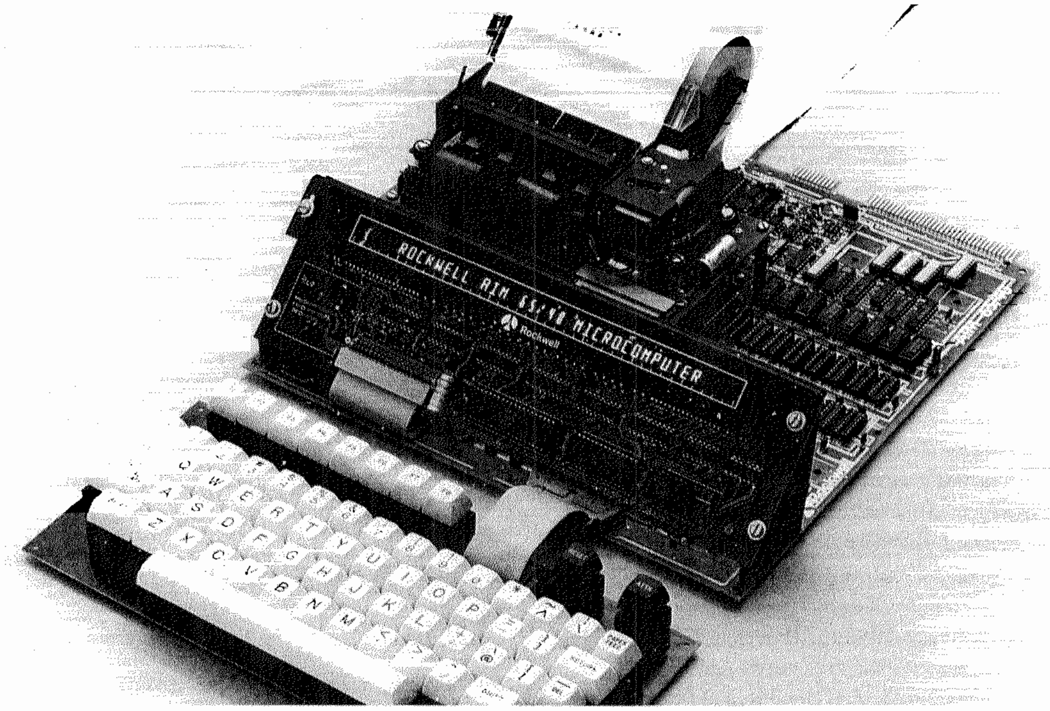
INTERACTIVE

ISSUE NO. 5

AIM 65/40 . . .



THE NEXT GENERATION!

Rockwell International

...where science gets down to business

EDITOR'S CORNER

I want to thank all you supporters who have been sending in articles, comments, suggestions etc. It's nice to know that INTERACTIVE has so many fans out there. We have a pretty good mix of articles in this issue with maybe a bias towards data files. But, that's what you seem to be interested in.

Keep in mind that this publication is a dynamic entity. You are the force behind it. Whatever you collectively say GOES. If you wish to influence the direction we're taking, then write an article about the subject you'd like to see. It's as simple as that!

I would like to see more articles on how to interface the AIM 65 to different devices such as A/D, D/A, counter chips, DVM chips, speech synthesizers, graphic output, etc. etc. etc. . . .

How about it?

I have received some good stuff in the area of CAD (Computer Aided Design). Not enough for a complete issue, though, so I'll start running them in issue #6 (or #7).

We're getting ready to do another update on the AIM 65 User's Guide. If you have found any errors or think we could explain something better, let us know. Send all comments to the attention of THE DOCUMENTATION MANAGER, Rockwell Intl., POB 3669, RC55, Anaheim, CA 92803.

Two interesting articles appeared recently in EDN magazine. The January 7, 1981 issue carried two articles which featured AIM 65. One of them showed how a mechanical engineer could simulate a physical model on a BASIC language equipped AIM 65. The other article gave complete details (hardware and software) so an AIM 65 (or other 6502/6522 system) could control the intensity or speed of ac operated devices such as lamps or motors through an interrupt driven zero crossing detector.

If you don't have access to this magazine, we can send you reprints of the articles. Just ask for EDN #1 if you want the ac power interface or EDN #2 for the digital simulation article. Send requests to the attention of SALES SUPPORT SERVICES, Rockwell Intl., POB 3669, RC55, Anaheim, CA 92803.

A version of the PASCAL programming language is now "in the works" for AIM 65. At this point, all the information I can give you is that it will consist of a five ROM set and be a subset of Standard Pascal which was defined in a book called "Pascal User Manual and Report" by Jensen and Wirth. No, there's no data sheet as of yet so please don't call or write until we say that more information is available. This is not a product announcement . . . just some advance information that is intended to give a hint about where Rockwell is heading. More on Pascal later.

Eric C. Rehnke
Newsletter Editor

All subscription correspondence and articles should be sent to:

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

FOR YOUR INFORMATION

From the Editor:

Here are some books that may help you along on the road to mastering microcomputers.

BASIC FOR HOME COMPUTERS by Albrecht, Finke, and Brown. Published by John Wiley & Sons (605 Third Ave., New York, NY 10016).

PROGRAMMING AND INTERFACING THE 6502 by Marvin De Jong. Published by Howard W. Sams & Co. (4300 W. 62nd St., Indianapolis, Ind 46268).

THE FOLLOWING BOOKS ARE AVAILABLE FROM ROCKWELL INTERNATIONAL AT SPECIAL PRICES:

6502 SOFTWARE DESIGN by Leo J. Scanlon. Published by Howard W. Sams & Co. 6502 Assembly language tutorial and hardware interfacing examples. $7.00 (U.S. & Canada) $9.00 (overseas)

MICROCOMPUTER SYSTEMS ENGINEERING by Camp, Smay, and Triska. Published by Matrix Publishers (30 NW 23rd Place, Portland, ORE 97210) General intro to microcomputing, 6502, 6800, and 8080 Assembly language programming, and some system design principles. $17.00 for U.S. and Canada and $19.00 overseas.

AIM 65 LABORATORY MANUAL AND STUDY GUIDE by Leo J. Scanlon. Published by John Wiley & Sons. Provides 17 programming and I/O experiments for the AIM 65. $5.00 (U.S. & Canada) or $7.00 (overseas)

ORDERING INSTRUCTIONS for books available from Rockwell: Orders must be accompanied by payment. U.S. and Canadian orders must be by check or money order and overseas payment must be drawn on U.S. bank. California residents add 6% state tax. Send orders to the attention of SALES SUPPORT SERVICES, Rockwell Intl, POB 3669, RC55, Anaheim, CA 92803.

CORRECTION TO THE AIM 65 USER'S GUIDE

There seems to be a problem with the program on pages 8-37 and 8-38 of the AIM 65 User's Guide (Rev 3, December 1979). Insert the sequence HERE JMP HERE between ;CONTINUE and the dotted line

(Continued on page 22)

COMING SOON . . . AIM 65/40

Rockwell International will shortly be introducing the AIM 65/40. The AIM 65/40 microcomputer is made up of an R6502 based single board computer with on-board expansion to 65 kilobytes of memory, a full graphic 280 × N dot matrix or 40-column alphanumeric printer, a 40-character alphanumeric display, and a full ASCII keyboard with user assignable function keys.

An advanced generation of Rockwell's popular AIM 65 microcomputer, the AIM 65/40 will be available as a complete system or as individual computer and intelligent peripheral modules.

The AIM 65/40 Series 1000 single board computer modules feature system address expansion up to 128K bytes with on-board memory up to 48 kilobytes of RAM and up to 32 kilobytes of ROM or EPROM. Six level priority interrupt logic and six 16-bit multi-mode timers are included for flexibility in production automation and laboratory control applications. Extensive I/O capability provides an RS-232C asynchronous communications interface channel with programmable data rates of up to 19,200 baud for terminals or modems, plus a 20 ma current loop TTY interface, dual audio cassette interfaces, and two user-definable 8-bit parallel ports with handshake control two 16-bit timer/counters and an 8-bit serial shift register.

Three additional 8-bit parallel ports are directly programmable as dictated by the user's application to provide more TTL level I/O or interface to keyboards, displays, and printer modules. Manufacturer supplied ROM resident software included with the AIM 65/40 Series 1000 computer provide I/O drivers for the intelligent peripherals and more. The printer connector is compatible with the Centronics parallel interface that is so popular with high speed dot matrix printers.

A buffered system bus accommodates off-board expansion via Rockwell's RM 65 microcomputer modules which include intelligent peripheral controllers for mini or standard floppy disks, CRT monitors and the IEEE-488 instrumentation bus, plus additional communications interfaces and a selection of RAM, ROM and PROM memory expansion options up to 128K bytes of memory and memory-mapped I/O capacity.

The AIM 65/40 Model 0600 graphics printer module consists of an intelligent microprocessor controller integrated with the printer mechanism. This module operates in two modes. Character mode operation prints upper and lower case ASCII characters, mathematical symbols, and semi-graphics character font formatted as 40-characters/line at 240 lines/minute. Full graphics mode outputs any data pattern desired as a 280×N dot matrix. With its own microprocessor controller, user changeable character generator ROM, thermal head drivers, motor control, and parallel handshake ASCII interface, this freestanding peripheral minimizes demand on the AIM 65/40 central processor, permitting maximum system performance.

The Model 0400 display module features a bright, crisp vacuum flourescent 40-character alphanumeric display. This stand-alone module has its own microprocessor controller for display of alphanumeric, special, and limited graphics characters, parallel handshake ASCII interface, support circuitry and operates from a single +5 volt power supply. Special control commands permit variable display timing, cursor control, autoscroll, and character blinking.

The Model 0200 keyboard module provides a terminal style alphanumeric and special character keyboard matrix with 64 keys, including locking ALL CAPS, control, and eight user definable function keys. Three keys labelled ATTN, RESET, and PAPER FEED have dedicated lines to the interface connector.

The AIM 65/40 Series 5000 incorporates a ROM resident software system and integrates all four modules into a complete microcomputer system. The interactive monitor software controls the AIM 65/40 system with single keystroke, self-prompting commands, supports software development with assembler, debug and control commands. A multi-file text editor supports both line and screen editing functions. Optional languages include a fully symbolic R6500 assembler and BASIC. FORTH, PASCAL, and PL/65 software packages are in development.

The AIM 65/40 is expected to be available sometime during the third quarter of 1981.

For price and delivery information contact your local Rockwell sales office.

DATA FILES FOR AIM-65 BASIC

Jerry K. Radke
U.S. Dept. of Agriculture

The storage and retrieval of data on a permanent (or semipermanent) medium is often necessary. Unfortunately, Rockwell AIM-65 BASIC does not provide data file capability for its cassette recorder interface. Even worse, Microsoft does not provide a listing of the BASIC it wrote for the AIM-65 so the user can easily modify it. However, the procedure presented here will provide the user of the AIM-65 with a cassette data file capability that is relatively painless though not very elegant.

I use two short BASIC subroutines to open files (one each for read and write) and one to write an end-of-file. These statements start at 9000. I usually reserve certain blocks of data statement numbers for certain subroutines which can be saved and loaded individually, e.g. 4000's are reserved for my real-time clock and timing subroutines, 5000's are my sorting subroutines, 6000's are for my formatted printing subroutines, etc. This allows me to build programs using these standard subroutines as modules.

In addition to the three subroutines, some BASIC statements are needed in the main program to control the tape recorder(s) and to select the active output device (AOD) and active input device (AID). The remote control lines to the tape recorders should be functional. The minimum procedure to write on tape is to call the subroutine at 9000 to open a file, set the AOD to "tape", print (via BASIC "PRINT" statements) to tape, returning AOD to "display", and finally end-filing the tape by calling the subroutine at 9100. This causes the 80 byte tape buffer to fill and dump to tape in blocks while automatically turning the tape recorder on and off. Reading tapes is performed by calling the subroutine at 9200 to open the file, setting the AID tape, "INPUTting" the data, and returning the AID to the "keyboard".

To make the data files compatible with text files that are written and read by EDITOR, a few additional things should be done. The first five characters "PRINTed" to the tape buffer should be the filename. (The first position in the buffer was set to indicate block zero by statement 9010 thus the filename takes up characters 2 through 6). The 7th character must not be a CR (SOD) or it will not be accepted by EDITOR as a text file. EDITOR also wants to see two consecutive CR's at the end of the file to indicate EOF. The EOF subroutine does this as well as filling the rest of the block with "nulls". However, the user is free to set up his 80 byte blocks to suit his own needs, e.g. a special character to indicate EOF. Obviously, to read data from tapes, a proper INPUT format is necessary to match the way the data is stored. The filename will also need to be INPUT from block 0.

The program on page 5 gives an example that we can follow. Statements 20 through 50 load array P$. Statement 60 inputs a title for the data (not the filename). Statements 90–120 sets up tape recorder 1 or 2 for output and turns the tape controls off. (User should respond with a 1 or 2 to statement 90). At statement 120, place tape recorder in "record" mode and answer query. Input "filename" at 140. Statements 150–230 actually do the writing to tape. Note that 170 prints the filename, a comma, and the number of data lines (N). Commas are necessary if more than one data element are to be read per line. Statement 240 turns the tape recorders on to allow the user to reposition the tapes if necessary. The tape read example is similar. Statements 560–630 input the data, 640–690 prints the data, and 700 turns the tape controls back on. The user can place the recorder in the "play" mode after the prompt "?" is displayed for statement 580. Of course, the tape should be properly placed in a gap just before the start of the desired file.

Statements should be kept to a minimum while the AOD or AID is set to "tape". If data is going to be written or read several different times in the program, return AOD or AID to "keyboard/display" after each PRINT or INPUT loop or routine. In other words, only have the AOD or AID set to "tape" when absolutely necessary. I have not tried all combinations possible, but do know that data can be easily written or corrected by the EDITOR and read as data by BASIC. I would be interested in hearing about any "discoveries" you make. If you have questions, I can be reached at 612/589-3411 during normal working hours.

This procedure offers quite a bit of flexibility, and I have left it this way even though a neater package could be written using WHEREIN and WHEREOUT and putting almost everything in the subroutines. One thing to remember with this routine is that the tape must be positioned so that block zero will be the first block read. This can be changed if desired, however. Also, a search procedure could be used to locate block zero of a given file.

MINIMUM STATEMENTS TO WRITE ON CASSETTE TAPE

```
     *
     *                 USER PROGRAM
     *
     GOSUB 9010        OPEN FILE WRITE
     POKE 42003,84     ACTIVE OUTPUT DEVICE SET TO
                         "TAPE"
     *
     *                 USER PRINT STATEMENTS TO
                         TAPE
     *
     POKE 42003,13     ACTIVE OUTPUT DEVICE
                         RETURNED TO "DISPLAY"
     GOSUB 9110        WRITE EOF ON TAPE
     END
```

MINIMUM STATEMENTS TO READ FROM TAPE

```
     *
     *                 USER PROGRAM
     *
     GOSUB 9210        OPEN FILE (READ)
```

```
POKE 42002,84       ACTIVE INPUT DEVICE SET TO
                      "TAPE"
*
*                   USER INPUT STATEMENTS TO
                      READ FROM TAPE
*
POKE 42002,13       ACTIVE INPUT DEVICE RETURNED
                      TO "KEYBOARD"
*
*                   USER PROGRAM
*
END
```

TAPE SUBROUTINES

```
9000 REM:  OPEN
       FILE (WRITE)
9010 POKE  278,0      $0116 TO 0 (SET 1ST CHAR IN BUFF
                        FOR BLK 0)
9020 POKE  42039,1    SET OUTPUT TAPE POINTER
                        ($A437) TO "1"
9030 POKE  360,0      BLOCK COUNT ($0168) TO ZERO
9040 POKE  41993,22   SET TAPE GAP
                        ($A409) TO $16
9050 RETURN
9100 REM:  WRITE-
       EOF
9110 POKE  42003,84   SET OUTFLG TO "T"
9115 PRINT CHR$(13)   OUTPUT OD,OD,QA
9120 NL=80-PEEK       CHECK POINTER FOR BUFFER
       (42039)          SPACE
9130 FOR NC=1 TO NL   FILL BUFFER WITH NULLS
9140 PRINT CHR$(0);
9150 NEXT NC
9160 POKE  42003,13   SET OUTFLG TO "D"
9170 RETURN
9200 REM:  OPEN
       FILE (READ)
9210 POKE  277,0      SET BLOCK ($0115) TO ZERO
9220 POKE  42038,80   SET COUNTER ($A436) TO END
                        ($50)
9230 RETURN
```

EXAMPLE PROGRAM

```
1   DIM P$(40)
10  REM: TAPE WRITE EXAMPLE
20  INPUT "# ENTRIES" ;N
30  FOR I=O TO N-1
40  PRINT "ENTRY # " ; I + 1; :INPUT PS(1)
50  NEXT I
60  INPUT "TITLE" ;H$
70  INPUT "STORE ON TAPE Y/N" ;A$
80  IF A$="N" THEN STOP
90  INPUT "T = "; T:T=T-1
100 POKE 42037, T:REM: SET TAPOUT
110 POKE 43008,204:REM: TURN TAPES OFF
120 INPUT "TAPE READY Y/N";A$
130 IF A$="N" THEN STOP
140 INPUT "FILENAME" ;A$
150 GOSUB 9010:REM: OPEN FILE
160 POKE 42003,84:REM: TAPE AOD
170 PRINT A$ ; """ ; N
180 PRINT H$
190 FOR I=O TO N-1
200 PRINT I + 1; "," ;P$(I)
210 NEXT I
220 POKE 42003,13:REM: DISPLAY AOD
230 GOSUB 9110:REM: WRITE EOF
240 POKE 43008,252:REM: TURN TAPES ON
250 END

500 REM: TAPE READ EXAMPLE
510 DIM R(40), R$(40)
520 INPUT "READ TAPE Y/N"; A$
530 IF A$="N" THEN STOP
540 INPUT "T ="; T:T=T-1
550 POKE 42036,T:REM: SET TAPIN
560 GOSUB 9210:REM: OPEN FILE
570 POKE 42002,84:REM: TAPE AID
580 INPUT A$,N
590 INPUT H$
600 FOR I=O TO N-1
610 INPUT R(I),R$(I)
620 NEXT I
630 POKE 42002,13
640 PRINT " "
650 PRINT! " ";PRINT!H$
660 FOR I=O TO N-1
670 PRINT! R(I); TAB(5);R$(I)
680 NEXT I
690 PRINT! " "
700 POKE 43008,252
710 END
```

Some useful locations:

Hex	Decimal	Label	Remarks
$0115	277	BLK	Block count for input (must be zero to start)
$0116	278	TABUFF	80 byte tape buffer starts here
$0168	360	BLKO	Block count for output (set to zero)
$A409	41993	GAP	Block gap for tape recorder
$A411	42001	PRIFLG	Printer "ON" = 0, "OFF" = 128 ($80)

MORE BASIC DATA FILES

Steve West and Frank Nunneley
Johannesburg, South Africa

(EDITOR'S NOTE: Yes, I know that you've already seen a data file handling program. But, this program is a bit different and it shows a neat way to add new commands to AIM 65 BASIC.)

The ability to process and store data on cassette greatly enhances the usefulness of BASIC programs.

Any system of this type should be easy to use. The method described here extends the instruction set of BASIC to include instructions to open and close files and to input and output data. The new instructions are:

(Continued from previous page)

$A409	41993	GAP	Block gap for tape recorder
$A411	42001	PRIFLG	Printer "ON" = 0, "OFF" = 128 ($80)
$A434	42036	TAPIN	Tape 1 or 2 controls for input) default = 1) if not changed
$A435	42037	TAPOUT	Tape 1 or 2 controls for output) (otherwise last)
$A436	42038	TAPTR	Tape buffer pointer for input
$A437	42039	TAPTR2	Tape buffer pointer for output
$A800	43008	DRB	Data Reg B for monitor 6522—PB4 and PB5 turn tape controls on and off. (1) (2)

Hex	Decimal	Remarks:
$CC	204	Both tapes OFF
$DC	220	Tape 1 on, 2 off
$EC	236	Tape 2 on, 1 off
$FC	252	Both tapes on

Useful Monitor Subroutines

Hex	Decimal	Hi Decimal	Lo Decimal	Remarks
$E6BD	59069	230	189	Toggle Tape #1 control
$E6CB	59083	230	203	Toggle Tape #2 control

PRINT#'NAME'1	Opens a cassette output file. The name of the file is in single quotes and is followed by the recorder number. (Default is T=1)
PRINT#A,B$	Outputs data to the currently open output file. Format is identical to standard PRINT statement.
PRINT##	Closes current output file.
INPUT#'NAME'2	Opens an input file by finding the file "NAME". The file name is again followed by the recorder number (Default to tape recorder 1)
INPUT#A$,B$	Inputs data from currently open input file.
INPUT##	Closes Input file.

Only one tape buffer is available while BASIC is in use, thus only one I/O file can be open at a time.

To use BASEX, BASIC must be limited to 3883 bytes in response to the question "MEMORY SIZE?" when entering BASIC. Answer "WIDTH?" as before, then ESCape to monitor and Load BASEX from cassette. Reenter BASIC using 6 and the extension program is ready to work. This order is important as the divert routine on page zero must be modified after BASIC is initialized.

The assembly listing follows. When entering this file in source it is recommended that the editor be placed above $800; the assembler symbol table can be placed between 200 and 800. This way the Editor won't be corrupted when the program is tested. After entering BASIC after assembling the file it will be necessary to modify the instructions on page zero using Mneumonic Entry. After the file is working and the initialization procedure from tape is used this is *not* required.

```
<*>=C8
<I>
 00C8 4C JMP 0F2D
 00CB EA NOP
 00CC
<
```

When the file is working dump it (object) to cassette, the link to the extension must be included here.

```
<D>
FROM=F2D TO=FFF
OUT=T F=BASEX T=1
MORE?Y
FROM=C8 TO=CB
MORE?N
```

```
2000                ;**  TAPE DATA FILES
2000                ; STEVE WEST AUG '80

2000                PHXY    =$EB9E
2000                PLXY    =$EBAC
2000                CRLF    =$E9F0
2000                LL      =$E8FE
2000                OUTFLG  =$A413
2000                INFLG   =$A412
2000                OUTDIS  =$EF05
2000                TOBYTE  =$F18B
2000                DILINK  =$A406
2000                DUMPTA  =$E56F
2000                TAPOUT  =$A435
2000                TAPIN   =$A434
2000                DRB     =$A800
2000                DU11    =$E50A
2000                NAME    =$A42E
2000                LOADTA  =$E32F
2000                PNTR    =$C6
2000                        *=$F2D
0F2D
0F2D                BASEXT
0F2D  C9 97                 CMP #$97
0F2F  F0 0C                 BEQ PRINT
0F31  C9 84                 CMP #$84
0F33  F0 34                 BEQ INPUT
0F35  C9 3A                 CMP #$3A
0F37  B0 03                 BCS NOTNUM
0F39  4C CC 00              JMP $CC
0F3C  60            NOTNUM  RTS

0F3D  48            PRINT   PHA
0F3E  20 9E EB              JSR PHXY
0F41  A0 01                 LDY #1
0F43  B1 C6                 LDA (PNTR),Y
0F45  C9 23                 CMP #'#
0F47  F0 06                 BEQ STATAP
0F49                PR1
0F49  20 FE E8              JSR LL
0F4C  4C 63 0F              JMP EXIT
0F4F                STATAP
0F4F  A9 54                 LDA #'T
0F51  8D 13 A4              STA OUTFLG
0F54  C8                    INY
0F55  B1 C6                 LDA (PNTR),Y
0F57  C9 27                 CMP #'''
0F59  F0 39                 BEQ OPENFL
0F5B  C9 23                 CMP #'#
0F5D  F0 4C                 BEQ CLOSE
0F5F                ST1
0F5F  88                    DEY
0F60  20 A0 0F              JSR UPPNTR
0F63  20 AC EB      EXIT    JSR PLXY
0F66  68                    PLA
0F67  38                    SEC
0F68  60                    RTS
0F69                INPUT
0F69  48                    PHA
0F6A  20 9E EB              JSR PHXY
0F6D  A0 01                 LDY #1
0F6F  B1 C6                 LDA (PNTR),Y
0F71  C9 23                 CMP #'#
0F73  D0 D4                 BNE PR1
0F75  A9 54                 LDA #'T
0F77  8D 12 A4              STA INFLG
0F7A  C8                    INY
0F7B  B1 C6                 LDA (PNTR),Y
0F7D  C9 27                 CMP #'''
0F7F  F0 07                 BEQ LOADFL
0F81  C9 23                 CMP #'#
0F83  F0 2F                 BEQ OFFTAP
0F85  4C 5F 0F              JMP ST1
0F88                LOADFL
0F88  20 C7 0F              JSR RDNAME
0F8B  8C 34 A4              STY TAPIN
0F8E  20 2F E3              JSR LOADTA
0F91  4C 63 0F              JMP EXIT
0F94                OPENFL
0F94  20 C7 0F              JSR RDNAME
0F97  8C 35 A4              STY TAPOUT
0F9A  20 6F E5              JSR DUMPTA
0F9D  4C 63 0F              JMP EXIT
0FA0  98            UPPNTR  TYA
0FA1  18                    CLC
0FA2  65 C6                 ADC PNTR
0FA4  85 C6                 STA PNTR
0FA6  90 02                 BCC UP1
0FA8  E6 C7                 INC PNTR+1
0FAA  60            UP1     RTS
0FAB                CLOSE
0FAB  20 F0 E9              JSR CRLF
0FAE  20 F0 E9              JSR CRLF
0FB1  20 0A E5              JSR DU11
0FB4                OFFTAP
0FB4  A9 CF                 LDA #$CF
0FB6  2D 00 A8              AND DRB
0FB9  8D 00 A8              STA DRB
0FBC  20 FE E8              JSR LL
0FBF  20 AC EB              JSR PLXY
0FC2  68                    PLA
0FC3  A9 8E                 LDA #$8E
0FC5  38                    SEC
0FC6  60                    RTS
0FC7                RDNAME
0FC7  C8                    INY
```

```
0FC8   20 A0 0F            JSR UPPNTR
0FCB   A0 00               LDY #0
0FCD   B1 C6       NEXT    LDA (PNTR),Y
0FCF   C9 27               CMP #'''
0FD1   F0 0E               BEQ ENDNAM
0FD3   99 2E A4            STA NAME,Y
0FD6   C8                  INY
0FD7   C0 05               CPY #5
0FD9   D0 F2               BNE NEXT
0FDB   20 A0 0F            JSR UPPNTR
0FDE   4C EE 0F            JMP RD1
0FE1   20 A0 0F    ENDNAM  JSR UPPNTR
0FE4   A9 20               LDA #'
0FE6   99 2E A4    EN1     STA NAME,Y
0FE9   C8                  INY
0FEA   C0 05               CPY #5
0FEC   D0 F8               BNE EN1
0FEE               RD1
0FEE   A0 01               LDY #1
0FF0   B1 C6               LDA (PNTR),Y
0FF2   C9 32               CMP #'2
0FF4   F0 AA               BEQ UPPNTR
0FF6   C9 31               CMP #'1
0FF8   D0 03               BNE RD2
0FFA   20 A0 0F            JSR UPPNTR
0FFD   88          RD2     DEY
0FFE   60                  RTS
0FFF                       *=$C8
00C8               DIVERT
00C8   4C 2D 0F            JMP BASEXT
00CB   EA                  NOP

00CC                       .END
```

As a final note, the BASIC data files are EDITOR compatible so that data to be processed can be produced by using the EDITOR.

AN EXAMPLE PROGRAM ILLUSTRATING THE USE OF THE NEW COMMANDS

Notes: No tape number was specified when opening the files thus tape recorder 1 is used (default)

At 600 is a subroutine to toggle the tapes to make rewind and fast forward possible.

SOME COMMENTS ON THE EXAMPLE BASIC PROGRAM:

Line Number	Action
45	turn tape #1 ON
55	wait for key when operator is ready
58	turn both tapes OFF
60	the output file is opened and called "NAMES"
100	.LAST indicates that the last name has been entered
140	end of output to TAPE routine
200	start of input from TAPE routine
220	looks for file with NAME= "NAMES"
230	prints heading (1st string in file)
260	inputs name from TAPE
270	has last been read?
280	echos to printer
300	closes file
600	TP=0 (both tapes OFF) TP=1 (#1 ON, #2 OFF) TP=2 (#1 OFF, #2 ON) TP=3 (both tapes ON)

```
10 PRINT!" EXAMPLE PROGRAM"
30 PRINT!" "
40 REM STORE NAMES   ON CASSETTE
45 TP=1:GOSUB600
50 PRINT" TAPE TO RECORD"
55 GETA$:IF A$=""  THEN55
58 TP=0:GOSUB600
60 PRINT#'NAMES'"NAME LIST"
70 FOR I=1TO30
80 INPUTA$
90 PRINT#A$ :  REM  # SO TO TAPE
100 IF A$=".LAST"THEN120
110 NEXT
120 REM CLOSE FILE
130 PRINT##
140 END
200 REM READ NAMES   FROM TAPE
210 PRINT"TAPE TO PLAY"
220 INPUT#'NAMES'H$
230 PRINT!TAB(5);H$
240 PRINT!" "
250 FOR I=1TO30
260 INPUT#A$
270 IFA$=".LAST"THEN300
280 PRINT!A$
290 NEXT
300 INPUT##
310 PRINT" D O N E ! !"
320 END
590 REM TAPE ON/OFF
600 POKE43008,207ANDPEEK(43008)OR16*TP
610 RETURN
```

A MOVE/RELOCATE ROUTINE

Anthony Chandler,
Montreal, Canada

SUMMARY

This routine will, at the user's option, either MOVE a block of data or RELOCATE a machine-language program from one area of memory into any other area of RAM from $0200 up. It can perform both forward and backward shifts, and resides entirely in Page Zero.

INTRODUCTION

Often the need arises to shift a block of data or a machine-language program from one set of locations in memory to another.

If a block of data, such as a "look-up" table has to be shifted, then a simple MOVE routine which sequentially reads each byte of data in the SOURCE area and writes it into the DESTINATION area is sufficient. Examples of MOVE routines are given on pages 6-26 and 6-27 of the R6500 Programming Manual.

However, if a machine-language program has to be shifted, then a simple MOVE routine may not be satisfactory. Those instructions in the program which use the absolute addressing mode (such as JMP 0345 or LDA 0567) have operands in the form of an address. If the operand points to an address within the span of the program being re-located, then the instruction must be modified so that its operand points to the corresponding address in the destination area. On the other hand, if the instruction refers to an address outside the span of the program, then it must be moved without alteration.

In order to shift programs, a more complex routine which calculates the necessary address changes is required.

In AIM 65, the memory area available for programs extends from address $0200 up to the limit of installed RAM ($1000 if 4K of memory is installed). Any MOVE/RELOCATE routine which occupies part of this area will naturally be restrictive, since the area it took up could not be used. A special effort has been made to enable the following routine to be located entirely in Page zero, which is not normally used for program instructions, so as to leave the entire working area from $0200 up free.

DESCRIPTION

Fig. 1 is a disassembly of the MOVE/RELOCATE routine. The program itself occupies addresses $0000–$00DD. Addresses $00EB–$00FF are "borrowed" from the Text Editor "Find" command for temporary storage, pointers and prompt messages. Loading of the "RELOC" routine will not disturb any operations of the Text Editor except the "Find" command and only then if an attempt is made to find a character string longer than 12 characters. The Text buffer addresses, stored in $00DF–$00E9 are preserved.

EXECUTION—RELOCATE

The program starts at $0000 and can be run using the * =0000 command or by setting up a linkage to $0000 via one of the Function keys. The following example illustrates the entries necessary to re-locate a program presently residing at addresses $0456 to $0567 to a destination starting at address $0234. In this example, the *address* of the last instruction is $0567—the last byte of the program might be at $0569, if the program terminated with a 3 byte instruction.

PROGRAM PROMPTS

S = START ADDRESS
F = FINISH ADDRESS
D = DESTINATION ADDRESS
MR = MOVE/RELOCATE

```
* =0000
G/
S=                          Enter 0456 (NOTE—NO ERRORS
                                        PERMITTED. IF
                                        INCORRECT DIGIT
                                        THEN RE-START
                                        PROGRAM)

S=0456F=                    Enter 0567
S=0456F=0567D=              Enter 0234
(Display wraps around)
0456F=0567D=0234MR=         Enter "R" (for re-locate)
                             (any other key except "M" will
                             do)
```

The routine will run, displaying a disassembly of the source program as the re-location takes place.

On completion, control returns to the Monitor. The next free available address following the re-located program ($0348 in the above example) will be found by examining memory locations 00F5-00F6 (LSB first—4803)

EXECUTION—MOVE

If the source addresses, $0456 to $0567 contain data (or text) then a similar procedure is followed.

In this case, however, the Source Finish address entered in response to the prompt "F= " should be one address less than that of the last byte of data (for example, 0566 instead of 0567).

After entering the addresses, the response to the move/relocate prompt "MR= " should be "M" for move.

The Destination Finish address to be found at $00F5-00F6 will be the address of the last byte of data moved (for example $0345). The next free address is $0346.

If the MOVE routine is used to shift the contents of the Editor's Text Buffer, then the Source Start address should be that shown (Low order byte first) at $00E3-00E4. The Source Finish address should be one less than the text end address shown at $00E1/E2. On completion of the MOVE operation, it will be necessary to reset the Text Buffer addresses as follows:

00E1	Text end address—same as 00F5
00E2	00F6
00E3	Text start address—same as Destination
00E4	Start
00E5	Text buffer end address—this can be any
00E6	address higher than that in 00E1-00E2 depending on the amount of free space required.

During execution of the MOVE option, no messages are displayed and return to the Monitor is very rapid.

OVERLAPPING

The routine permits backward overlapping—for programs, the DESTINATION START address must be at least three addresses lower than the SOURCE START. For a data MOVE, there is no restriction.

Forward overlapping is not possible, but a program or data block can be temporarily re-located or moved to a high or low memory area and then shifted back to overlay its original source area.

SELF-REPRODUCTION

Incidentally, the program will successfully re-locate itself and so, if the terminating instruction were replaced with instructions calculating a new destination, it could become self-perpetuating until its progeny filled available RAM.

STORING ON CASSETTE TAPE

When dumping the routine for storage on to cassette tape, the addresses to dump are FROM= 0000 TO= 00DD
MORE? Y
FROM= 00F7 TO= 00FF

This procedure avoids recording on tape the Editor's Text start and finish addresses from $00E1 to $00E6. This means that, when "RELOC" is loaded from tape at some future time, it will not affect any Text Editor which is set up.

PROGRAM LISTING AND COMMENTS

The following temporary stores and pointers are used:

SOURCE START (S)	$00EB	(LO)
	00EC	(HI)
CURRENT SOURCE ADDRESS	00ED	
	00EE	
SOURCE FINISH (F)	00EF	
	00F0	
OPERAND ADDRESS (from instruction being read)	00F1	
	00F2	
DESTINATION START (D)	00F3	
	00F4	
CURRENT DESTINATION ADDRESS	00F5	
	00F6	

Prompt messages are stored (in ASCII) as follows:

M = 00F7	/ 53	3D	46	3D	S = F =	
00FB	/ 44	3D	4D	52	D = M R	
00FF	/ 3D	*	*	*	=	(* = unchanged)

0000	A2	LDX	#00	INITIALIZE. X INDEXES MESSAGE BYTES
0002	A0	LDY	#00	Y INDEXES PROGRAM BYTES EACH INSTRUCTION
0004	20	JSR	00D2	DISPLAY PROMPT MESSAGE ASKING FOR ADDRESS
0007	20	JSR	0090	GET 4-DIGIT ADDRESS AND STORE IT
000A	E0	CPX	#0C	SEE IF 12 DIGITS (ALL THREE ADDRESSES)
000C	D0	BNE	0004	IF NOT-BACK FOR NEXT ADDRESS
000E	20	JSR	00D2	DISPLAY FINAL PROMPT ("MR=")

```
0011  20   JSR  E973     REDOUT—SEE IF USER
                         WANTS MOVE OR
                         RELOCATE
0014  C9   CMP  #4D      IF HE SAYS "M" THEN—
0016  F0   BEQ  007E     GO TO MOVE ROUTINE FOR
                         STRAIGHT COPY
0018  A5   LDA  ED       OTHERWISE, GET CURRENT
001A  8D   STA  A425     SOURCE ADDRESS FROM ED/
001D  A5   LDA  EE       EE AND PUT IT IN SAVPC AT
                         A425/A426
001F  8D   STA  A426
0022  20   JSR  F46C     DISASM—INTERPRET
                         INSTRUCTION & DISPLAY IT
0025  A5   LDA  EA       LENGTH—ACCUMULATOR
                         HAS LENGTH MINUS ONE
0027  C9   CMP  #02      IS IT A 3-BYTE
                         INSTRUCTION?
0029  D0   BNE  006E     NO—SO GO MAKE
                         STRAIGHT COPY
002B  A0   LDY  #01      YES—IS A 3-BYTE SO MAY
                         HAVE TO ALTER
002D  B1   LDA  (ED),Y   GET FIRST BYT OF OPERAND
002F  85   STA  F1
0031  C8   INY
0032  B1   LDA  (ED),Y   SECOND BYT OF OPERAND
0034  85   STA  F2       OPERAND INTO F1/F2
0036  38   SEC           SUBTRACT SOURCE START
0037  A5   LDA  F1       ADDRESS FROM OPERAND
0039  E5   SBC  EB       TO SEE IF OPERAND POINTS
                         TO ADDRESS BELOW
                         SOURCE START
003B  A5   LDA  F2
003D  E5   SBC  EC
003F  90   BCC  006E     IF SO—CARRY CLEAR AND
                         NO CHANGE REQUIRED
0041  A5   LDA  EF       SUBTRACT OPERAND FROM
0043  E5   SBC  F1       SOURCE FINISH ADDRESS
0045  A5   LDA  F0       TO SEE IF OPERAND POINTS
                         TO ADDRESS ABOVE
                         SOURCE FINISH
0047  E5   SBC  F2
0049  90   BCC  006E     IF SO—CARRY CLEAR AND
004B  18   CLC           NO CHANGE REQUIRED.
004C  A5   LDA  F1       OPERAND REQUIRES
                         CHANGING SO PREPARE TO
                         ADD. ADD OPERAND TO
                         DESTINATION START
                         ADDRESS
004E  65   ADC  F3
0050  48   PHA           TEMPORARILY STORE LO-
                         BYT SUM ON STACK
```

```
0051  A5   LDA  F2
0053  65   ADC  F4
0055  AA   TAX           TEMPORARILY STORE HI-
                         BYT SUM IN X
0056  38   SEC           NOW SUBTRACT SOURCE
                         START ADDRESS FROM SUM
0057  68   PLA           GET LO-BYT SUM
0058  E5   SBC  EB
005A  48   PHA           STORE IT ON STACK
005B  8A   TXA           GET HI-BYT SUM FROM X
005C  E5   SBC  EC
005E  A0   LDY  #02
0060  91   STA  (F5),Y   PUT ADJUSTED OPERAND
0062  88   DEY           INTO CURRENT
                         DESTINATION PLUS 3
0063  68   PLA
0064  91   STA  (F5),Y   AND PLUS 2
0066  88   DEY
0067  B1   LDA  (ED),Y   NOW GET OP-CODE FROM
                         CURRENT SOURCE
0069  91   STA  (F5),Y   PUT IT IN CURRENT
                         DESTINATION
006B  4C   JMP  0071     GO TO UPDATE AND END
                         CHECK
006E  20   JSR  00C6     MAKE STRAIGHT COPY OF
                         COMPLETE INSTRUCTION
0071  20   JSR  00AD     INCREMENT CURRENT
                         SOURCE AND DESTINATION
                         ADDRESSES BY LENGTH OF
                         INSTRUCTION PLUS ONE
0074  20   JSR  EA13     CLEAR THE DISPLAY
                         (CRLOW)
0077  20   JSR  00A3     SEE IF PAST END—CARRY
                         CLEAR IF SO
007A  B0   BCS  0018     NOT AT END SO GO BACK
                         FOR NEXT INSTRUCTION
007C  90   BCC  008D     BRANCH ALWAYS (AT END)

007E  THE FOLLOWING ROUTINE IS JUMPED TO IF USER
      REQUIRES A MOVE OPERATION RATHER THAN
      RELOCATE. IT TRANSFERS A STRAIGHT COPY, BYTE
      BY BYTE FROM SOURCE INTO DESTINATION

007E  A9   LDA  #01      SET LENGTH TO ONE
0080  85   STA  EA
0082  20   JSR  00C6     TRANSFER THE DATA
0085  20   JSR  00AF     INCREMENT CURRENT
                         SOURCE AND DESTINATION
                         ADDRESSES BY ONE
0088  20   JSR  00A3     SEE IF PAST END—CARRY
                         CLEAR IF SO
```

```
008B  B0  BCS  007E      NOT AT END SO BACK FOR
                         NEXT BYT OF DATA
008D  4C  JMP  FEE9      PATC10—CLEAR DISPLAY
                                 —HOME TO
                                  MONITOR
                                 —REVELATION 6.14
```

0090 THIS SUB-ROUTINE GETS A 4-DIGIT ADDRESS AND STORES IT, LO-BYT FIRST, IN TWO ADJACENT PAIRS OF THE STORE STARTING AT $00EB. WHEN CALLED FOR THE FIRST TIME, X = 0

```
0090  20  JSR  E3FD      RBYTE—GET TWO DIGITS
                         (HI ORDER)
0093  95  STA  EC,X      STORE THEIR HEX VALUE
0095  95  STA  EE,X      SAME AGAIN
0097  20  JSR  E3FD      RBYTE—GET NEXT TWO
                         DIGITS (LO ORDER)
009A  95  STA  EB,X      STORE
009C  95  STA  ED,X      AGAIN
009E  E8  INX            INCREMENT X READY FOR
                         NEXT ADDRESS
009F  E8  INX
00A0  E8  INX
00A1  E8  INX
00A2  60  RTS
```

00A3 THIS SUB-ROUTINE CHECKS TO SEE IF THE CURRENT SOURCE ADDRESS HAS EXCEEDED THE SOURCE FINISH ADDRESS—IF SO, THE MOVE OR RELOCATE IS COMPLETE.

```
00A3  38  SEC
00A4  A5  LDA  EF
00A6  E5  SBC  ED
00A8  A5  LDA  FO
00AA  E5  SBC  EE
00AC  60  RTS            IF NOT PAST END, CARRY
                         REMAINS SET
```

00AD THIS SUB-ROUTINE INCREMENTS THE CURRENT SOURCE AND CURRENT DESTINATION STORES BY AN AMOUNT EQUAL TO THE LENGTH OF THE LAST-INTERPRETED INSTRUCTION PLUS ONE, SO AS TO POINT TO THE NEXT INSTRUCTION TO BE READ

IF DATA IS BEING MOVED, THE LENGTH (IN $00EA) IS SET TO #01 AND THIS SUB IS ENTERED AT $00AF SO THAT SOURCE AND DESTINATION ADDRESSES ARE INCREMENTED BY ONE EACH TIME

```
00AD  E6  INC  EA        ADD ONE TO LENGTH
00AF  18  CLC
00B0  A5  LDA  EA
00B2  65  ADC  ED
00B4  85  STA  ED
00B6  90  BCC  00BA
00B8  E6  INC  EE
00BA  18  CLC
00BB  A5  LDA  EA
00BD  65  ADC  F5
00BF  85  STA  F5
00C1  90  BCC  00C5
00C3  E6  INC  F6
00C5  60  RTS
```

00C6 THIS SUB-ROUTINE IS CALLED WHEN NO MODIFICATION OF THE OPERAND IS REQUIRED. IT COPIES A COMPLETE INSTRUCTION FROM THE ADDRESS POINTED TO BY CURRENT SOURCE, INTO THE ADDRESS POINTED TO BY CURRENT DESTINATION

```
00C6  A4  LDY  EA        GET LENGTH OF
                         INSTRUCTION
00C8  B1  LDA  (ED),Y    GET BYT FROM SOURCE
00CA  91  STA  (F5),Y    PUT IT IN DESTINATION
00CC  88  DEY
00CD  CO  CPY  #FF       ANY MORE ?
00CF  DO  BNE  00C8      YES—GO BACK FOR NEXT
                         BYTE
00D1  60  RTS
```

00D2 THIS SUB-ROUTINE DISPLAYS THE FOUR PROMPT MESSAGES WHICH ARE STORED IN ASCII AT $00F7 ET SEQ. WHEN CALLED FOR THE FIRST TIME, Y = 0 AND IS USED TO INDEX ALONG THE MESSAGE TABLE.

EACH MESSAGE ENDS WITH AN EQUALS SIGN, = (ASCII #3D), AND THIS IS USED TO DETERMINE THE END OF EACH PROMPT MESSAGE

```
00D2  B9  LDA  00F7,Y    GET THE CHARACTER
00D5  20  JSR  E97A      OUTPUT—DISPLAY THE
                         CHARACTER
00D8  C8  INY            READY FOR NEXT
                         CHARACTER
00D9  C9  CMP  #3D       IS IT "=" ?
00DB  D0  BNE  00D2      NO—SO GET ANOTHER
                         CHARACTER
00DD  60  RTS
```

TTY OUTPUT UTILITY PROGRAMS

Mark Reardon
Rockwell International

Many peripheral devices (printers, CRT Monitors) can use inputs in the form of a 20 ma current loop or RS-232. The AIM 65 has a built-in 20 ma current loop that can be utilized, or the loop can be modified to being an RS-232 (DOC. No. 230: RS-232C Interface for AIM 65).

One large problem still remains. For the AIM 65 Firmware to use the TTY port, the Keyboard/TTY switch must be in the TTY position. Unfortunately, the AIM 65 then uses the TTY port for all of the inputs that usually come from its Keyboard. Most printers have no way of communicating back to the AIM 65. In order for the keyboard to retain control, one of the following programs can be used. Each uses the TTY subroutine in the AIM 65 Monitor (OUTTTY=$EEA8). They also require the user to enter the correct values for the baud rate in locations $A417 and $A418. The first program (ECHO) utilizes the DILINK ($A406) vector to intercept all data on the way to the display/printer and then redirects it to both the TTY and display/printer. If this program or any other program that modifies DILINK is assembled on the AIM 65 the object code has to be directed to an external device.

If the object code is directed to memory, the AIM 65 will lock up. To free it, the power has to be turned off. Reset will not correct the problem. The second program (UOUT) is a user output program. It allows the user to select the TTY port by responding to the OUT= prompt with a U.

In this way any command that uses the Outall subroutine will direct its output to the TTY port. AIM 65 Basic uses Outall for all of its printing commands. Unfortunately, AIM 65 Basic also sets the Outflag to equal P. To use the user output program the instruction: "POKE 42003,85," needs to be inserted.

In actual use there have been two major sources of failure with these programs. The easiest to cure is if the baud rate isn't entered properly. To determine the appropriate values do the calculations as shown below. The second source of trouble has been that different manufacturers have designed their peripheral requiring different inputs than are provided. In these situations these two programs had to be modified to satisfy the peripheral's needs.

ECHO PROGRAM

```
0000                    OUTTTY=$EEA8
0000                    CR=$0D
0000                    LF=$0A
0000                    NULL=$FF
0000                    DILINK=$A406
0000                    *=DILINK
A406    00 02                   .WOR ECHO     :SET VECTOR TO THIS ROUTINE
A408                            *=$200
0200    C9 0D       ECHO        CMP #CR       :CR?
0202    D0 0A                   BNE NOTCR     :No, JUST OUTPUT IT
0204    20 A8 EE                JSR OUTTTY    :YES, ADD LF AND NULL
0207    A9 0A                   LDA # LF
0209    20 A8 EE                JSR OUTTTY
0200    A9 FF                   LDA #NULL
020E    4C A8 EE    NOTCR       JMP OUTTTY    :OUTPUT AND RTS
0211                            .END
```

UOUT PROGRAM

```
0000                    OUTTTY=$EEA8
0000                    CR=$0D
0000                    LF=$0A
0000                    NULL=$FF
0000                    UOUT=$10A
0000                            *=UOUT
010A    00 02                   .WOR START    :VECTOR TO PROGRAM
010C                            *=$200
0200    90 12       START       BCC RETRN     :NO SETUP
0202    68                      PLA           :A ON STACK
0203    C9 0D                   CMP #CR       :IF CR ALSO SEND
0205    D0 0A                   BNE NOTCR     :A LF AND NULL
0207    20 A8 EE                JSR OUTTTY    :OUTTTY ALSO SENDS
020A    A9 0A                   LDA #LF       :TO DISPLAY/PRINTER
0200    20 A8 EE                JSR OUTTTY
020F    A9 FF                   LDA #NULL
0211    4C A8 11    NOTCR       JMP OUTTTY
0214    60          RETRN       RTS
0215                            .END
```

METHOD TO CALCULATE BAUD RATES FOR THE AIM 65

When used with terminals running at 1200 baud and up, the Rockwell AIM 65 needs to have the Baud Rate entered manually. To calculate the values to enter perform the procedure outlined below:

Note: All variables are integers and have us/bit as their units.

1. 10^6/(Baud Rate) = X
2. X-67 us/b = Y
3. Y/256 = Z remainder W
4. $A417 = Z in Hex
5. $A418 = W in Hex

Examples: Baud Rate 4800

1. 10^6/4800 Baud = 208
2. 208−67 us/b = 141
3. 141/256 = 0 Remainder 141
4. \$A417 = 0_{10} = 00_{16}
5. \$A418 = 141_{10} = $8D_{16}$

Baud Rate 150

1. 10^6/150 Baud = 6667
2. 6667−67 us/b = 6600
3. 6660/256 = 25 Remainder 200
4. \$A147 = 25_{10} = 19_{16}
5. \$A418 = 200_{10} = $C8_{16}$

DATA STATEMENT GENERATOR

G. Brinkmann
W. Germany

Remember the last time you had to convert a machine language program to data statements so your Basic program could poke it into RAM somewhere? I'll bet you really enjoyed having to convert each hex byte into decimal and then typing it in. No? Well, then maybe you'll find this program will come in handy next time around.

What it does is convert hex data to decimal and generate BASIC data statements with the decimal data. The statements that it generates are sent out to the audio cassette interface which is used as temporary storage. The input is in the form of hex numbers which could come from the conversion program itself, as is in the example or, from memory with a minor change to the conversion program.

Note that this approach needs only one tape without remote control and only "on board" assembly language routines. The following example converts the first 26 HEX-values of R. Reccia's program (INTERACTIVE 1) into BASIC-DATA-Statements and writes them to tape.

It works as following:

—the HEX-values of the assembler language program are put into the BASIC-Program by DATA-statements. They must be ended by an "END" DATA (or any other special mark, see lines 90, 260).

—In line 190 you are asked for the line-number of the first DATA-statement to be generated, depending on your BASIC-program.

—Line 210 performs a call to WHEREO and opens the outfile. If it is a tape, with a gap of 80 (POKE 41993,128).

—The main loop starts at line 230, the STRING S$ is filled with the statement-number and the constant "DATA".

—In line 260 we read the HEX-input-data until "END". The data is added to S$ after converting to decimal in a subroutine. Each DATA-line takes 10 items.

—The PRINT-statements (line 350) write the STRING S$ to any open output, adds 1 to the statement-number and goes to the start of the main loop (line 230). Note that until now the first statement-line has a linenumber of d+1 (where d was your input).

—If the END-mark has been read, the last DATA-statement will be printed, followed by the statement-line "d" with a counter of all DATA-items.

—The file will be closed in line 410 through a jump to B52B, a BASIC-routine which prints a CTRL/Z, closes the file and waits for the new input.

—The HEX to DECIMAL conversion takes place in statement 450–560 and uses the STRING H$ in 170. Leading zeroes in the HEX-numbers are not needed.

—If an error occurs, the faulty item will be printed to the printer and the file is closed. Therefore, you should make a trial run before going to tape (by hitting RETURN after OUT=) and any error will go to the printer (which has not to be on).

When everything worked ok until now, you have a file with DATA-statements on tape. To read it into your actual program, just use a statement as

```
100READ N:FOR I = 0 TO N-1:READ X:POKE xxxx+I,X:NEXT
```

Remember, the first DATA-statement contains a counter of the following DATA-items. So you don't have to bother about it, the first READ will get it for you. This is extremely useful during the test phase, where changes occur quite frequently.

The next step is to load the statements into your BASIC program with the LOAD command. Be sure that you have chosen the right line-number, the LOAD command will over-write duplicate line-numbers. However, while testing, it might save you deleting the old lines.

If you are working with the ASSEMBLER and the BASIC at the same time, you could change the READ in line 260 to PEEK's. This saves you the initial typing in of DATA-statements and the conversion will be done by BASIC. However, you should either use a counter or a unique mark as 0,0,0 to find an end to the data.

Of course, the data need not to be in memory at all. You can generate DATA-statements by reading from keyboard or by using your BASIC-program to compute them from other data. I use this program regularly while computing moving averages and other statistics and then replacing the old values by the new ones for the next run.

```
70 DATAA9,B7,8D,2,A8,20,10,F2,A?,23,20,4A,F2
80 DATAA2,0,BD,69,0F,20,4A,F2,E8,C9,21,D0,F5
90 DATA END
100 REM HEX TO DECIMAL
110 REM GENERATES DATA-LINES ON TAPE-FILE
120 REM G. BRINKMANN
130 REM AUF'M GRAEVERICH 19A
140 REM D-5414 VALLENDAR
150 REM WEST GERMANY
160 REM INIT
170 H$="0123456789ABCDEF?"
180 REM FIRST LINE FOR COUNT OF DATA ITEMS
190 INPUT"NR OF FIRST DATA-LINE";D1:D=D1+1
200 REM OPEN TAPE-FILE WITH LONG GAP
210 POKE 4,113:POKE 5,232:POKE 41993,128
220 X=USR(0)
230 S$=STR$(D)+"DATA"
240 REM 10 ITEMS PER LINE
250 FOR N=1 TO 10
260 READ A$:IF A$="END" THEN 390
270 REM SUBROUTINE HEX -> DECIMAL
280 GOSUB 470
290 REM ON ERROR CLOSE FILE
300 IF A1$<>"ER"THEN 310
305 POKE 42003,13:PRINT!"ERROR IN LINE ";D:GOTO430
310 IFN>1 THEN S$=S$+","
320 REM STRING CONCATENATION
330 S$=S$+A1$:NEXT
360 REM OUTPUT TO ANY OPEN FILE; INC LINE NUMBER
370 PRINT S$:D=D+1:GOTO 230
380 REM PRINT LAST LINE AND THEN FIRST
390 PRINT S$
400 S$=STR$(D1)+"DATA"+STR$((D-D1-1)*10+N-1)
410 PRINT S$
420 REM CLOSE OUTPUT FILE
430 POKE 4,43:POKE 5,181:X=USR(0)
440 REM JUMP TO BASIC INPUT
450 END
460 REM SUBROUTINE HEX -> DECIMAL
470 IF LEN(A$)=1 THEN A$="0"+A$
480 FOR I=1 TO 17
490 IF MID$(A$,1,1)=MID$(H$,I,1) THEN A=16*(I-1):GOTO 520
500 REM AFTER LAST NEXT => ERROR
510 NEXT:GOTO 580
520 FOR I=1 TO 17
530 IF MID$(A$,2,1)=MID$(H$,I,1)THEN A=A+I-1:GOTO560
540 NEXT:GOTO 580
550 REM IT'S A GOOD ONE
560 A1$=STR$(A):RETURN
570 REM PRINT ERROR MSG
580 A1$="ER":RETURN
```

CASSETTE LOAD UTILITY
. . . For AIM 65

Mark Reardon
Rockwell International

This multi-purpose utility program allows you to load programs with offset and recover programs that have load errors.

For example, suppose you wish to reload a program to reside at $0500 that was originally dumped from $0200. First, start the program by pressing the 'F1' key. The 'FROM=' prompt should appear first. Enter 0200 to specify where the program used to reside in memory and press the 'RETURN' key. Answer the 'TO=' prompt with 0500 to show where the program is going to be loaded. (Programs can only be offset by even page amounts. For example, if a program originally resided at $0236, it could only be offset to $0436, $0636, $0A36 etc. not $0400, $0777, or $0100. Get it? This is because the offset calculation is done only on the page number (upper byte) and not the byte number (lower byte).)

The rest of the cassette load prompts are the same as the normal ones in the standard cassette load routine.

This program will also let you load a program even though there are loading errors. This, at least, gives you a chance to recover a program that would otherwise be impossible to recover. The normal cassette load routines will stop when an error occurs.

```
2000                  NAME    =$A42E
2000                  CKSUM   =$A41E
2000                  TAPAR   =$A436
2000                  ADDR    =$A41C
2000                  S1      =$A41A
2000                  TEMP    =$0117
2000                  ;
2000                  TAISET  =$EDEA
2000                  GETTAP  =$EE29
2000                  PLXY    =$EBAC
2000                  PHXY    =$EB9E
2000                  NAMO    =$E8CF
2000                  OUTALL  =$E9BC
2000                  SADDR   =$EB78
2000                  COMIN   =$E1A1
2000                  FROM    =$E7A3
2000                  TO      =$E7A7
2000                  ADDRS1  =$F910
2000                  CRLOW   =$EA13
2000                  BLANK   =$E83E
2000                  CHEKA   =$E54E
2000                  NXTADD  =$E2CD
2000                  NUMA    =$EA46
2000                  CLRCK   =$EB4D

2000                          *=$10C            ;SET UP F1 KEY
010C
010C  4C 61 00                JMP START

010F                          *=$00
0000  00              ERRO    .BYT $00
0001  45 52           MSG     .BYT 'ERRORS IN '
000B  4C 4F           MSG1    .BYT 'LOADIN',$C7
0011  C7
0012  44 4F 4E        MSG2    .BYT 'DON',$CE
0015  CE
```

```
0016  20 9E EB    TAPE    JSR PHXY
0019  20 EA ED    READ    JSR TAISET          ;SET UP TAPE
001C  20 29 EE    SYNC    JSR GETTAP          ;GET A CHAR
001F  C9 23               CMP #'#             ;BLOCK START
0021  F0 06               BEQ FOUND
0023  C9 16               CMP #$16            ;SYN?
0025  D0 F2               BNE READ
0027  F0 F3               BEQ SYNC
0029  A2 00       FOUND   LDX #0              ;STORE IN BUFFER
002B  20 29 EE    MORE    JSR GETTAP          ;GET A CHAR
002E  9D 16 01            STA TEMP-1,X
0031  E8                  INX
0032  E0 52               CPX #$52            ;BUFF FULL
0034  D0 F5               BNE MORE            ;NO
0036  20 AC EB            JSR PLXY
0039  60                  RTS

003A  20 9E EB    COUNT   JSR PHXY
003D  AE 36 A4            LDX TAPAR           ;BUFF POINTER
0040  E0 4F               CPX #79             ;BUFF EMPTY
0042  D0 05               BNE TIBI            ;NO
0044  20 16 00            JSR TAPE            ;READ A BLOCK
0047  A2 00               LDX #00             ;RESET POINTER
0049  BD 17 01    TIBI    LDA TEMP,X          ;GET CHAR
004C  E8                  INX                 ;INC BUFF POINTER
004D  8E 36 A4            STX TAPAR           ;SAVE POINTER
0050  20 AC EB            JSR PLXY
0053  E0 00               CPX #00             ;X<>0 THEN ADD CKSUM
0055  F0 09               BEQ RET
0057  4C 4E E5            JMP CHEKA           ;ADD TO CKSUM

005A  A5 00       ERROR   LDA ERRO            ;0=NO ERRORS
005C  D0 02               BNE RET
005E  E6 00               INC ERRO            ;MAKE<>0
0060  60          RET     RTS

0061  20 A3 E7    START   JSR FROM            ;ORIG ADDR
0064  20 3E E8            JSR BLANK           ;LEAVE A SPACE
0067  20 10 F9            JSR ADDRS1          ;ADDR TO S1
006A  20 A7 E7            JSR TO              ;NEW ADDR
006D  38                  SEC
006E  AD 1D A4            LDA ADDR+1
0071  ED 1B A4            SBC S1+1
0074  8D 1B A4            STA S1+1            ;OFFSET VALUE
0077  20 13 EA            JSR CRLOW           ;CLEAR DISPLAY
007A  20 CF E8            JSR NAMO            ;FILE NAME
007D  20 16 00    BLOCK   JSR TAPE
0080  A2 05               LDX #5
0082  8E 36 A4            STX TAPAR
0085  AD 16 01            LDA TEMP-1          ;BLK NO
0088  D0 F3               BNE BLOCK           ;NOT BLK 0
008A  BD 16 01    AGAIN   LDA TEMP-1,X
008D  DD 2D A4            CMP NAME-1,X        ;CMP NAMES
0090  D0 EB               BNE BLOCK           ;DIFFERENT
```

```
0092  CA                  DEX
0093  D0 F5               BNE AGAIN
0095  A2 0A               LDX #MSG1-MSG
0097  20 F2 00            JSR OUT           ;DISPLAY LOADING
009A  20 3A 00   GETCH    JSR COUNT         ;GET A CHAR
009D  C9 3B               CMP #'          ; ;RECORD START
009F  D0 F9               BNE GETCH
00A1  20 4D EB            JSR CLRCK         ;CLEAR CKSUM
00A4  E8                  INX
00A5  20 3A 00            JSR COUNT         ;RECORD LENGTH
00A8  AA                  TAX
00A9  F0 39               BEQ STOP          ;0=DONE
00AB  20 3A 00            JSR COUNT
00AE  18                  CLC
00AF  6D 1B A4            ADC S1+1          ;ADD OFFSET
00B2  8D 1D A4            STA ADDR+1
00B5  20 3A 00            JSR COUNT
00B8  8D 1C A4            STA ADDR
00BB  20 3A 00   LOAD2    JSR COUNT         ;GET DATA AND STORE
00BE  A0 00               LDY #0
00C0  20 78 EB            JSR SADDR         ;STORE AND CMP
00C3                      ;TO ELIMINATE MEMORY FAIL ERRORS
00C3                      ;REMOVE 'BEQ OK' AND 'JSR ERROR'
00C3  F0 03               BEQ OK            ;DID MEM ACCEPT?
00C5  20 5A 00            JSR ERROR
00C8  C8         OK       INY               ;Y=1
00C9  20 CD E2            JSR NXTADD        ;ADD Y TO ADDR
00CC  CA                  DEX               ;COUNT BYTES
00CD  D0 EC               BNE LOAD2
00CF  20 3A 00            JSR COUNT
00D2  CD 1F A4            CMP CKSUM+1
00D5  D0 08               BNE ERR
00D7  20 3A 00            JSR COUNT
00DA  CD 1E A4            CMP CKSUM
00DD  F0 BB               BEQ GETCH         ;CKSUMS OK
00DF  20 5A 00   ERR      JSR ERROR
00E2  D0 B6               BNE GETCH

00E4  20 13 EA   STOP     JSR CRLOW
00E7  A2 00               LDX #00
00E9  A5 00               LDA ERRO          ;0 IF NO ERRORS
00EB  86 00               STX ERRO
00ED  F0 01               BEQ NOE
00EF  2C                  .BYT $2C          ;CODE FOR BIT ABS
00F0  A2 11      NOE      LDX #MSG2-MSG     ;FINAL MSG AND RTS
00F2  B5 01      OUT      LDA MSG,X
00F4  48                  PHA
00F5  20 BC E9            JSR OUTALL
00F8  E8                  INX
00F9  68                  PLA
00FA  10 F6               BPL OUT           ;MSB=1
00FC  60                  RTS
00FD                      .END
```

INTERRUPT-DRIVEN KEYBOARD FOR THE AIM 65

Dr. Will Cronyn
Borrego Springs, CA

A common requirement in interactive computer systems is the entry of ASCII characters through the keyboard at random or erratic intervals when a program is executing. The program may be computational, process control, monitoring or some combination of these or other functions. The AIM 65 monitor routines require an explicit call to the keyboard and all (i.e. READ, RBYTE, etc.) except RCHEK demand a response before execution continues. The results would be disastrous if your AIM 65 controlled desert irrigation system had to wait 4 weeks before resuming execution for you to return from your summer vacation in Alaska to answer the question: Do you want the citrus put on a 3-days-a-week watering schedule? You could lace your program with calls to RCHEK but such calls, which consume 959 microseconds each (if there is no keyboard entry), can consume a large fraction of the execution time of the computer in spite of the fact that they are utilized for only a tiny fraction of the time.

One solution to the problem was described by De Jong in issue 3 of *Interactive*. He suggested the fundamental solution to the problem: generate interrupts for which the interrupt service routine looks for a keyboard entry. To allow continuation of program execution in the absence of a keyboard entry, De Jong modified AIM Monitor routines. The result is an interrupt routine which requires $A3 (163) bytes of code in 87 lines. In addition to the fairly lengthy code, it does not appear that his routines are fully debounced, i.e. debounced on both keystroke initiation and termination.

My solution is to use two interrupt service routines: one to jump from an executing main program to JSR READ, and the other to jump from READ (in the most likely event that no keyboard entry is available) back into the main program. Not only does this approach work but also it uses unmodified monitor routines and is instructive in its utilization of a dynamically programmed interrupt vector. The interrupt service routines require $40 (64) bytes of code in 29 lines.

DETAILED PROGRAM DESCRIPTION

There are three parts to the code which appears in the listing: (1) system configuration and initialization, $200-22B; (2) a "main" program which provides an immediate, positive verification that the interrupt-driven keyboard is functioning properly, $22C-24C; and (3) the interrupt routines themselves in a location which would be appropriate for most 4K AIM applications, $FCO-FFF. The interrupt routine sequences and configurations can best be understood by referring to the $\overline{IRQ}$ signal display. The T1 timer counter ($A004,5) is loaded with $FFFF, which produces an interrupt 65 milliseconds execution of the main program begins. The timer latch ($A006,7) is loaded with $4000. Thus, in the T1 free-run mode (UACR loaded with $40), when T1 times out after 65 milliseconds, which results in a jump to MNSVC, the contents of the T1 latch is transferred to the counter, thereby setting up another interrupt 16 milliseconds later. The interrupt vector is reconfigured to RDSVC and the T1 latch is loaded with $FFFF. Thus after 16 milliseconds in MNSVC the interrupt results in a jump to RDSVC, which returns program execution to the "main" program for another 65 milliseconds. Parameters for the next cycle are established by reconfiguring the interrupt vector to MNSVC and loading the T1 latch with $4000.

It may appear that 16 milliseconds is a long time to decide whether or not READ will actually be presented with a keyboard entry. However, because of timing requirements in READ which are based on the need to debounce key stroke and key release (a total of about 11 milliseconds) this time cannot be significantly reduced. In tests I performed, errors were evident at an allowance of $2800 microseconds, while none were seen at $2C00. I tested the program at keystroke rates up to about 540/minute (my maximum single-key stroking rate) with no sign of errors.

Note that the stack pointer is saved in SAVSP when MNSVC is entered. This procedure is required because normally, i.e. when there is no keyboard entry for READ, exit from READ is achieved through use of the interrupt rather than through an RTS within READ itself. Thus the stack is not properly restored and since there are 3 layers of subroutines within READ it would be unnecessarily difficult and risky to keep track of the depth of the stack when READ is exitted via interrupt.

The "main" program was a key element in testing and debugging the interrupt-driven keyboard. Through the display of "?" at the rate of about 3/second, with a carriage return/line feed after 10 "?", it provides an immediate indication that *both* the "main" program and the keyboard program are functioning. Of course a character entered through the keyboard would normally be placed in a buffer accessible to other parts of the program instead of simply being displayed via OUTPUT. The source code, even in its fully annotated form, is short enough that it, the Assembler symbol table, and the object code can all be co-resident in the AIM during development or modification.

```
2000                ;THIS PROGRAM ENABLES
2000                ;THE AIM-65 TO HAVE
2000                ;AN INTERRUPT-DRIVEN
2000                ;KEYBOARD,I.E.ENTRY
2000                ;WITHOUT EXPLICIT
2000                ;ENTRY CALLS.3 PARTS
2000                ;TO THIS CODE:1-IN-
2000                ;TERRUPT CONFIGURA-
2000                ;TION;2-DUMMY MAIN
2000                ;PROGRAM WHICH DIS-
2000                ;PLAYS 3"?"/SEC, 10
2000                ;"?"/LINE;3-INTER-
2000                ;RUPT SERVICE ROU-
2000                ;TINES.WRITTEN BY:
```

```
2000                ;DR.WILL CRONYN
2000                ;SYMBIOTIC DATA COMM
2000                ; P.O. BOX 626
2000                ;BORREGO SPRINGS,CA
2000                ;714-767-5498  92004
2000                ;9DEC1980.

2000                ;MONITOR ROUTINES.
2000                ;ALL EXCEPT "READ"
2000                ;ARE FOR DUMMY MAIN
2000                ;PROGRAM.
2000                NUMA    =$EA46
2000                CRLF    =$E9F0
2000                OUTPUT  =$E97A
2000                READ    =$E93C
2000                QM      =$E7D4

2000                ;IRQ VECT/T1 CONFIG.
2000                IRQV4   =$A400
2000                UACR    =$A00B
2000                UT1L    =$A004
2000                UT1LL   =$A006
2000                UIER    =$A00E
2000                ; PAGE 0 VARIABLES
2000                        *=$00
0000                CNTR    *=*+1
0001                ; MAIN ONLY.

0001                ; INTERRUPT CONFIG
0001                        *=$0200
0200
0200  A9 C1                 LDA #<MNSVC
0202  8D 00 A4              STA IRQV4
0205  A9 0F                 LDA #>MNSVC
0207  8D 01 A4              STA IRQV4+1
020A                ;T1 FREE-RUN MODE!
020A  A9 40                 LDA #$40
020C  8D 0B A0              STA UACR
020F                ;DISABLE ALL VIA
020F                ;INTRPTS EXCEPT T1
020F  A9 7F                 LDA #$7F
0211  8D 0E A0              STA UIER
0214  A9 C0                 LDA #$C0
0216  8D 0E A0              STA UIER
0219                ;INTRPT "MAIN" AFTER
0219                ; 65 MSEC=$FFFF USEC
0219  A9 FF                 LDA #$FF
021B  8D 04 A0              STA UT1L
021E  8D 05 A0              STA UT1L+1
0221                ;INTRPT READ AFTER
0221                ;16 MSEC=$4000 USEC.
0221  A9 00                 LDA #0
0223  8D 06 A0              STA UT1LL
0226  A9 40                 LDA #$40
0228  8D 07 A0              STA UT1LL+1
022B  58                    CLI

022C                ;START "MAIN" PROGRM
022C  A2 0A         BEGIN   LDX #10
022E                ; DONT HAVE INTRUPTS
022E                ;DURING PRINT OF "?"
022E  78            IDLE    SEI
022F  20 D4 E7              JSR QM
0232  58                    CLI
0233  20 3F 02              JSR DELAY
0236  CA                    DEX
0237                ; ARE WE UP TO 10?
0237  D0 F5                 BNE IDLE
0239  20 F0 E9              JSR CRLF
023C  4C 2C 02              JMP BEGIN
023F                ;FOR DELAY HAVE 2
023F                ;LOOPS-OUTSIDE=$80,
023F                ; INDEX=CNTR.
023F                ;INSIDE=$FF,INDEX=Y
023F  A0 FF         DELAY   LDY #$FF
0241  A9 80                 LDA #$80
0243  85 00                 STA CNTR
0245  88            LOOP1   DEY
0246  D0 FD                 BNE LOOP1
0248  C6 00                 DEC CNTR
024A  D0 F9                 BNE LOOP1
024C  60                    RTS

024D                ;INTRPT SRVC RTNS.
024D                ;MNSVC LEAPS FROM
024D                ;"MAIN"TO READ;RDSVC
024D                ;LEAPS FROM READ TO
024D                ;"MAIN".BECAUSE OF
024D                ;INTRPT-DRIVEN EXIT
024D                ;FROM READ,MUST SAVE
024D                ;STCK PNTR @ SAVSP.
024D                ;NEXT INTRPT AFTER
024D                ;MNSVC IS RDSVC & VV
024D                        *=$0FC0
0FC0                SAVSP   *=*+1
0FC1  48            MNSVC   PHA
0FC2  8A                    TXA
0FC3  48                    PHA
0FC4  BA                    TSX
0FC5  8E C0 0F              STX SAVSP
0FC8                ;SET INTRPT VECTOR
0FC8                ; FOR NEXT INTRPT
0FC8                ;CYCLE(NOT CURRENT)
0FC8  A9 E4                 LDA #<RDSVC
0FCA  8D 00 A4              STA IRQV4
0FCD  A9 0F                 LDA #>RDSVC
```

A BASIC HINT

Howard A. Chinn
S. Yarmouth, MA

Issue No. 1 of INTERACTIVE called attention to the use of the AIM 65 text editor for editing BASIC programs. Mention was not made, however, of the use of the text editor to write BASIC programs that contain both direct (calculator mode) and indirect (programming mode) commands. This feature (which is not available on a TRS-80 until you upgrade to a disc system) provides an opportunity for many interesting applications.

Listing No. 1 is that of a short demonstration program prepared in the text editor and printed using the *Editor's* "L" command. This program was recorded on tape using the *Editor's* "L" command. Next, BASIC is entered and the program loaded using *BASIC'S* "LOAD" and with the printer turned "OFF" (for this particular demonstration). Listing No. 2 was generated automatically *while the program was being loaded!*

Listing No. 2 shows that a title and explanation is printed without the distracting "REM"'s. Program lines 10 to 40 are then placed in RAM. Next, the POKE command turned the printer "ON". The list command did its thing just as if you had typed in the command using the keyboard. And, finally, the "RUN" command ran the program automatically and since the printer was still "ON" the result is shown on the printout. The program, of course, resides in RAM. It could have been made to disappear had the original listing contained "NEW" at its end.

In a nutshell, when using the AIM 65 text editor any entry without a line number becomes a direct command and those with line numbers are indirect commands that are placed in RAM in the usual fashion.

The possibilities of this feature of the AIM 65 are limited only by your imagination.

Now, can someone tell me how to write a BASIC program in the text editor including the essential "CTRL Z" and a command to automatically turn off the cassette recorder after a dump to tape?

(The "Z" at the end of Listing #1 is a control Z).

LISTING NO. 1

```
=(L)
/
OUT=
?!"BASIC PGM VIA EDITOR"
?!"================
=====""
?!"AUTOMATICALLY LISTS
AND RUNS PROGRAM"
?!"ALSO TURNS PRINTER ON
AUTOMATICALLY"
?!"FOR LIST AND RUN"
10 FOR N=1 TO 5
20?N"X15="N*15
30 NEXT N
40 END
POKE 42001, 128
LIST
RUN
Z
```

LISTING NO. 2

```
BASIC PGM VIA EDITOR
================
====
AUTOMATICALLY LISTS AND
  RUNS PROGRAM
ALSO TURNS PRINTER ON
  AUTOMATICALLY
FOR LIST AND RUN
LIST
  10 FOR N =1 TO 5
  20 PRINTN"X15="N*15
  30 NEXT N
  40 END  .
RUN
  1 X15= 15
  2 X15= 30
  3 X15= 45
  4 X15= 60
  5 X15= 75
```

```
0FCF  8D 01 A4          STA IRQV4+1
0FD2              ;LENGTH-NEXT INTRPT
0FD2              ; CYCLE=$FFFF USEC
0FD2  A9 FF             LDA #$FF
0FD4  8D 06 A0          STA UT1LL
0FD7  A9 FF             LDA #$FF
0FD9  8D 07 A0          STA UT1LL+1
0FDC  58                CLI
0FDD  20 3C E9          JSR READ
0FE0              ;DONT ALLOW INTRPT
0FE0              ; DURING OUTPUT
0FE0  78                SEI
0FE1  20 7A E9          JSR OUTPUT
0FE4              ;@ EXIT FRM MNSVC
0FE4              ;SET INTRPT FOR LEAP
0FE4              ; FROM "MAIN"
0FE4  A9 C1       RDSVC LDA #<MNSVC
0FE6  8D 00 A4          STA IRQV4
0FE9  A9 0F             LDA #>MNSVC
0FEB  8D 01 A4          STA IRQV4+1
0FEE              ;AT TERM OF THIS
0FEE              ;INTRPT CYCLE NEXT
0FEE              ;WILL HAVE 16 MSEC
0FEE  A9 00             LDA #0
0FF0  8D 06 A0          STA UT1LL
0FF3  A9 40             LDA #$40
0FF5  8D 07 A0          STA UT1LL+1
0FF8              ;NOW RESTORE A,X,SP
0FF8  AE C0 0F          LDX SAVSP
0FFB  9A                TXS
0FFC  68                PLA
0FFD  AA                TAX
0FFE  68                PLA
0FFF  40                RTI
1000                    .END
```

(Continued from page 2)

above the IRQ Interrupt Processing section of the program. Also change the instruction BNE INTRET in the IRQ Interrupt Processing section to read BEQ INTRET.

The disassembly listing will also have to be changed. Add a JMP 0388 instruction between the CLI and LDA #40 instructions. The BNE 0392 will then be changed to BEQ 0395 because that part of the program is shifted upwards in memory.

UNHELPFUL USR HELPER

For some unknown reason, the following program lines were omitted from the BASIC USR HELPER article on page 18 of issue #3.

The following lines are required:

```
0 DB=13*11+11:F=15:FA=15*16+10:GO TO 3
1 POKE4,DB:POKE5,F:RETURN:SET UP FOR SETARD
2 POKE4,FA:POKE5,F:RETURN:SET UP FOR CALLIT
3 REM PROGRAM MAY START HERE
```

Note that the definition on line 0 will speed up operation by eliminating the required conversions to decimal every time lines 1 or 2 are called.

NEWSLETTER REVIEW

From the Editor:
The Sept/Oct issue of the Target, a newsletter dedicated entirely to the AIM 65 was, perhaps, the best issue of that newsletter that I've seen. In it were two articles that should tickle the fancy of most any serious AIM 65 user. The first article showed how to hook up the new General Instrument Programmable Sound Generator (AY3-8910) to the Aim 65 and presented a software driver to make the thing generate telephone touch tones from phone numbers which are stored in memory.

I have played with this chip quite a bit and am really impressed with all its capability. The AY3-8910 interfaces very easily with the user R6522.

The other neat article that was in the issue presented complete plans (hardware and software) for an EPROM programmer that can program virtually all of the most popular EPROMS—2708, both styles of the 2716 and 2532. The software is self prompting and the hardware design is complete down to the AC power supply.

The Sept/Oct issue (1980) of Target is easily worth the $6.00 yearly subscription rate (it's published bimonthly). Outside of the U.S. and Canada the price is $12.00. Contact Donald Clem, RR#2, Spencerville, OH 45887.

BEHAVIORAL SCIENCES AIM-65 USERS GROUP

Workers in the behavioral and biological sciences who are currently using, or are interested in using the AIM 65 are invited to participate in a user's group now forming. Areas of interest include hardware and software for experimental control, data acquisition, statistical analyses, and other applications. If interested, please write, outlining areas of interest, current and planned projects, etc., to Dr. J. W. Moore, Jr., Box 539 MTSU, Murfreesboro, TN 37132.

LETTERS TO THE EDITOR

Dear Eric:

In a previous letter I complained about the lack of readability of many of the programs in issues #1 and #2 of INTERACTIVE. This letter is to thank you and commend you for the fine job you have done in issue #3 in rendering the programs more readable. The only one which is faint at all but still is quite readable is the simultaneous equations from George Sellers.

Here is a question you might be able to answer in the journal. Does anyone have a machine language program which will make a software conversion from ASCII to Baudot and output serial Baudot on the AIM 65's 20 miliampere current loop? A relay could then be used to transfer the Baudot to the 60 miliampere current loop of a Model 15 five level teletype. A perhaps related question—can the 20 miliampere TTY loop output of the AIM 65 be used to output to a printer and still use the AIM 65 keyboard? If so, where would the KBD/TTY switch be placed?

Another question—Since the AIM 65 monitor has routines in it which convert shifted characters so that the output is entirely capitals (no lower case) how can the AIM 65 board be used to feed a printer the necessary codes for lower case? I thought perhaps Dr. DeJong's program for the Interrupt Driven Keyboard on page 12 would answer this, but his routine contains at location ØC7F "if alpha characters do not shift" just as does the monitor. Could one just leave out the routine between ØC7F and ØC85 and get lower case characters output?

Keep plugging along and keep up the good work. Happy to see that INTERACTIVE is getting larger all the time. Thanks.

Sincerely,
John U. Keating, M.D.
8415 Washington Blvd.
Indianapolis, IN 46240

Dear John,

I don't know of any program available to convert the TTY port to Baudot. Doesn't sound too difficult, however. See the program on page 13 of this issue for the procedure for using the TTY port without regard to the TTY/KBD switch. I would assume that lower case output could be achieved by modifying an input program (such as DeJong's) and writing a new output program.

Eric

Dear Editor,

I must apologize. I am rather negligent in sending in programming "goodies" to share and this contribution does not make up for it. However, I noticed in Issue 2, there was an 18 line step disassembler. This should make it even easier; excluding the F3 jump, it is only 3 lines long. If printout is desired, it requires all of 4 lines.

```
0112    JMP    00D0    (this is arbitrary)

00D0    INC    A419
```

```
00D3     JSR     E71D
00D6     RTS
```

To run, toggle the printer off. Next, disassemble the first instruction of the program under examination using the K command and a RETURN following the / prompt. This sets up the various flags and registers. To disassemble subsequent instructions, just press the F3 key.

The printing version goes as follows:

```
0112     JMP     00D0     (again, this is arbitrary)

00D0     INC     A419
00D3     JSR     E71D
00D6     JSR     F04A
00D9     RTS
```

Toggle the printer off, and disassemble the first instruction as above. Hit the PRINT key to print the first instruction. Each press of F3 will disassemble and print the next line.

Michael L. Brachman
3513 Lake Ave. #307
Wilmette, IL 60091

Dear Editor:

I think I've hit on a good way to build data files on tape from AIM BASIC. This is an alternative to the method described by Ralph Reccia in Issue No. 1.

To write a file on tape, insert the following line in the BASIC code before the first PRINT statement you wish to send to tape:

```
POKE4,113:POKE5,232:X=USR(X)
```

This line calls the monitor subroutine WHEREO, which issues the familiar prompts OUT=, F=, T=. Answer these prompts with T, your desired file name, and 1 or 2. This initializes a tape file with the given name. From here on, all BASIC PRINT statements will direct output to the tape buffer, and when the buffer is filled it will be dumped to tape.

Don't forget to close the tape file before leaving the BASIC program. This is necessary to ensure recording the last dab of output. To close, insert the following line after the last PRINT which you want directed to tape:

```
POKE4,10:POKE5,229:X=USR (X)
```

This calls the monitor subroutine DU11, which closes the file and redirects output to the display/printer. As a final touch, optional but nice, stop the tape recorder by inserting the line:

```
POKE43008,207 AND PEEK(43008).
```

(I've assumed that you have the tape recorder remote control connected.)

To read a tape file, insert the following code before the INPUT statements:

```
POKE4,72:POKE5,232:X=USR(X)
```

This calls WHEREI, which issues input prompts, searches for the desired file, and loads the first block into the buffer. Additional blocks are loaded as they are needed. To restore normal operation, insert the line:

```
POKE42002,13
```

A potential problem on input from tape and be sidestepped by ending the file with a distinctive end-of-file flag, say 9999, when it is written. Thus, the end of file can be detected on input by testing each datum as it is read. There is room for some ingenuity here.

Adroit use of POKE42002,84 and POKE42002,13 permit reading alternately from the tape and from the keyboard. The tape file need not be re-initialized each time. POKE42003,84 and POKE42003,13 serve a similar function for output.

Incidentally, I've found that the tape recorder remote controls as provided on the AIM65 interject intolerable noise into the recordings. This is because the power ground is in common with the signal ground and it can be remedied by electrically isolating the power circuit. I use opto-isolators and transistors, but the relay method shown on the back page of Issue No. 1 is probably better.

The TEXT EDITOR can also be useful in dealing with these files. For example, I've prepared a data file of our natural gas usage for the past five years. For this, it was convenient to set up a text file in which each line was one month's gas use. After appending an end-of-file flag, this file was dumped on tape under the file name GAS by means of the editor's L command. The advantage here is that the file can be proofed prior to recording with the help of the T, B, U, D, K, I, and F commands.

How about sending BASIC output to a serial printer? I've found that when the KB/TTY switch is in the TTY position, output is routed to the serial port. Unfortunately, this also disables the keyboard. One way out is to insert the line

```
WAIT 43008,08,08
```

which stops program execution until the KB/TTY switch is thrown to TTY. To restore normal operation, insert

```
WAIT 43008,08
```

which again halts execution until the switch is returned to KB. Don't forget to set the baud rate parameters.

I have found the AIM65 to be very educational, as was the case with the KIM-1 before it. I use both. I appreciate the support Rockwell is giving AIM65 through this newsletter, as well as through peripherals and tech notes.

Earl O. Knutson
51 Ralph Place
Morristown, N.J. 07960

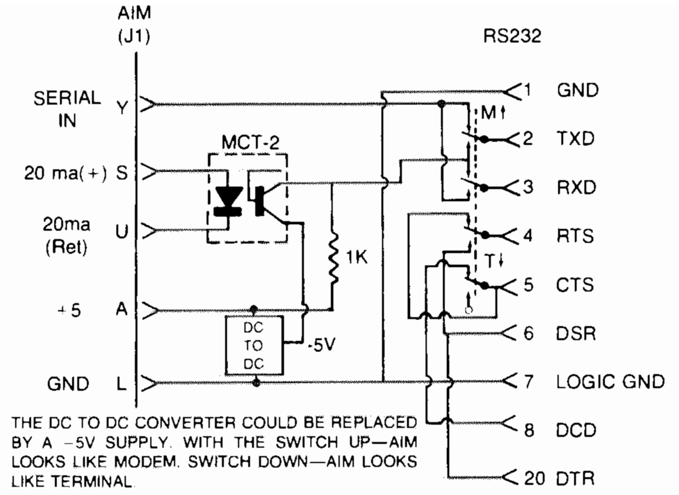
EASY RS232C

R. M. Dumse
Rockwell Int'l

To meet the RS232C requirements it is necessary to convert the TTL levels of the 6500 Series I/O devices on the AIM to RS232C levels. TTL levels are defined as values below 0.8V for a logical zero and above 2.4V for a logical one, with 0V and 5V being the outside limits. The middle region is undefined, meaning a TTL device operating with an input between 0.8V and 2.4V could interpret it to be either a zero or a one. Its output is therefore indeterminate. To have TTL circuits work correctly we must make sure that these levels are correct. RS232 levels are different. A logical one is defined to be any voltage between −3V and −15V, a logical zero between +3V and +15V in the "C" version. The region between −3V and +3V is indeterminate. Note that this is inverted to the way we normally think of logic, a one being negative going and a zero being positive.

To communicate across an RS232 interface, the AIM must be able to send and receive all RS232 signals at these levels. Although not well documented, the AIM is already equipped with a receiver that will translate RS232 signals to TTL levels. This receiver accepts an input from pin Y on the Applications (J1) Connector. Part of the circuitry used is shared with the 20ma current loop receiver. The 20ma current loop transmitter can easily be converted to RS232 levels off the board with the circuitry detailed below.

Not yet mentioned is the fact that RS232 devices communicate serially. The format is generally selectable with at least one mode that is identical to the Teletype format used by the AIM with one start bit and two stop bits. We can therefore use the software in the AIM's Monitor to communicate when the convertor is added.



If the device to be connected has a "handshaking" version of the RS232, it is necessary to generate handshaking signals that allow continuous communication. The circuit shown below uses a scheme of simply "wrapping around" any handshaking signals to meet this end. That is, when it is set to be a modem, a Request To Send (RTS) is wrapped around to the Clear To Send (CTS) line. (Note: To further confuse the issue these signals are negative logic. A zero, meaning level between +3V and +15V, is considered the true condition ie: a Request To Send is a positive voltage when true.)

The circuit shown will work well at speeds in excess of 9600 baud if the AIM 65 used has a 3.3K ohm resistor in R24. This resistor is labelled on the board and can be found behind the printer. Older AIM 65's have a 1K ohm resistor in that position which will not work. Replacing that resistor with the higher value will correct the problem, but will void the AIM's warranty. Refer to section 9. 2. 3. of the AIM 65 USER'S GUIDE for direction on initializing and operating the serial interface.

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