO 380
    
```

BASIC TIME SAVER

Gordon Smith
Rockwell International

(EDITORS NOTE: According to Gordon, the basis for the program came from a similar program published in TARGET which he modified quite extensively and added auto-line numbering. This article was reprinted from the Rockwell Anaheim Hobby Club Newsletter.)

This issue I have what I think is a real goody for all of you who are using AIM BASIC and are either marginal or lazy typists or both – I suspect that includes most of us. I have named the program “BASIC HELPER” because that is what it does. It is a combination automatic line numberer and common basic command automatic typist.

I will describe usage of the program first and then how it works. The program occupies the top two pages of a 4K AIM (0E00-0FFF). Consequently when BASIC is entered via the ‘5’ Key, you must respond to the “MEMORY SIZE” question with 3584 or less. This gives you 3054 bytes free for keying-in segments of your program. Note that this places no limitation on the size of the program you are keying-in, because as these 3054 bytes are filled they may be dumped on tape. Then the next segment with its proper line numbers may also be keyed in after typing “NEW.”

As each segment fills memory it is dumped on tape and the next segment then keyed-in. This is permissible on AIM because unlike PET and APPLE, loading a program from tape does not automatically wipe out the old program. It appends the later program to the earlier one and only if both segments have some common line numbers is there any conflict. In this case any common numbered lines from the last blocked tape will be the lines that survive.

Sooo – this is an unusual machine language program used with BASIC because it ultimately takes none of the BASIC space.

After the BASIC is enabled, exit via ESCAPE and load the BASIC HELPER PROGRAM (or VISA VERSA). I use the F1 and F3 keys to activate automatic line numbering with shorthand and F2 to activate only the shorthand. This later option is used when keying in someone else’s program (or fixes to your own) when nice even increments between lines are not achievable.

With the F1 keys the display asks “FROM = ” to which you respond with the starting line number desired (delete is allowed or the last four hex characters will be used-no leading zero’s required). Hit “space” or “RETURN” and the display will prompt with “INC = .” At this point key-in the increment you want and hit “space” or “RETURN” and you are in WARM START BASIC with the starting line number already showing. Enter the rest of the line, hit RETURN, and the new line number is there ready for you.

This automatic line numbering is handy but it is only part of the story. By using the control key and any of the alphabet keys (Except M) and F2 and F3 you can also get an automatic entry of what I think are the most used, longest, or messiest-to-type BASIC commands. For instance Control I gives you “INPUT,” Control L gives you “LEFT\$(” etc. There are even some split BASIC commands. Control O gives you “ON” then you type the variable name or expression without any spaces. When you key in “space,” the program completes the GOTO portion of the ON---GOTO statement. Control F3 gives ON---GOSUB and Control T gives you IF---THEN.

The complete list of shorthand commands and mnemonic aids for helping to remember them are given in a table following the program listing and command tables.

As I see it there are three major benefits of this program:

- 1) It makes it much faster to type
- 2) There are many fewer typing errors
- 3) The correct form for these commonly used statements are either input or prompted (\$ in strings aid the left (as a prompt)).

The program works as follows: The F1 key jumps to 0E00 which clears the display and then calls the “FROM” subroutine. The line number is stored in two places OFFB,C where it is held for updating and OFF1,2 for processing to input it to BASIC which then displays it. The segment starting at 0E18 inserts a space and then sequentially outputs INC by three output subroutine calls.

The JSR EAAE at 0E27 is the ADDIN subroutine which is not described in the manuals. It outputs “ = ” and waits for four (more or less) hex characters. If less it assumes leading zero; if more it accepts the last four. The increment value may be as many as four digits and is stored in OFF3,4. In all of the above cases the data is stored in high byte-low byte sequence (it is not an address in the conventional sense).

The segment of the program from 0E39 through 0E3F sets a Flag in OFF7 to indicate if the auto increment mode is ON (OO) or OFF (FF). The F2 entry point is at 0E3D.

The group of instructions from 0E42 through 0E4E set up the user input mode (55 in A412 and the user input vector to location 0E62 in Location 0108,9).

The next group through 0E5E initializes conditions so that the last character output (OFF5) was apparently a carriage return so that the line number will be output if that mode is active. Any value other than 20 in OFFD is OK-20 indicates that a split command has had the first part entered and is waiting for the “space” code to enter the second segment.

The OO in OFF6 indicates that the program is not in the middle of inputting a line number.

The segment named 'USER PROCESSOR' is the point where the user input is vectored to. In BASIC at this point, the Y register is pointing to the BASIC input buffer location for the next key-in so it must be saved (in this case on the stack). The USER input may have to turn something on when it is entered the first time so the first time the carry is clear on subsequent entries the carry must be set. In this case it doesn't make any difference so that I could have left out the instructions in OE64, 66, and 67 and the SEC instructions in OE92 and OECF. But I left them in because I am trying to teach some with this column too.

The mode decode function works as follows: The test at OE6E determines if the auto-line option is desired-if it is the test at OE72 determines if it is necessary to output a line number or if we are in process-if the answer is yes we execute the segment at OE94. We will come back to this later. The test at OE76 determines if we are in process of generating a command. These in-process tests are necessary because only one character is generated and supplied to BASIC and BASIC comes back for the next one. If we are in process of generating a command, the jump at OE78 is executed. Otherwise, we use the CUREAD subroutine to read a character from the keyboard and then save it in the last character buffer, OFF5.

This character must be examined to see if it is a control-alpha (OE83) to start outputting a new command at OEF9 or if it matches (20 matching 20) the split command wait indicator in OFFD (OE88). If so, the JMP OFIA in OE8A is executed. If it is none of these, it is an old-fashioned, plain ordinary key stroke input. In this case BASIC's Y register is restored, the input character recovered, and the RTS takes it back to the BASIC input processing.

The Auto increment processing is performed as follows: The test in OE97 determines if this is the first digit to be processed. If it is, the Line input in Process Flat at OFF6 is set to 4 and also the character count is checked for zero (OEA4). If it is, the segment of instructions from OEO1 through OEF7 restores flags and increments (in

decimal) the line number for the next line. This program then returns to look for the next input character.

If it was not the final pass, the segment from OEA6 through OEBD shifts the next digit (most significant digit first) into OFFO. OEB4 counts the number of digits down by one and OEC2 through OEC7 converts it to ASCII and holds it in OFFA for output after the BASIC Y register is restored OECA through OEDO.

If the character input was a control character (value less than 1F) the program will commence from OEF9. Since this entry is for the start of a new command the code input 01 to 1E will be used as a pointer to the start of text. It is moved to the Y register and is used to point to a byte in Table 1 "POINTERS TO START OF EACH COMMAND." This is accomplished in locations OEF9 through OEFA. That byte is then used to index the text table. "COMMANDS IN ASCII," Table 2. The Y value is then incremented and saved for the next pass. If the input byte was 00 (OF07), it indicates "end of text" so some flags are restored and the next key is input (OF27 to OF2c). If the input byte was 20 (OFOB) it indicates a split command so the wait code (20) is stored in OFFD and the next key is input (OF14-OF17). If neither of these, the code is output using the sequence of instructions OEC7-OEDO in the auto- line number output section.

If the 20 matches 20 test in OE88 indicates that the second segment of a split statement is to be inserted, the program starting at OF1A through OF25 is executed. This cancels the "wait" Flag at OFFD and puts a low value (00) into the last command byte (OFF5) so that it indicates a shorthand command in-process and then loads the pointer for the next character of the command. It then continues to execute as if it were a normal shorthand command.

This is a longer and more complex program than I have generated for this column before but I think that you will like it. I think I will have it loaded whenever I am keying-in a BASIC program. It saves so much time and aggravation.

SHORTHAND COMMANDS AND MNEMONIC AIDS

A = ABS(P = POKE
B = TAB(TABBBB	Q = RND(QUANTITY
C = MID\$(CENTER\$	R = RETURN
D = DATA	S = STR\$(
E = RIGHT\$(END\$	T = IF+++ THEN TEST
F = FOR	U = USR(
G = GOTO	V = VAL(
H = LEN(HOW LONG	W = INT(WHOLE VALUE
I = INPUT	X = RESTORE X-OUT READ
J = GOSUB JUMPSUB	Y = READ PARAMETER
K = GET GET KEY	Z = STEP SIZZE
L = LEFT\$(F1 = IS NOT USABLE
M = IS NOT USABLE	F2 = DEFFN FUNCTION
N = NEXT	F3 = ON+++ GOSUB
O = ON+++ GOTO	

010C 4C JMP 0E00
 010F 4C JMP 0E3D
 0112 4C JMP 0E00

BASIC HELPER PROGRAM

0E00 20 JSR E9F0
 0E03 20 JSR E7A3
 0E06 AD LDA A41C
 0E09 8D STA OFF2
 0E0C 8D STA OFFC
 0E0F AD LDA A41D
 0E12 8D STA OFFB
 0E15 8D STA OFF1

INPUT INCREMENT

0E18 20 JSR E83E
 0E1B A9 LDA #49
 0E1D 20 JSR E97A
 0E20 A9 LDA #4E
 0E22 20 JSR E97A
 0E25 A9 LDA #43
 0E27 20 JSR E97A
 0E2A 20 JSR EAAE
 0E2D AD LDA A41C
 0E30 8D STA OFF4
 0E33 AD LDA A41D
 0E36 8D STA OFF3

BYPASS AUTO LINE NO.

0E39 A9 LDA #00
 0E3B F0 BEQ 0E3F 02
 0E3D A9 LDA #FF
 0E3F 8D STA OFF7

SET UP USER INPUT

0E42 A9 LDA #55
 0E44 8D STA A412
 0E47 A9 LDA #62
 0E49 8D STA 0108
 0E4C A9 LDA #0E
 0E4E 8D STA 0109

PARAMETER SET UP AND GO TO WARM START

0E51 A9 LDA #0D
 0E53 8D STA OFF5
 0E56 8D STA OFFD
 0E59 A9 LDA #00
 0E5B 8D STA OFF6
 0E5E 4C JMP B003

USER/PROCESSOR

0E62 98 TYA
 0E63 48 PHA
 0E64 B0 BCS 0E68 02
 0E66 68 PLA
 0E67 60 RTS

MODE DECODE ——— SUBSTITUTE OR READ

0E68 AC LDY OFF5
 0E6B AD LDA OFF7
 0E6E D0 BNE 0E74 04
 0E70 C0 CPY #0D
 0E72 F0 BEQ 0E94 20
 0E74 C0 CPY #1F
 0E76 B0 BCS 0E7B 03
 0E78 4C JMP 0F0F
 0E7B 20 JSR FE83
 0E7E 8D STA OFF5
 0E81 C9 CMP #1F

0E83 90 BCC 0EF9 74
 0E85 CD CMP OFFD
 0E88 D0 BNE 0E8D 03
 0E8A 4C JMP 0F1A
 0E8D 68 PLA
 0E8E A8 TAY
 0E8F AD LDA OFF5
 0E92 38 SEC
 0E93 60 RTS

AUTO-INCREMENT

0E94 AD LDA OFF6
 0E97 D0 BNE 0EA1 08
 0E99 A9 LDA #04
 0E9B 8D STA OFF6
 0E9E 8D STA OFF8
 0EA1 AD LDA OFF8
 0EA4 F0 BEQ 0ED1 2B
 0EA6 A9 LDA #00
 0EA8 8D STA OFF0
 0EAB A9 LDA #04
 0EAD 8D STA OFF9
 0EB0 18 CLC
 0EB1 2E ROL OFF2
 0EB4 2E ROL OFF1
 0EB7 2E ROL OFF0
 0EBA CE DEC OFF9
 0EBD D0 BNE 0EB0 F1
 0EBF CE DEC OFF8
 0EC2 AD LDA OFF0
 0EC5 69 ADC #30
 0EC7 8D STA OFFA
 0ECA 68 PLA
 0ECB A8 TAY
 0ECC AD LDA OFFA
 0ECF 38 SEC
 0ED0 60 RTS

PREPARE FOR NEXT LINE

0ED1 A9 LDA #20
 0ED3 8D STA OFF5
 0ED6 A9 LDA #00
 0ED8 8D STA OFF6
 0EDB F8 SED
 0EDC 18 CLC
 0EDD AD LDA OFFC
 0EE0 6D ADC OFF4
 0EE3 8D STA OFFC
 0EE6 8D STA OFF2
 0EE9 AD LDA OFFB
 0EEC 6D ADC OFF3
 0EEF 8D STA OFFB
 0EF2 8D STA OFF1
 0EF5 D8 CLD
 0EF6 B8 CLV
 0EF7 50 BVC 0E7B 82

SHORTHAND COMMAND INSERTION

0EF9 A8 TAY
 0EFA B9 LDA 0FD1,Y
 0EFD A8 TAY
 0EFE B9 LDA 0F2F,Y
 0F01 C8 INY
 0F02 8C STY OFFE
 0F05 C9 CMP #00
 0F07 F0 BEQ 0F27 1E
 0F09 C9 CMP #20
 0F0B F0 BEQ 0F14 07
 0F0D D0 BNE 0EC7 B8

0F0F AC LDY OFFE
 0F12 D0 BNE 0EFE EA
 0F14 8D STA OFFD
 0F17 4C JMP 0E7B
 0F1A A9 LDA #00
 0F1C 8D STA OFFD
 0F1F 8D STA OFF5
 0F22 AC LDY OFFE
 0F25 D0 BNE 0EFE D7
 0F27 A9 LDA #20
 0F29 8D STA OFF5
 0F2C 4C JMP 0E7B

POINTERS TO START OF EACH COMMAND

<M> = 0FD1 00 01 06 0B
 < > 0FD5 11 16 1E 22
 < > 0FD9 27 2C 32 38
 < > 0FDD 3C 43 45 4A
 < > 0FE1 52 57 5C 63
 < > 0FE5 69 71 76 7B
 < > 0FE9 80 88 8D 8D
 < > 0FED 8D 94 9A

COMMANDS IN ASCII

<M> = 0F30 41 42 53 28
 < > 0F34 00 54 41 42
 < > 0F38 28 00 4D 49
 < > 0F3C 44 24 28 00
 < > 0F40 44 41 54 41
 < > 0F44 00 52 49 47
 < > 0F48 48 54 24 28
 < > 0F4C 00 46 4F 52
 < > 0F50 00 47 4F 54
 < > 0F54 4F 00 4C 45
 < > 0F58 4E 28 00 49
 < > 0F5C 4E 50 55 54
 < > 0F60 00 47 4F 53
 < > 0F64 55 42 00 47
 < > 0F68 45 54 00 4C
 < > 0F6C 45 46 54 24
 < > 0F70 28 00 0D 00
 < > 0F74 4E 45 58 54
 < > 0F78 00 4F 4E 20
 < > 0F7C 47 4F 54 4F
 < > 0F80 00 50 4F 4B
 < > 0F84 45 00 52 4E
 < > 0F88 44 28 00 52
 < > 0F8C 45 54 55 52
 < > 0F90 4E 00 53 54
 < > 0F94 52 24 28 00
 < > 0F98 49 46 20 54
 < > 0F9C 48 45 4E 00
 < > 0FA0 55 53 52 28
 < > 0FA4 00 56 41 4C
 < > 0FA8 28 00 49 4E
 < > 0FAC 54 28 00 52
 < > 0FB0 45 53 54 4F
 < > 0FB4 52 45 00 52
 < > 0FB8 45 41 44 00
 < > 0FBC 53 54 45 50
 < > 0FC0 00 00 00 44
 < > 0FC4 45 46 46 4E
 < > 0FC8 00 4F 4E 20
 < > 0FCC 47 4F 53 55
 < > 0FD0 42

1E (EDITOR'S NOTE: If you'd like a set of stick-on labels for the basic one-key entry program, send \$1 to Ron Riley, POB 4310, Flint, Mich. 48504. These Labels are printed on adhesive-backed stock with all the proper Basic commands printed on them.)

PROM PROGRAMMER CARD FOR AIM 65

A PROM Programmer and Code Editor (CO-ED) module is now available as a plug-on peripheral for the AIM 65 printing microcomputer from Rockwell.

The PROM memory devices programmed with the new module may then be used with any 6500-based system, including AIM 65, Microflex 65, and SYSTEM 65.

The module provides PROM check, read and verify functions in addition to programming. Data load, verify and dump, each with offset, and an object code editor (CO-ED) are additional features included in the module's built-in ROM firmware. CO-ED controls a program pointer and can search, disassemble and modify R6500 object code programs.

R6551 ACIA CHIP NOW AVAILABLE

The R6551 Asynchronous Communication Interface is now available from Rockwell. This new device offers several advantages over older ACIA designs. The main advantage is that the R6551 contains its own on-chip baud rate generator with 15 program-selectable rates from 50 baud to 19,200 baud. The only additional component required is a standard 1.8432 MHz crystal.

The R6551 has programmable word lengths of 5, 6, 7, or 8 bits; even, odd, or no parity; and 1, 1½ or 2 start bits. Besides the normal interface control lines (RTS-Request To Send, CTS-Clear To Send, and DCD-Data Carrier Detect) the R6551 provides two additional lines, to further enhance the modem interface. These two lines are DTR-Data Terminal Ready (which indicates the R6551 status to the modem) and DSR-Data Set Ready (indicates the status of the modem to the R6551).

The built-in programmable baud rate generator offers certain advantages to the system designer. Since fewer external components are required for boards designed around the R6551, more compact and/or more densely designed systems are possible. This in turn translates to a cost savings which can become very substantial as the quantity of systems to manufacture increases.

For data sheets and more information on the R6551 ACIA contact your local Rockwell sales office.

The PROM programmer CO-ED module, part number A65-901, plugs directly into the expansion connector of the AIM 65 microcomputer. The module includes 1K byte of R2114 static RAM which, when used with the 4K RAM AIM 65 model, allows single-pass programming of 4K × 8 PROMs. It also includes internal logic to select PROM programming characteristics for the Intel 2758, 2716, or 2732, or the TI 2508, 2516 or 2532 without switch or jumper changes.

The module requires only a single supply voltage, +5 VDC @ 0.7 amp, which is usually available from the power supply for the host AIM 65. Appropriate PROM programming voltage levels are generated by an on-board DC-DC converter.

The module measures approximately 4.4 inches wide by 6.7 inches long and is fully assembled, tested and warranted.

For more information contact the Electronic Devices Division of Rockwell International, P.O. Box 3669, Anaheim, CA 92803. Telephone (714) 632-3729 or your local Rockwell sale office.

NEW APPLICATION NOTE

A new application note entitled PRINTER CONTROL WITH THE R6522 is (document #256) is now available from Rockwell. This note describes how the AIM 65 can be used to directly control all the functions of a dot matrix printer mechanism through the on-board user R6522 VIA chip. The printer mechanism chosen is the Two-Day Corporation 80 column bidirectional 10600 series.

This 24 page app. note actually contains two complete software/hardware interface schemes—one for each of the two printer mechanisms available from the Two-Day Corp.

One of the models (the 10600A) has a synchronous motor drive while the other (the 10600B) has a stepper motor drive and an independent paper feed.

For a copy of this or other app. notes write: Literature Request, Rockwell International, Box 3669 RC55, Anaheim, CA 92803. Be sure to specify the document numbers.

INTERRUPT DRIVEN KEYBOARD

Marvin DeJong
Pt. Lookout, MO

(Ed. note-Although the author cites amateur radio as an example application for an interrupt driven keyboard, this technique is just as relevant in the possible industrial uses of the AIM 65. Interrupt driven systems are becoming increasingly useful in data gathering applications as well as machine control areas.)

The AIM 65 Monitor polls the AIM 65 keyboard for key depressions by calling subroutines. These subroutines wait for a key to be depressed before continuing to execute the commands that have been entered or before continuing to process the data that have been entered. There are certain situations in which this treatment of the keyboard is undesirable. In this application note such a situation is described, and a routine to read the keyboard on an interrupt basis is described.

Suppose an amateur radio operator wishes to use the AIM 65 to send either Morse code or RTTY (radioteletype). The AIM 65 monitor routines could be used for this, but the send routine would have to wait for a key depression before it could send the character, and the operator would have to wait for the send routine to finish sending the character before he could type in a new character. The usual technique calls for a buffer that stores the characters typed on the keyboard, and concurrently sends the characters at a prescribed speed. Thus, the operator can type in characters as quickly as he can type, and the send routine empties the buffer at the prescribed speed. This form of keyboard operation is difficult, if not impossible, to achieve with the AIM 65 monitor software which, in addition to reading the keyboard, must also debounce the keys.

An alternative approach is to let the send program (or any other program in which this interrupt approach is used) continue operating, but use a regular interrupt to scan the keyboard to see if any new characters have been entered. If a new character has been entered on the keyboard, it can be stored in a buffer to await its turn to be processed by the main program. If no new character has been keyed, the interrupt routine branches around the buffer storage instructions.

The listings given here form a routine that will read the AIM 65 keyboard on an interrupt basis. The initialization routine sets up the interrupt vector to point to the interrupt routine at SOBFF (of course, the locations of all of these routines may be changed). Next the initialization routine sets up the T1 timer on the user's 6522 to produce equally spaced interrupts, at five millisecond intervals. (Longer intervals can also be used, but shorter intervals may produce keybounce errors.) The last instruction in the initialization routine produces an infinite loop that simulates the user's main program, a Morse code send program for example.

The interrupt routine starting at SOBFF is very similar to the AIM 65 GETKEY subroutine in the AIM 65 monitor. Most of the coding is taken

from that routine, with some important modifications to make it operate on an interrupt basis. Note that all the registers are saved by the interrupt routine. Also note that the interrupt routine contains a JSR \$ODOO instruction. If a key depression is detected, then the accumulator contains the ASCII representation of the key just prior to the JSR \$ODOO instruction. The subroutine at \$ODOO is expected to place the accumulator contents in a memory location where it can be processed by the main program, a buffer for example.

Finally, we have included a display routine at \$ODOO that displays the key just pressed on the AIM 65 display. This routine is included to test the initialization and interrupt routines. It has no other use.

```

‡ I/O
UT1L   = $A004
UT1CH  = $A005
UT1LL  = $A006
UARC   = $A00B
UIER   = $A00E
IRQV2  = $A404
CPIY   = $A42A
CPIY1  = $A42E
ROLLFL = $A47F
DRA2   = $A480
DRB2   = $A482
‡ MONITOR SUBROUTINES
ONEKEY = $ED05
ONEK2  = $ED0B
OUTDD1 = $EF7B
PHXY   = $EB9E
PLXY   = $EBAC
ROW1   = $F421

```

```

2000                                     * = $0E00
OE00
OE00   A9 FF                               LDA #<INTRN
OE02   8D 04 A4                           STA IRQV2
OE05   A9 0B                               LDA #>INTRN
OE07   8D 05 A4                           STA IRQV2+1
OE0A   78                                  SEI
OE0B                                       ‡
OE0B                                       ‡SET T1&CB1 INT FLAG
OE0B   A9 C0                               LDA #$C0
OE0D   8D 0E A0                           STA UIER
OE10                                       ‡
OE10                                       ‡
OE10                                       ‡SET T1 IN FREE
OE10                                       ‡RUNNING MODE
OE10   A9 40                               LDA #$40
OE12   8D 0B A0                           STA UARC

```

0E15		‡	0C24	‡
0E15		‡SET T1 LATCHES	0C24	‡CHECK ROW 1
0E15	A9 88	LDA #88	0C24	AD 82 A4 LDA DRB2
0E17	8D 06 A0	STA UT1LL	0C27	4A LSR A
0E1A	A9 13	LDA #13	0C28	‡
0E1C	8D 05 A0	STA UT1CH	0C28	‡IF=1, NO CTRL OR
0E1F	58	CLI	0C28	‡SHIFT
0E20		‡	0C28	B0 20 BCS GETK1
0E20		‡PROGRAM SIMULATION	0C2A	‡
0E20		ME	0C2A	‡CLMN 5,6,7(CNTRL
0E20	4C 20 0E	JMP ME	0C2A	‡SHIFTL,SHIFTR)
			0C2A	A2 03 LDX #3
0E23		*=\$0BFF	0C2C	‡
0BFF			0C2C	‡CTRL OR SHIFT,WHICH
0BFF		‡SAVE REGISTERS	0C2C	A9 7F LDA #7F
0BFF		INTRN	0C2E	GETK0
0BFF	48	PHA	0C2E	38 SEC
0C00	20 9E EB	JSR PHXY	0C2F	6A ROR A
0C03		‡	0C30	48 PHA
0C03		‡CLEAR T1 INT	0C31	‡
0C03	AD 04 A0	LDA UT1L	0C31	‡LETS GET CTRL OR
0C06		‡	0C31	‡SHIFT INTO X
0C06		‡SEE IF KEY DOWN	0C31	20 0B ED JSR ONEK2
0C06	AD 82 A4	LDA DRB2	0C34	AD 82 A4 LDA DRB2
0C09	C9 FF	CMP #FF	0C37	‡
0C0B	F0 07	BEQ ROO1	0C37	‡ONLY ROW 1
0C0D		‡	0C37	4A LSR A
0C0D		‡ACCEPT LAST KEY	0C38	90 06 BCC GETK00
0C0D	0D 7F A4	ORA ROLLFL	0C3A	68 PLA
0C10		‡	0C3B	CA DEX
0C10		‡INVERT STROBES	0C3C	D0 F0 BNE GETK0
0C10	49 FF	EOR #FF	0C3E	‡
0C12	D0 43	BNE ROONEK	0C3E	‡NO KEY, SO EXIT
0C14		‡	0C3E	F0 5B BEQ NOKEY
0C14		‡CLEAR MASK	0C40	‡
0C14		ROO1	0C40	‡GET STBKEY INTO X
0C14	A2 00	LDX #00	0C40	GETK00
0C16	8E 2A A4	STX CPIY	0C40	68 PLA
0C19		‡	0C41	‡
0C19		‡GO THROUGH KB ONCE	0C41	‡CLMN INTO X
0C19		‡AND RETURN IF ANY	0C41	AD 2B A4 LDA CPIY1
0C19		‡KEY	0C44	‡
0C19		‡Y=ROW(1-8)& STBKEY	0C44	‡COMPLEMENT STRBS
0C19		‡=COLUMN, IF NO KEY	0C44	49 FF EOR #FF
0C19		‡Y=0,	0C46	‡
0C19		STBKEY =#FF	0C46	‡CTRL OR SHIFT TO X
0C19		‡	0C46	AA TAX
0C19	20 05 ED	JSR ONEKEY	0C47	‡
0C1C	88	DEY	0C47	‡SET MSK=#01
0C1D	30 7C	BMI NOKEY	0C47	EE 2A A4 INC CPIY
0C1F		‡	0C4A	‡
0C1F		‡CHCK CLMN 5,6,7	0C4A	‡NOW GET ANY KEY
0C1F	A9 8F	LDA #8F	0C4A	GETK1
0C21	8D 80 A4	STA DRA2	0C4A	20 05 ED JSR ONEKEY

0C4D		‡		0C71		‡SHIFT
0C4D		‡CHCK THE ROW (1-8)		0C71	F0 24	BEQ GETK7
0C4D	88	DEY		0C73		‡
0C4E		‡		0C73		‡CTRL?
0C4E		‡CHCK IF CTRL OR		0C73	29 10	AND ##10
0C4E		‡SHIFT		0C75		‡
0C4E	D0 09	BNE GETK1B		0C75		‡NO, GO GETK5
0C50		‡		0C75	F0 06	BEQ GETK5
0C50		‡ENTERED LAST		0C77	68	PLA
0C50	AD 2B A4	LDA CPIY1		0C78		‡
0C53		‡		0C78		‡MASK OFF 2 MSB FOR
0C53		‡		0C78		‡CONTROL
0C53		‡IF CLMN 5,6,7,8 DO		0C78	29 3F	AND ##3F
0C53		‡IT AGAIN		0C7A		‡
0C53	C9 F7	CMF ##F7		0C7A		‡EXIT TO DISP
0C55	B0 04	BCS GETK2		0C7A	4C 98 0C	JMP GETK8
0C57		‡		0C7D		GETK5
0C57		‡GET CTRL OR SHIFT		0C7D	68	PLA
0C57		ROONEK		0C7E		‡SAVE IT
0C57	90 42	BCC NOKEY		0C7E	48	PHA
0C59		GETK1B		0C7F		‡
0C59	30 40	BMI NOKEY		0C7F		‡IF ALPHA CHARS DO
0C5B		GETK2		0C7F		‡NOT SHIFT
0C5B	EA	NOP		0C7F	29 40	AND ##40
0C5C	EA	NOP		0C81	D0 14	BNE GETK7
0C5D	EA	NOP		0C83	68	PLA
0C5E		‡		0C84	48	PHA
0C5E		‡MULT BY 8		0C85		‡
0C5E	98	TYA		0C85		‡ONLY LSB
0C5F	0A	ASL A		0C85	29 0F	AND ##0F
0C60	0A	ASL A		0C87		‡
0C61	0A	ASL A		0C87		‡DO NOT INTERCHANGE
0C62		‡		0C87		‡(SPACE) OR 0
0C62		‡NOW A HAS ROW ADDR		0C87	F0 0E	BEQ GETK7
0C62		‡FROM ROW 1		0C89		‡
0C62	A8	TAY		0C89		‡ACC>=\$0C?
0C63		‡		0C89	C9 0C	CMF ##0C
0C63		‡ADD CLMN TO Y		0C8B		‡
0C63	AD 2B A4	LDA CPIY1		0C8B		‡YES ACC>=\$0C
0C66		GETK3		0C8B	B0 05	BCS GETK6
0C66	4A	LSR A		0C8D		‡
0C67	90 03	BCC GETK4		0C8D		‡EXIT
0C69	C8	INY		0C8D	68	PLA
0C6A	D0 FA	BNE GETK3		0C8E	29 EF	AND ##EF
0C6C		‡		0C90	D0 06	BNE GETK8
0C6C		‡GET THE CHR		0C92		‡
0C6C		GETK4		0C92		‡ACC>=\$0C
0C6C	B9 21 F4	LDA ROW1,Y		0C92		GETK6
0C6F	48	PHA		0C92	68	PLA
0C70		‡		0C93		‡
0C70		‡CTRL OR SHIFT USED?		0C93		‡BIT 4=1
0C70	8A	TXA		0C93	09 10	ORA ##10
0C71		‡		0C95	D0 01	BNE GETK8
0C71		‡BRCH IF NO CTRL OR		0C97		GETK7

SUPER SIMPLE AUTO-START

Under normal circumstances, when power is applied to the AIM 65, the reset line is automatically asserted by the power-on-reset circuitry associated with the 555 timer (Z4) and the CPU starts executing the reset sequence contained in the Monitor ROM.

While this sequence of events is fine for most uses, there are others, such as OEM installations and dedicated controller applications, which require that the system comes up running a user written operating program without the need for any special operator intervention.

The first solution that usually comes to mind involves the replacement of the Monitor ROMS. However, there is an easier way to accomplish the same effect without having to sacrifice all those built-in I/O drivers.

The only restriction is that the user program must start at address \$B000, \$B003, or \$D000 (corresponding to the '5,' '6' or 'N' key vectors).

If you have the Assembler or BASIC ROMs installed in your system, try holding down the 'N' key (or '5' key) while you turn on the power. Notice how it comes up running in the assembler?

In a dedicated controller application, where the keyboard isn't being used, the same effect can be achieved by installing a 16-pin DIP header

in the keyboard socket on the main board and shorting the two pins that correspond to the '5,' '6' or 'N' key.

If you wish the system to automatically jump to \$B000 on power up, short pins 11 and 13 on the DIP header (pins 12 and 14 for address \$B003) or pins 3 and 14 for a starting address of \$D000.

For OEM applications where the keyboard is still needed, the same effect can be achieved by temporarily shorting the correct pins with a reed relay driven by a timer chip. The time constant should be slightly longer than that of the power-on-reset timer (Z4) for proper operation.

CORRECTIONS TO ISSUE #2

In the DISASSEMBLER UTILITY on page 11, the periods should be removed from in front of the labels DEB and LECT in the source listing.

The OFFSET LOADER program on page 13 was missing the immediate symbol (#) from all four of the immediate instructions (locations 0200, 0205, 0222, and 0252).

In the WE'VE GOT OUR EARS ON article on page 10, the correct Post Office box is 3669 (not 33093).

A note table was left out of the AIM 65 SOUND article on page 8. It belongs right before the section headed "HERE'S HOW TO MAKE MUSIC".

B0=251 (B below first C)	B=124
C=237 (first C)	C1=117 (C above first C)
C#=224	C1#=111
D=211	D1=104
D#=199	D1#=99
E=188	E1=93
F=177	F1=88
F#=167	F1#=83
G=157	G1=78
G#=149	G1#=73
A=140	A1=69
A#=132	

In the AIM PLOT program on page 4, the opcode at location 0232 should be 8D (not BD). Also, the bit instructions at locations 02A9 and 02BA are somewhat misleading. What the author really needed in this situation was a BIT IMMEDIATE instruction. But, as the 6502 doesn't have such an instruction, he had to simulate it. He did this by finding the proper bit pattern in the AIM 65 monitor ROMS and using the address of this bit pattern as the operand instead of the bit pattern itself thereby accomplishing the same effect as a BIT IMMEDIATE instruction.

Has anyone generated a table of all the bit patterns (\$00-\$FF) available in the AIM 65 monitor ROMS? I'd sure like to publish it.

```

0C97 68          PLA
0C98          ⋮
0C98          ⋮GO DISPLAY
0C98          GETKB
0C98 20 00 0D    JSR DISP
0C9B          ⋮
0C9B          ⋮RESTORE REGISTERS
0C9B          NOKEY
0C9B 20 AC EB    JSR PLXY
0C9E 68          PLA
0C9F A9 00      LDA #$00
0CA1 8D 2A A4    STA CPIY
0CA4 40          RTI
0CA5          *#=$0D00
0D00
0D00          DISP
0D00 A2 13      LDX ##13
0D02 09 80      ORA ##80
0D04          ⋮
0D04          ⋮CONVERT X INTO ADDR
0D04          ⋮FOR DISPLAY
0D04 20 7B EF    JSR OUTDD1
0D07 60          RTS
0D08          .END
    
```

LETTERS TO THE EDITOR

Dear Editor

I would like to see a clearer explanation of how to output data to the printer than that illustrated in the AIM 65 Users Manual, ref Chapter 7. I would like to write an assembly language program, and then at some point in that program have the printer print the contents of either one of the index registers, accumulator, or a memory location. Quite simply put, I'd like an easy to follow subroutine that would do as shown:

```
LDA XXXX ;some memory location
JSR PRINT ;print contents of acc.
```

then return to a users program.
Is there an easy way to do this?

Thank you
R. A. Fairman

Mr. Fairman,

There is a subroutine called NUMA (\$EA46) that will output a hex value in the accumulator as two ASCII digits, but if the printer is not enabled, it will just be sent to the display. So, to do what you want will require making sure the printer is on before you JSR to NUMA. The best way to accomplish this is with a short subroutine that gets included in your program. The subroutine will have to save the present printer flag from PRIFLG (\$A411), force the printer flag on by making it \$80, do a JSR to NUMA, and then restore the printer flag to whatever it was before.

```
PRTBYT SEC
ROR PRIFLG
JSR NUMA
ASL PRIFLG
CLC
RTS
```

Bit 7 is the only bit in the printer flag (PRIFLG) that has any meaning. It is this bit that gets tested to see if output gets sent to the printer (see line 2411 and 2412 in the AIM 65 monitor listing for an example of testing the printer flag). This means that the remaining 6 bits are free for our use. The first thing that is done upon entry to PRTBYT is to set the carry flag to a '1.' This will be put into bit 7 of the print flag location by the ROR PRIFLG instruction to "turn the printer on." The ROR PRIFLG instruction actually does two things. First, it rotates the carry flag into the bit 7 position of the printer flag (which turns the printer on) and also saves the previous print flag in bit 6 so it can be restored after we're finished. The instruction JSR NUMA outputs the character to the printer. To restore the printer flag to its original condition we execute the ASL PRIFLG instruction. This simply shifts bit 6 back to the bit 7 position and moves the '1' which was shifted into the bit 7 position from the carry flag back to the carry flag. The carry flag is then cleared to prepare for our return to the main routine.

the Editor

Dear Sir,

Congratulations on a really-usable, cost-effective AIM65 and now a newsletter to match.

ADDITION TO MARK REARDONS REAL TIME CLOCK

(issue #1 p. 11)

```
250 REM NOW THE TIME IS H HOURS, M MINUTES, S
    SECONDS
251 REM BUG! NOT IF H OR M CHANGED WHILE PEEKING.
252 REM IF THEY DID TIME COULD BE OUT BY ONE HOUR.
253 REM FIX! TO DOUBLE CHECK THE TIME.
254 IF PEEK(220)=H OR PEEK(221)=M THEN GO TO 240
255 REM NOW! THE TIME IS H HOURS, M MINUTES, S
    SECONDS.
```

≠ = NOT EQUAL TO (sorry, no arrows).

SORRY! YOU LOSE YOUR BET we did know how to edit BASIC programs but bet you didn't know you don't need tape to do it.

If memory is split between the EDITOR and BASIC then a LOAD can be done using the memory read routine used by the Assembler. The procedure is described below and assumes you have 4K of memory to be split down the middle to the EDITOR and BASIC.

1. Allocate the MREAD routine (FADO) to the user input vector at HEX 0108. 0108 = DO 0109 = FA.
2. Initialize BASIC answering 2047 to MEMORY SIZE? prompt. From here on only re-enter BASIC using command "6." *width = 60 (Basic Editor)*
3. Escape from BASIC and initialize the EDITOR answering; FROM "800" TO "FFF" to the prompts. From here on only re-enter the EDITOR using command "T."
4. You may now load your program into the EDITOR with the usual commands. Entry may be either from tape or the keyboard but observe the following rules;
 - a. The top line must always be a SPACE only.
 - b. The bottom line must always be CONTROL Z only.
 - c. Always exit the EDITOR using "T" then "Q". This leaves the pointer on the top line.
5. When you want to LOAD your program into BASIC exit the EDITOR and re-enter BASIC. LOAD in the usual way but answer "U" to the IN prompt where upon you will find your program being zapped into BASIC memory space.
6. You may now RUN your program in the usual way but escaping to the EDITOR when editing is required and then re-load the program into BASIC for executing. *ESC*

7. Sometimes you may find it more useful to do program modifications direct on the BASIC program leaving the EDITOR unaltered so you can quickly restore the original by re-loading. To re-load you must re-enter the EDITOR to get the pointer on the top line otherwise nothing will get loaded.

This EDITOR-BASIC technique has hidden depths which only become apparent with use and the application of a little ingenuity. So get to it!

Here are some clues.

1. The EDITOR can have direct commands entered into it such as NEW or RUN and these will be directly executed as the program is loaded.
2. Normal program statements may be entered without line numbers and these will be directly executed as well. This is particularly useful for POKEing in machine code without occupying BASIC program space.
3. If the EDITOR contains two lines with the same number the second will overlay the first. So don't erase a line in order to replace it—just type in the new line after it. Erase a line only when your sure you won't want to go back to it.
4. A line can be temporarily erased either by inserting REM before the line number or by inserting just the line number on the line after the one to be erased.
5. If you are going to LOAD an EDITOR tape containing direct commands into BASIC the tape must have remote control connected.
6. If REMarks are entered into the EDITOR with-out line numbers then they will not get loaded into BASIC space. Thus it is possible to have a lavishly commented EDITOR tape for development use and a fast loading BASIC tape for the user.
7. You can LOAD part of a program in the EDITOR by putting a SPACE line before the section wanted and a CONTROL-Z line after it. Don't use "T" when exiting the EDITOR, leave it pointing at the SPACE line before the section you want.
8. Because LOAD does not erase existing lines, a large program can be built up and debugged by over-laying from a fairly small EDITOR space.

An extension of the above techniques allows the writing of long self-loading over-laid programs operating within the AIM 65's 4K ram. The approach is ideal for Automatic Test Equipment (A.T.E.) programs and if the Editor so wishes I will write further on it in the future.

KEN FULLBROOK
England

Ken,
You're right. I didn't know how to edit BASIC without using the cassette. Thanks a bunch!!! I'm sure our readers will appreciate it.

The Editor

SOFTWARE REVIEW

by the EDITOR

How would you like to develop 1802 programs on your AIM 65?

Or, how would you like to be able to set up a library of MACROS which can be called from your assembly language programs?

If either, or both of these things interests you, then you'll be interested in a new software package for the AIM 65 called MACRO.

MACRO is actually a pre-processor that works in conjunction with the AIM 65 assembler. Its function is to accept a source file that contains macro calls, expand those macros by looking them up in a library file, and outputting a new source file with all the macros expanded so that the AIM 65 ROM assembler can assemble it.

The macro library file must be set up which defines all the macros which are to be used and must be memory resident at the time the input file is submitted for expansion. (makes AIM 65 sound like a large machine, doesn't it?)

Here's an example of what it looks like:

SAMPLE MACRO

```
INCD POINTR
```

SAMPLE MACRO DEFINITION

```
&INCD
INC !1
BNE *+4
INC !1+1
&
```

SAMPLE MACRO OUTPUT

```
INC POINTR
BNE *+4
INC POINTR+1
```

(The '&' character is used both to start and terminate a macro definition)

Now that last little programming sequence (incrementing a double byte pointer) is something 6502 programmers do alot of.

The same technique can be used to set up a cross assembler for most any other CPU. (6800,1802,8080 etc) Pretty excitin' stuff!!!

According to the documentation that accompanies MACRO, the minimum usable system is an AIM 65 with 2K of RAM, the assembler ROM, and remote control of least one cassette deck. The price is \$15 which includes documentation and a cassette of the object code. The source code for MACRO is available either on cassette or as a listing for an additional \$30. (This would enable you to adapt MACRO to your 6502 floppy system)

So far, I haven't found any bugs in the system (I'm good at finding bugs) and it worked right the first time I tried it.

It's available from: POLAR SOLUTIONS
Box 268
Kodiak, Alaska 99615

TEMPERATURE CONVERSION PROGRAM

(This program was reprinted from the Rockwell Hobby Club Newsletter of the Anaheim CA facility).

If you've ever had the need to convert temperatures from Celsius to Fahrenheit, then here is a program written in AIM 65 BASIC that will make life a little easier.

Just follow the prompts and type in the start and end values in degrees Fahrenheit and the program will print out a table of the temperature in degrees centigrade (Celsius).

PROGRAM

```
490 INPUT "START"; S
492 INPUT "FINISH"; F
493 PRINT! "DEG.", "DEG."
494 PRINT! "FAR.", "CELS."
495 DEF FNA(A)=INT(A*100+.5)/100
500 FOR I=S TO F
505 R=(I-32)*5/9
510 PRINT! I,FNA(R)
515 NEXT I
```

SAMPLE PRINTOUT

```
START? 0
FINISH? 8
DEG.      DEG.
FAR.      CELS.
0         -17.78
1         -17.22
2         -16.67
3         -16.11
4         -15.56
5         -15
6         -14.44
7         -13.89
8         -13.33
```

FOUND HIDING . . .

a BASIC command not found in the manual

**Dale Hall
Torrance, CA**

Statement	Syntax/Function	Example
POS	POS (expression) Returns print head position 0-19. requires a dummy argument.	Print POS(0)

BASIC USR HELPER

**Georges-Emile April
Montreal, Canada**

(Ed. note-If you call many machine language subroutines from your BASIC programs, this routine should be able to save you some time.)

I find it inconvenient to have to use POKE statements every time I wish to use machine language programs; so I wrote the following set of machine language programs which may be assembled on top of RAM memory, or placed in a ROM somewhere else.

Version shown here is in last page of 4K RAM. See listings.

The programs work in the following manner:

- a) Program SETADD

When called, this program takes the argument passed to it by BASIC and places it in 4 and 5 to be used as an address by next access to "USR" function.

- b) Program CALLIT

This program uses program SETADD to set up address of "USR" then calls program

The programs are used as follows:

Two subprograms (lines 1 and 2) are written in Basic.

Line 1 sets up 4 and 5 to point to SETADD, then returns.

Line 2 sets up 4 and 5 to point to CALLIT, then returns.

Two situations may arise:

- a) It is desired to call machine language program (lets us call it SUB1), that needs an argument (ARG).

The following sequence will call it the first time:

```
100 GOSUB1: X=USR(SUB1): X=USR(ARG)
```

Where SUB1 is decimal value of address of machine language program we wish to call. Subsequent calls to the same program can be made simply by X=USR(ARG) since address has been set up by line 100.

b) It is desired to call machine language program (SUB2). That needs no argument (e.g. input data).

The followings will do just that:

```
150 GOSUB2: X=USR(SUB2)
```

or

```
160 GOSUB1: X=USR(SUB2): X=USR(DUMMY)
```

Lines 150 and 160 are fully equivalent but line 150 will execute faster. As was the case in a), subsequent calls to same program can simply be X=USR(DUMMY). Where DUMMY can be any valid variable or constant, since program needs no argument.

It should be noted that program SETADD returns a value of 0 to basic so that the following sequence does not modify the value of Y:

```
170 GOSUB1: Y=Y + USR(SUB1)
```

Therefore line 100 could have been written:

```
100 GOSUB1: X=USR(SUB1) + USR(ARG)
```

ROUTINES TO EASE USE OF USR(X)

```
; $DF=223, $DB=219
```

```

==0FD8 FIXIT 6006B0 JMP (FIX1) ; TRANSFORM DATA TO FIXED POINT
; THE FOLLOWING SETS UP DATA AS ADDRESS OF 'USR', THEN RETURNS
; TO BASIC WITH USR(X)=0
; SHOULD BE USED WHEN FUNCTION REQUIRES ARGUMENT
; 'GOSUB1' SETS UP LINK TO THIS ROUTINE

==0FD8 SETADD A5A9 LDA EXP
C990 CMP #90 ; SEE IF TOO LARGE FOR SIGNED TREATMENT
D000 BNE OK
A5AA LDA MSD ; IF TOO LARGE, TAKE MSD & MSD1 AS DATA
8505 STA LINKH
A5AB LDA MSD1
==0FE7 COMMON 8504 STA LINKL
A900 LDA #0
85A9 STA EXP ; MAKE USR(X)=0
60 RTS

==0FEE OK 20D80F JSR FIXIT ; TRANSFORM ARGUMENT TO ADDRESS
A5AC LDA LSD1
8505 STA LINKH
A5AD LDA LSD
4CE70F JMP COMMON ; GO COMPLETE TRANSFER

; THE FOLLOWING SETS UP ADDRESS FROM ARGUMENT
; THEN CALLS ROUTINE
; USED WHEN NO ARGUMENT IS NEEDED
; 'GOSUB2' SETS UP LINK TO THIS ROUTINE

; $DF=223, $FA=250

==0FFA CALLIT 20D80F JSR SETADD
600400 JMP (LINK) ; CALL ROUTINE
==1000 END . END
ERRORS= 0000

```

QUICK INSERTION ROM SOCKETS

Ron Riley
Flint, MI

I recently purchased PL-65 and after switching between BASIC and PL-65 ROMS several times, I decided to look into using zero-insertion force sockets. The problem with most of these sockets is that they require more room than AIM 65 has available. I did finally locate one that fits, however. It is part #504012459 and is available from WELLS ELECTRONICS INC., 1701 S. MAIN ST., SOUTH BEND, IND 46613. Since the zero insertion sockets gets plugged into the normal ROM socket, no desoldering is necessary. ROMs can now be swapped in and out with no danger of damaging the ROM or the socket.

BASIC RECOVERY PROCEDURE

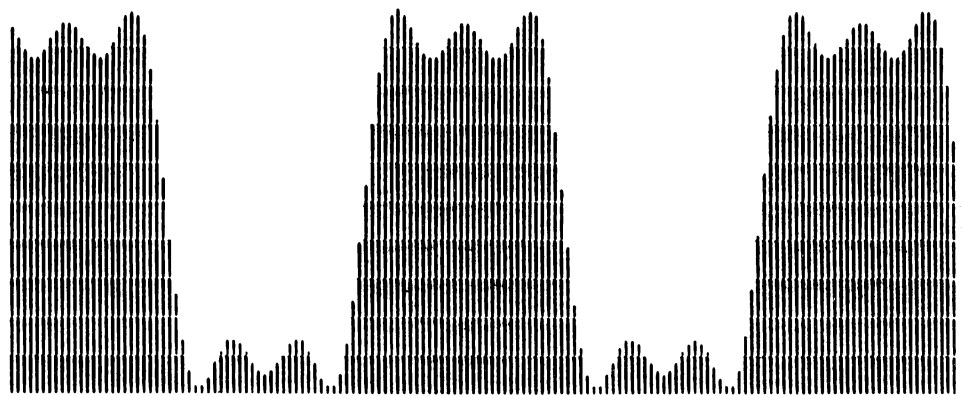
by Antonio Berges
Dominican Republic

How many times have you entered a rather long BASIC program into the AIM 65, confidently entered the monitor and then inadvertently hit the "5" key (rather than the "6" key) to reenter BASIC? You saw the ominous MEMORY SIZE? question and probably thought your BASIC program was "down the tubes."

As long as you don't press the RETURN key, you're safe. The program in memory can be recovered by hitting the ESC key followed by M 01 RETURN / 00 B9. BASIC can now be reentered with the "6" key. What you've done is to replace the JMP SCEA3 at location \$0000 to a JMP SB900.

SNEAK PREVIEW

Here's an example of the type of graphics you'll be able to generate with a program that will be in the next issue of INTERACTIVE. Stay tuned!!!



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

Bulk Rate U.S. POSTAGE RATE Santa Ana Calif. PERMIT NO. 15
--