SUPER LANGUAGE
A Home Computer Assembly Language Series

Joy Talk is Cheap

A Low-Cost RS232 Interface Through the TI-99/4A Joystick Port

By Paul Urbanus
6302 Elgin #278
Lubbock, TX 79413

Caveat Joytalker

This article is not for the beginner. If you have electronic construction experience (and some skill in soldering) you can successfully complete the Joytalk system. The hardware required approximately 8 hours to fabricate (including time to gather the parts) in the 99'er lab. The cost for all the parts was under $40. Remember, a mistake in hardware construction is more costly than in software construction—it cannot be corrected with just a few keystrokes!

In the process of computer programming, there eventually comes a time to communicate your results to the outside world. For personal computers, the RS232 serial interface has become the standard link allowing you to communicate to a printer, a plotter, or other peripheral. This first article will describe the hardware required to implement the RS232 output function through the joystick port of the TI-99/4A. (Software will be covered next month.) In this way, users who don't have a peripheral expansion system can output to a printer or other serial device using only a Mini Memory cartridge and some low-cost hardware. The software/hardware combination allows the setting of baud rate (110 – 19.2K), stop bits, parity, and auto carriage return/line feed. The worst baud rate error occurs at 19.2K baud, and is less than 0.2%. A provision is also included to add baud rates which are not preprogrammed.

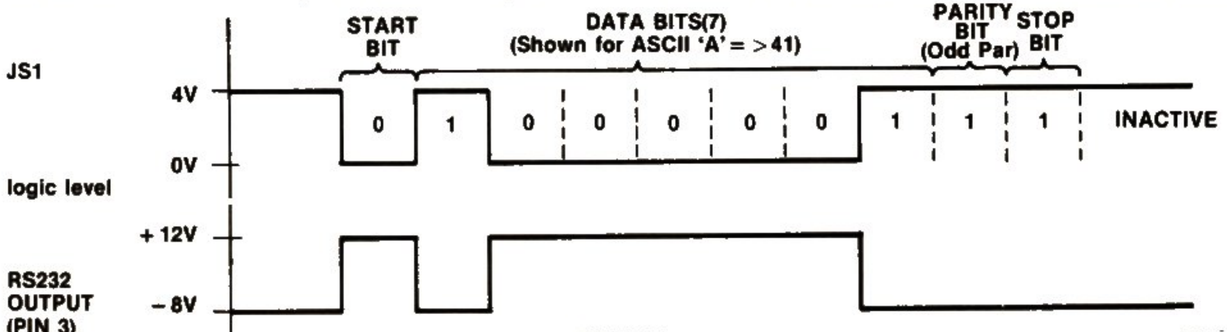
To better understand the hardware and software design tasks, a definition of "RS232" is needed. RS232 is a serial communications standard which defines both electrical specifications and a data transfer protocol. Its electrical characteristics include such things as voltage and loading levels. The relationship between these logic and voltage levels is of interest to us. Notice that the electrical levels are inverted from the logic levels (logic 0 = +V and logic 1 = –V) for RS232.

A transfer protocol is needed for proper flow of data. For the RS232, this protocol specifies the serial data format, as well as the method of *handshaking*. The handshaking in this case involves checking the DATA TERMINAL READY signal to ensure that the remote device (i.e., the printer) is ready to accept data. The serial data format is shown in Figure 1. Notice that there are four distinct pieces which are put together to form the actual data which is transmitted. In this case, an ASCII "A" which occupies seven bits is being transmitted. There are also 3 control bits, which are required both to mark the beginning (START BIT) and end (STOP BIT) of a character, and to perform limited error checking (PARITY BIT). Thus 10 bits are actually transmitted (7 character bits + 3 control bits). For every character transferred, 3 extra bits have been added to "control" the transfer. These control bits are completely transparent to an RS232 user, who merely sends the 7-bit character code to an output subroutine, at which point the control bits are added. Conversely, the receiving device strips off the 3 control bits and uses only the 7-bit character. In the time between transmission of characters, the output logic level is set to 1 (negative RS232 level).

The Hardware

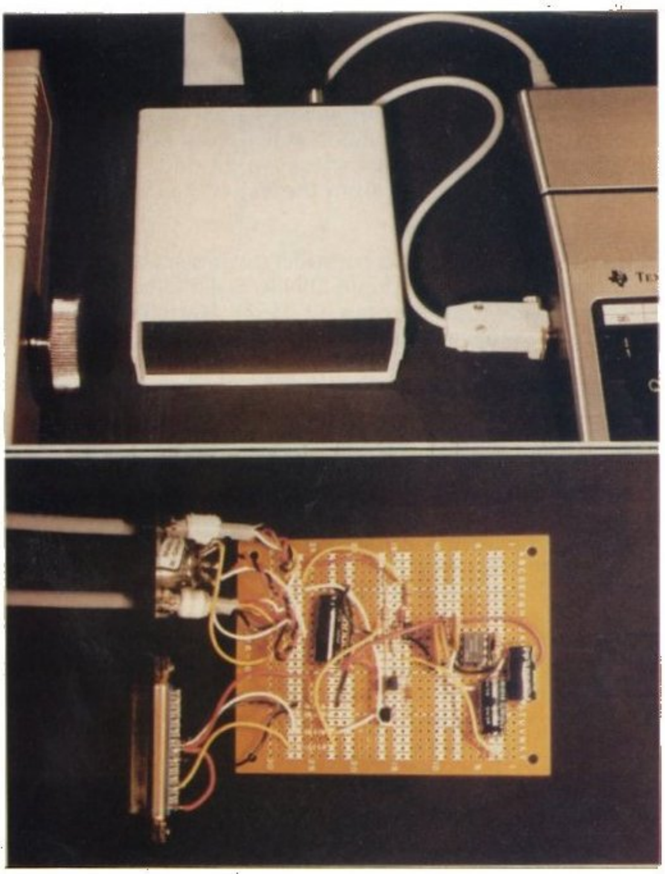
Before the hardware design is started, the I/O structure of the joystick port must be determined. In the 99/4A, the keyboard and joysticks are mapped into an 8 by 8 matrix. The matrix column select lines are *active low*, and are driven by an 8-output *open-collector decoder*. This decoder is controlled by three lines from the system I/O chip (TMS9901). Six of the column selects scan the keyboard. The remaining two are buffered and brought out to the joystick port to select player 1 or player 2 input (or neither). There are five input lines from the joystick input (UP, DOWN, LEFT, RIGHT, and FIRE), but only one joystick select line may be active at a time. Two pins on the joystick port have no internal connection. The absence of power or ground on the joystick port poses a problem. To get around this, ground and +12V must be *stolen* from the video output connector where they are provided to power the RF modulator.

Given the joystick port structure and available power, im-



RS232 TIMING

Figure 1



Top photo: *Joytalk*, installed between a printer and the Home Computer. **Bottom photo:** The completed *Joytalk* interface, minus the case.

plementing the RS232 output function requires three basic blocks:

1. A negative supply voltage to generate the negative voltage levels as required by the RS232 standard.
2. Circuitry to translate the joystick select level to RS232 compatible levels. It is necessary to maintain the RS232 serial output at an inactive level when the joystick select is inactive. This requires that the joystick select level be inverted; otherwise the remote device will see continuous start bits. Since RS232 levels are bipolar (see Figure 1), the inverter stage should have a bipolar output.
3. Finally, some circuitry is needed to translate the printer busy signal to a level which moves between Hi-Z and ground, and is also compatible with the scan matrix levels. If the "device busy" is asserted when the keyboard is being scanned, improper results will be returned. Therefore, the busy signal needs to be gated onto the scan matrix only when it needs to be checked. The remaining joystick select is used for gating control.

The completed hardware design is shown in Figure 4. A look at this schematic reveals the details of the design. The negative voltage supply is generated using a charge-and-dump technique. At the heart of the design is a 555 timer (IC1) free running at approximately 30 KHz. On the positive half of each output cycle, "bucket" capacitor C3 is charged through D1. When the output transitions to ground, D1 turns off and D2 turns on, allowing C3 to "dump" part of its charge into C4. C4 holds the negative voltage level while C3 is recharging.

For the RS232 output, both level translation and inversion must be performed. A common-emitter circuit consisting of Q3, D3, R3, R4, R5 forms an inverter with bipolar output levels. When the joystick select (JS1) is inactive (+4v), Q3 should be off, and the RS232 output will be negative (Figure 1). An active (0V) joystick select should turn on D3 and Q3, raising the RS232 output voltage to +11, allowing for voltage of approximately 2 volts. This is the desired threshold voltage, because it is midway between the joystick output levels. R4 protects Q3 from output shorts, while R3 limits zener current.

Finally, Q1, Q2, D4, R1, and R2 buffer and gate the busy signal. Q2 performs the gating function by keeping the collector of Q1 in the high impedance state if the JS2 output is inactive (high). When JS2 is active (low = 0V), Q2 turns off—allowing the busy input level to ground. The DOWN joystick level is inverted from the actual RS232 level.

Construction Notes

All of the wire needed to construct the project was obtained from one six-foot cable (5-pin DIN to 5-pin DIN) purchased at Radio Shack (Cat. number 42-2151). Starting at one end of the cable, the connector with about 9 inches of cable was cut off to be used for connector J2 as shown in Figure 2 (plugs into the TI-99/4A monitor jack). From the same end of the cable, a 10-inch section of cable was cut off for use with the 9-pin D-type connector (plugs into the joystick port). At each of the three ends, about an inch of the thick outside cable jacket was carefully cut away from the wires inside. There are four wires inside—red, white, black, and yellow. Each wire is wrapped in fine copper wire strands. Unwind the strands from each wire and cut off all of them except for one set which should be carefully twisted into a fifth wire. To obtain wire for hooking up the components in the box, cut another 20″ section of cable. Carefully cut away the entire outside grey jacket from this section, unwind the copper strands from the four colored wires, and discard the strands.

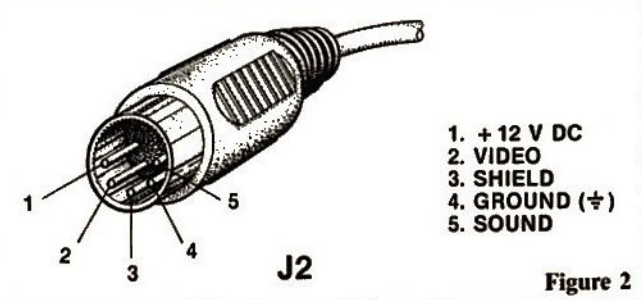


Figure 2

Step two in the construction phase consists of cutting the holes on the back plate of the Radio Shack case (Cat. number 270-218). Follow the hole-drilling template in Figure 7 (see p. 71). Drill very gently so the plastic plate does not crack. Using 4-40 screws and nuts, fasten the connectors J3 and J4 into their respective holes. Be sure to place the connectors through the mounting holes from the outside of the plate. Then pass the prepared cable end of the 5-pin DIN male connector through the plate marked for J2 in Figure 7. Lay the panel aside for now and prepare the J1 connector cable assembly. Connect the wires as shown in Figure 3. (Note that even though only three wires are required, all five connections are made; this adds strength to the cable and does not affect the operation). Pass the free end of the J1 cable through the hole for the J1 cable in the rear panel. The rear panel is now ready to be attached to the circuit board.

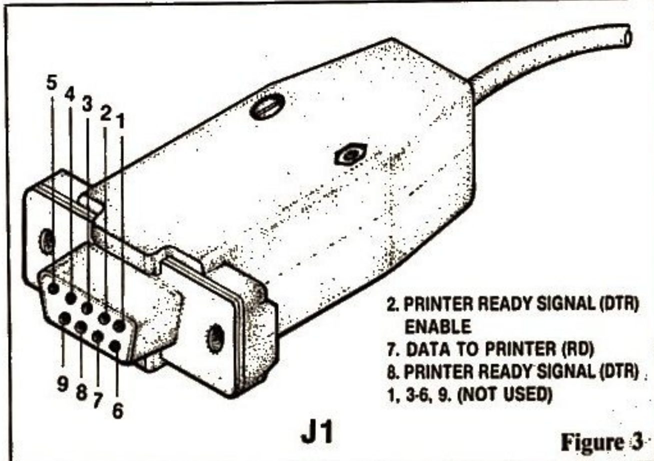

5
4
3
2
1
9
8
7
6
J1
2. PRINTER READY SIGNAL (DTR) ENABLE
7. DATA TO PRINTER (RD)
8. PRINTER READY SIGNAL (DTR)
1, 3-6, 9. (NOT USED)

Figure 3

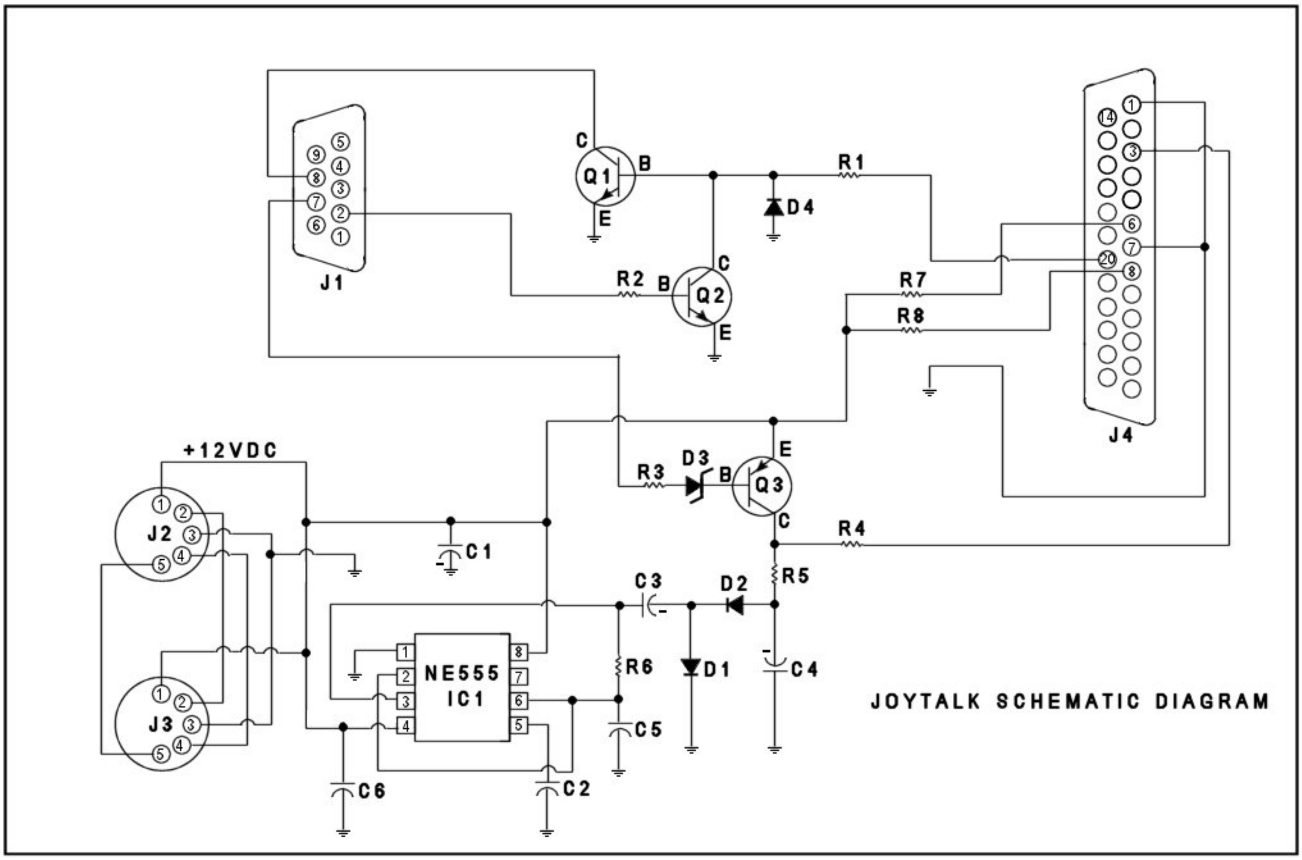


Figure 4

PARTS LIST FOR JOYTALK

SCHEMATIC SYMBOL	QTY	RADIO SHACK PART NO.	DESCRIPTION
J1	1	276-1538	D-Subminiature female 9-position connector
	1	276-1539	9-position D-Subminiature connector hood for above
J2	1	42-2151	6-foot 5-pin to 5-pin DIN cable (use one end)
J3	1	274-005	5-pin chassis socket DIN type
J4	1	276-1548	D-Subminiature female 25-position connector
1C1	1	276-1723	Integrated Circuit Timer (NE555)
Q1, Q2	2	276-2016	NPN transistor (2N3904)
Q3	1	276-2034	PNP transistor (2N3906)
D1, D2, D4	3	276-1620	switching diode (1N914)
D3	1	276-562	9.1V zener diode (1N4739)
C1	1	272-1016	100 uF/35v electrolytic capacitor
C3, C4	3	272-1015	47 uF/35v electrolytic capacitor
C2, C5, C6	3	272-135	0.1 uF disk capacitors
R1, R2 R4, R5, R6 R7, R8	7	271-1328	3.3K ohm, 1/4 watt resistor
R3	1	271-1317	470 ohm, 1/4 watt resistor
	1	270-218	Deluxe Plastic Enclosure (2 1/8″×5″×5 1/4″)
	1	276-1995	8-pin low profile socket (for NE555 IC)
	1	276-162	IC-LSI Perfboard (for mounting circuit parts)
	4	64-3011	4-40 × 1/4″ steel round head machine screws
	4	64-3018	4-40 steel hex machine screw nuts
(optional)	1	64-2801	Science Fair Electronic Tool Set (includes 30 watt soldering iron, needle-nosed pliers, wire cutters, screwdrivers, etc.)

[A version of the JoyTalk device is available in a more compact design as a commercial product from one of our advertisers. JOYPRINT (tm) is offered as a finished product from Model Masters at 2512B E. Fender Ave., Fullerton, CA 92631. The suggested retail price for JOYPRINT is $59.95 (for those of you unwilling to endure the experience of buidling JoyTalk from scratch). — Ed.]

Continued

SUPER LANGUAGE
A Home Computer Assembly Language Series

JoyTalk is Cheap

PART II: SOFTWARE FOR THE RS232 INTERFACE THROUGH THE TI-99/4A JOYSTICK PORT

By Paul Urbanus
6302 Elgin #278
Lubbock, TX 79413

This is the second part of a series on converting the joystick port of the TI-99/4A into a low-cost printer interface. The last article (June, 1983) presented construction plans for the hardware required.

At this point in the project, you've built the hardware—all you need is software to complete the system. The source code for the controlling Assembly Language program, Listing 1, is intended to run in Mini Memory. Most Mini Memory owners have limited systems, so the program has been designed to load with an absolute origin at the initial assembly load point in the Mini Memory cartridge (>7118). Listing 2 is the object (machine) code, which you can enter using EASYBUG. (Be sure to re-initialize the Mini Memory before entering any code.) After you've entered all the code, add the program names and entry addresses to the REF/DEF table starting at address >7FF0. The name and address data is given at the end of the program listing. You also have to set the RAM pointers starting at location >701C in Mini Memory to the values shown at the end of the assembly listing.

The Software

Because speed is essential for this output operation, the controlling software program is in TMS9900 Assembly Language. This program prints a string passed to it from a CALL LINK statement in a TI BASIC program. The main program loop is shown in Listing 1, Sections K, L, M and N. This program uses registers in the faster console CPU RAM. To preserve the BASIC environment, you must save the data in this register area into a temporary buffer. (Before control returns to the BASIC program, this memory must be restored.) Once the BASIC environment has been saved, the program gets the string from BASIC and stores it in a buffer, using the STRREF utility located in the Mini Memory cartridge. The program then calculates the number of control bits. At this point, one character from the string is removed from the string buffer and has start, stop, and parity control bits added. This character—now a piece of data in its final form—is sent to the subroutine that performs the actual character transmission. If the buffer is empty when the next character is requested, the BASIC register data is rolled back in, and control returns to BASIC.

The character transmit subroutine, shown in Sections HH, II and JJ, performs several tasks. It must check the device busy signal before the start of a character transmission. The CRU (Communications Register Unit) of the TMS9900 makes this check and sets the joystick select levels as well. If the device is continuously busy, the keyboard is scanned for the BREAK command in TI BASIC—[FCTN] [4]—about 3 times a second. If the break keys are pressed, BASIC register data is rolled back in, and control returns to BASIC. This is consistent with the operation of the TI RS232 peripheral. If the device is ready (not busy), the baud counter is loaded. The current bit to output is checked, and pin 7 of J1 is set to the proper level with a set/reset bit instruction. After a delay equal to the time necessary to transmit one bit, a check is made to see if all bits have been transmitted. If not, the baud counter is reloaded, and the process starts over again. If all bits have been output, control returns to the main Assembly Language routine.

Using Joytalk

Now that you have the hardware built and the software ready, the next step is to try it out. First, open the Joytalk case so voltage measurements can be made, disconnect the monitor/modulator cable from the computer, and connect the 5-pin plug from the Joytalk into the computer video output jack. Then plug the monitor/modulator cable into Joytalk's 5-pin DIN jack. Turn on the monitor, then the computer. Using a voltmeter, check for the negative voltage supply at the minus side of capacitor C4. Then check for the positive 12 volt supply at IC1 pin 5.

If the computer is not working normally with Joytalk plugged in, or one of the voltages is not present, recheck your wiring. If the wiring is correct, check the polarity of D1, D2, C3, and C4. Also check that the correct transistor connections were made. Once everything is working properly, plug in the RS232 connector from your printer to Joytalk's RS232 connector J4.

To test your Joytalk interface, you'll call two Assembly Language programs from TI BASIC. The first of these sets up the RS232 parameters. These parameters include: baud rate (110 to 19200), stop bits (1 or 2), parity (space, mark, even, odd or none), suppression of automatic carriage return/line feed, suppression of line feed only, and number of data bits (7 or 8). Figure 1 shows how to calculate the number which specifies the desired parameter. (The example given calculates the parameter value for 1200 baud, 1 stop bit, odd parity and 7 data bits.) Once you've calculated this number, it is passed to the parameter-setting subroutine by the following TI BASIC statement:

CALL LINK("JSET",*numeric expression or variable*)

PARAMETER	VALUE	ADD VALUE
BAUD RATE		
110	0	
150	1	
300	2	
600	3	
1200	4	4
2400	5	
4800	6	
9600	7	
19200	8	
USER1	9	
USER2	10	
⋮	⋮	
USER7	15	
DATA BITS		
7	0	0
8	16	
PARITY		
SPACE	0	
MARK	32	
EVEN	64	
ODD	96	96
NONE	128	
STOP BITS		
ONE	0	0
TWO	256	
AUTO CARR RET		
ENABLED	0	0
DISABLED	512	
AUTO LINE FEED		
ENABLED	0	0
DISABLED	1024	
	TOTAL	100

FIG. 1 PARAMETER VALUE CALCULATION:
1200 BAUD, 1 DATA BIT, ODD PARITY, 1 STOP BIT,
AUTO CR&LF

The second subroutine you call from TI BASIC is a string output routine. It outputs through the joystick port the contents of the string passed to it by TI BASIC. The software will add and send out carriage returns and line feeds if you set the proper parameters (*enable* carriage returns and line feeds). The format of the TI BASIC statement for string output is

CALL LINK("JOUT",*string expression or variable*)

The following short program tests the Joytalk interface. This test uses the following parameters: 7 bits, odd parity, 1200 baud, 1 stop bit. Other combinations of parameters could be used, however.

```
100 REM 7 DATA BITS 1 STOP BIT ODD PARITY 1200 BAUD
110 CALL LINK("JSET",100)
120 INPUT A$
130 REM OUTPUT STRING TO JOYSTICK RS232
140 CALL LINK("JOUT",A$)
150 GOTO 120
```

All calls to the Joytalk software must use the CALL LINK statement. You cannot access the Joytalk software through the TI BASIC PRINT statement because no other software entry points are provided.

User-Defined Baud Rates

Although all the standard baud rates are available with the Joytalk program, provisions are included to allow you to program your own baud rates. To calculate the new baud counter value, first calculate the time (microseconds) of one data bit. This time is equal to 1,000,000/(baud rate). Using this time (BTIME), calculate two numbers (X, Y) using the following formula:

$$BTIME = 41.33 + 9.33*(X) + 0.667(Y)$$

with $0<X<4096$ and $0<Y<15$. After you've calculated X and Y, join them to form one 16-bit word with the following formula:

$$BAUD\ TABLE\ VALUE = X + Y*4096$$

You need to enter new values into the baud rate table beginning at USERBD (>7404). Each user will take one 16-bit word. USER1 will occupy the word beginning at >7404; USER2 will occupy the word >7406, and so on. To implement these USER baud rates, merely incorporate the appropriate value from Figure 1 when calculating the RS232 parameter.

Comments

Well, you now have a low-cost serial interface which allows you to talk to the outside world through TI BASIC—or Assembly Language if you modify the program. And you still have about 3K of unused RAM in the Mini Memory cartridge just waiting to be filled . . .

> *In the previous section of* Joytalk *(June 1983), the schematic diagram on page 65 (Fig 4.) had some components inadvertently switched.* Debugs, *on page 76 in this issue, contains a corrected diagram.*

Listing 1

```
        TITL 'JOYSTICK RS232'

*
*       RS232 OUTPUT THRU JOYSTICK
*
*       BY   PAUL URBANUS
*
***  SYSTEM EQUATES
*
PAD     EQU  >8300          START OF FAST 16 BIT CPU RAM
FAC     EQU  PAD+>4A        FLOATING ACCUMLATOR
KUNIT   EQU  PAD+>74        KEYBOARD # TO BE SCANNED
KCODE   EQU  PAD+>75        KEYCODE IS RETURNED
STATUS  EQU  PAD+>7C        GPL/SYSTEM STATUS FLAGS
GPLWS   EQU  PAD+>E0        SYSTEM WORKSPACE
FASTWS  EQU  PAD            SOFT 232 WORKSPACE AT START OF RAM
*
***  BASIC UTILITIES IN MINI MEMORY ROM
*
NUMREF  EQU  >6044          * UTILITY VECTORS
STRREF  EQU  >604C          *   FOR ROUTINES
XMLLNK  EQU  >601C          *     LOCATED IN
ERR     EQU  >6050          *       MINI MEMORY ROM
*
*
        AORG >7118          START OF AVAILABLE MINI MEMORY RAM
*
        EVEN
*
*** MISCELLANEOUS MASKS AND DATA EQUATES
*
STPSTS  DATA >0100      A       STOP BITS MASK
PARMK1  DATA >0080              * PARITY
PARMK2  DATA >0040              *   MASK
PARMK3  DATA >0020              *     BITS
QTYMSK  DATA >0010              NUMBER OF DATA BITS MASK
BAUDMK  DATA >000F              BAUD TABLE INDEX MASK
LFMASK  DATA >0200              AUTO LINE FEED MASK
CRMASK  DATA >0400              AUTO CARRIAGE RETURN MASK
PARBIT  DATA >0100              DEFAULT PARITY BIT POSITION
STRMSK  DATA >0001              START BIT MASK
STPMSK  DATA >0700              STOP BIT MASK
H0001   DATA >0001              USED IN PARITY SETTING ROUTINE
H00     BYTE >00
HFF     BYTE >FF
*
***  RAM BUFFERS AND RAM VARIABLES
*
BSCBUF  BSS  32                 ROLLOUT MEMORY FOR FAST RAM
*
STRBUF  BSS  256                INPUT DATA BUFFER
*
STATRS  DATA >0062      B       RS232 PARAMETER WORD
BSCRET  BSS  2                  BASIC RETURN ADDR SAVE LOC.
*
*
************************************************************
*
*            CRU INIT SUBR
*
*       CALLED BY:  BL  @SETUP
*
*       REG USE: R1,R12
*
*       SET R12 CRU BASE TO POINT TO SCAN MATRIX DECODER
*       SET SCAN DECODER TO SELECT JOYSTICK 2(JS2-=0V)
*       THIS ACTION GATES BUSY ONTO THE DOWN INPUT OF THE
*       JOYSTICK INPUT BUSS.
*
************************************************************
*
```

```
*      R2 = TEMP,SCRATCH SHIFT
*      R3 = BAUD COUNTER (LOOP PERMANENT)
*      R4 = LOOP COUNTER VARIABLE(BITCNT)
*      R5 = TEMP VARIABLE FOR BITLOOP
*      R6 = PERMANENT BIT COUNT
*      R7 = BUFFER POINTER
*      R8 = BUFFER LENGTH
*      R9 = 3RD LEVEL(INNERMOST) SUBROUTINE LINK
*      R10= 2ND LEVEL SUBROUTINE LINK
*      R11= 1ST LEVEL(OUTERMOST)SUBROUTINE LINK
*
************************************************************
*
*
JOUT   EQU  $
       LIMI 0
       MOV  R11,@BSCRET   K      SAVE LINK TO BASIC
       BL   @GETSTR              GET THE BASIC STRING
       BL   @SAVEIT              SAVE FAST RAM CONTENTS
       MOV  R7,@FASTWS+14        PASS BUFFER POINTER..
       MOV  R8,@FASTWS+16        ...AND BUFFER LENGTH
       LWPI FASTWS               GET READY TO GO FASTER!!!
       BL   @BITCNT              FIND NUMBER OF BITS TO X-MIT
       BL   @SETUP               SET UP JOYSTICK MUX
FETCH2 EQU  $
       MOVB *R7+,R1       L      GET NEXT CHAR FROM BUFFER
       DEC  R8                   ADJUST REMAINING CHAR COUNT
       JLT  RET2
       BL   @OUTCHR              ADJUST & OUTPUT ONE CHAR
       JMP  FETCH2               ...AND LOOP IN NOT END OF STR.
RET2   EQU  $
       BL   @CKAUTO       M      CHECK AUTOMATIC OPTIONS
BRKRET EQU  $                    RETURN ENTRY IF BREAK KEY DOWN
       LWPI STRBUF               TEMP REGS TO RESTORE FAST RAM
       BL   @RESTOR       N      RESTORE DATA FOR BASIC
       MOV  @BSCRET,R11          RESTORE CALLER ADDRESS...
       MOVB @H00,@STATUS         CLEAR ERROR IN CASE OF BREAK
       RT                        ...AND RETURN
*
*
************************************************************
*
*         DATA OUTPUT ROUTINE
*
*    CALLED BY:  BL   @OUTCHR
*
*    THIS ROUTINE DOES SEVERAL THINGS:
*
*    1. ADDS CONTROL BITS(START/STOP/PARITY) TO DATA
*    2. GETS CURRENT BAUD RATE VALUE
*    3. OUTPUTS THE CHARACTER
*
*
************************************************************
*
OUTCHR EQU  $
       MOV  R11,R10       O      SAVE SUBROUTINE LINK
       SRL  R1,8                 RIGHT ADJUST  OUTPUT BYTE
       BL   @PARSET              SET UP START, STOP, & PARITY
*                                BITS IN DATA BYTE
       BL   @GETBD        P      GET BAUD RATE AND SHIFT COUNT
       MOV  R6,R5                COPY # BITS FROM PERMANENT REG
       BL   @SENDIT              TRANSMIT THE DATA
       B    *R10                 RETURN TO CALLER
*
************************************************************
*
*          AUTOMATIC CARRIAGE RETURN & LINE FEED
*
*       CALLED BY: BL @CKAUTO
*
*       THIS ROUTINE CHECKS THE PARAMETER WORD AND SENDS
*       A CARRIAGE RETURN OR LINE FEED IF ENABLED TO DO SO
*
************************************************************
*
CRLF   BYTE >0D,>0A       Q      DATA FOR AUTO CR & LF
       EVEN
CKAUTO EQU  $
       MOV  R11,R9        R      SAVE SUBROUTINE LINK
       MOV  @STATRS,R3           COPY STATUS WORD
       CZC  @CRMASK,R3           IS AUTO CARR RET ENABLED.
       JNE  CHKLF                IF NOT, CHECK FOR LINE FEED
       MOVB @CRLF,R1             COPY CARRIAGE RETURN CODE
       BL   @OUTCHR              OUTPUT CARRIAGE RETURN
       MOV  @STATRS,R3           RESTORE PARAMETERS IN REG
CHKLF  CZC  @LFMASK,R3           WHAT ABOUT LINE FEED?
       JNE  AUTORT               RETURN IF NOT ENABLED
       MOVB @CRLF+1,R1           GET LINE FEED ASCII CODE
       BL   @OUTCHR              AND SEND IT
AUTORT EQU  $
       B    *R9           S      RETURN
*
```

```
        EVEN
*
PIN7AD EQU  36                   ADDR OF SCAN MATRIX DECODER
PIN7EN EQU  >0700                TURN ON LAST DECODER OUTPUT
*
SETUP  LI   R12,PIN7AD    C      LOAD CRU ADDRESS OF DECODER
       LI   R2,PIN7EN            SELECT PIN 2 ON JOYSTICK PORT
       LDCR R2,3                 ...AND SET IT TO GROUND
       RT
*
**********************************************************
*
*   SUBROUTINE TO GET AN INPUT STRING FROM BASIC
*
*      REGISTERS AFFECTED:
*
*      R0 - ZOT
*      R1 - ZOT
*      R2 - ZOT
*      R7 - RETURNS STRING BUFFER POINTER
*      R8 - RETURNS STRING LENGTH
*
**********************************************************
GETSTR EQU  $
       CLR  R0            D      GET STRING PARAMETER
       LI   R1,1                 GET FIRST(AND ONLY) PARAMETER
       LI   R2,STRBUF            LOAD BUFFER POINTER
       MOVB @HFF,*R2             MAX BUFFER LEN = 255
       BLWP @STRREF              GET STRING
       MOV  R2,R7                COPY BUFFER POINTER
       MOVB *R7+,R8              GET LENGTH BYTE
       SRL  R8,8                 RIGHT ADJUST LENGTH BYTE
       RT
*
************************************************************
*
*          SAVE AND LOAD LOOPS FOR FAST RAM SAVE/LOAD
*
*      CALLED BY:  BL   @SAVEIT - SAVE FAST RAM DATA IN
*                                 EXTERNAL BUFFER
*
*                  BL   @RESTOR   RESTORE FAST RAM DATA
*                                 FROM EXTERNAL BUFFER
*
*      REGISTER USAGE:  R0,R1,R2
*
************************************************************
*
SAVEIT EQU  $
       LI   R0,FASTWS     E      SET LOAD POINT IN FAST RAM
       LI   R1,BSCBUF            LOAD START OF CODE TO BE MOVED
       JMP  MOVENT               JUMP AND DO BLOCK MOVE
*
RESTOR EQU  $
       LI   R0,BSCBUF     F      THIS TIME BUFFER IS SOURCE
       LI   R1,FASTWS            AND FAST RAM IS DESTINATION
MOVENT LI   R2,32                32 BYTES TO SAVE/RESTORE
MOVLP1 MOV  *R0+,*R1+            MOVE TWO BYTES OF CODE
       DECT R2                   DECREMENT BLOCK LENGTH CNTR
       JNE  MOVLP1               IF NOT DONE, MOVE TWO BYTES
       RT                        PASS CONTROL TO OUTPUT ROUTINE
*
************************************************************
*
*    SET UP RS 232 PARAMETERS FROM BASIC
*
*    THIS ROUTINE INPUTS A NUMERIC VALUE AND USES THIS
*    VALUE TO SPECIFY THE RS232 PARAMETERS
*
*    CALLED IN BASIC BY:  CALL LINK("JSET",<num expr/var>)
*
************************************************************
*
H03    BYTE 3             G      ERROR CODE RETURNED FROM
*                                CONVERT FLT-PT TO INTEGER
       EVEN
MAXSET DATA >0800         H      MAX VALUE OF SETUP INTEGER
*
JSET   CLR  R0            I      ZERO FOR SCALAR NUMERIC
       LI   R1,1                 PICK UP FIRST AND ONLY PARAM
       BLWP @NUMREF              GET THE NUMERIC PARAMETER
       BLWP @XMLLNK              GO TO CONSOLE ROM CODE TO...
       DATA >1200                ...CONVERT FLTPT TO INTEGER
       CB   @H03,@FAC+10         CHECK FOR OVERFLOW ERROR
       JEQ  BADV                 INDICATE ERROR TO USER
       C    @FAC,@MAXSET         IS PARAMETER OUT OF RANGE
       JHE  BADV                 IF YES, JUMP AND INDICATE IT
       MOV  @FAC,@STATRS         SAVE NEW RS232 SETUP PARAMS.
       RT                        BYE! BYE!
*
BADVAL EQU  >1300                'BAD VALUE' ERROR MESSAGE
*
BADV   EQU  $
       LI   R0,BADVAL     J      LOAD BAD VALUE POINTER
       BLWP @ERR                 CALL ERROR HANDLER
*
************************************************************
*
*          JOYSTICK OUTPUT (MAIN ROUTINE)
*
*    CALLED IN BASIC BY: CALL LINK("JOUT",<str exp/str var>)
*
*         REGISTER USAGE
*
*    R0 = VARIABLE SHIFT COUNT
*    R1 = DATA IN MSBYTE
```

```
**********************************************************
*
*   DATA FORMAT FOR RS232 PARAMETERS
*
*            WORD: @STATRS
*
*¦15-11¦10-9¦ 8 ¦ 7 ¦ 6 ¦ 5 ¦ 4 ¦ 3 ¦ 2 ¦ 1 ¦ 0 ¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦   ¦___¦___¦___¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦         ¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦         ¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦    0-15=INDEX INTO
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦    SOFTWARE BAUD
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦    TIMER TABLE(BAUD RATE)
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦
*   ¦     ¦   ¦   ¦   ¦   ¦   ¦__ 0=7 DATA BITS
*   ¦     ¦   ¦   ¦   ¦   ¦       1=8 DATA BITS(IGNORE PARITY)
*   ¦     ¦   ¦   ¦   ¦   ¦
*   ¦     ¦   ¦   ¦   ¦   ¦     0=SPACING PARITY
*   ¦     ¦   ¦   ¦   ¦   ¦     1=MARKING PARITY
*   ¦     ¦   ¦   ¦___¦___¦____ 2=EVEN PARITY
*   ¦     ¦   ¦                 3=ODD PARITY
*   ¦     ¦   ¦                 4=NO PARITY
*   ¦     ¦   ¦__ 0=1 STOP BIT
*   ¦     ¦       1=2 STOP BITS
*   ¦     ¦
*   ¦     ¦__ 0=AUTO LINE FEED ENABLED
*   ¦     ¦   1=AUTO LINE FEED DISABLED
*   ¦     ¦
*   ¦     ¦__ 0=AUTO CRLF ENABLED
*   ¦         1=AUTO CRLF DISABLED
*   ¦
*   ¦-- UNDEFINED(SET TO ZERO)
*
**********************************************************
*
*           PARAMETER SETTING SUBROUTINE
*
*      CALLED BY:  BL  @PARSET
*
*    THIS SUBROUTINE MODIFIES THE DATA IN R1LSBYTE
*
*      1. ADJUSTS FOR DATA LENGTH(7 OR 8 DATA BITS)
*      2. INSERTS THE CORRECT NUMBER OF STOP BITS(1 OR 2)
*      3. INSERTS THE START BIT
*      4. CALCULATES AND INSERTS THE PARITY BIT(IF SELECTED)
*
*      REGISTER USAGE: R0-R4
*
*      R1 - RETURNS DATA WITH CONTROL BITS READY FOR SHIFTER
*
**********************************************************
*
PARSET EQU  $
       MOV  @STATRS,R3  T    COPY RS232 STATUS WORD
       COC  @PARMK1,R3       IS PARITY DESIRED?
       JEQ  RETPAR           RETURN IF NOT NEEDED.
       COC  @PARMK2,R3       CHECK IF NEED TO CALC PARITY
       JEQ  CALCPA           IF SO, JUMP AND DO IT
FXDPAR MOV  R3,R4            NO PARITY CALC, SO SET IT(0/1)
       SRL  R4,5             MOVE PARITY STATE TO LSBIT
       JMP  PBIT             JUMP AND SET THE BIT
CALCPA EQU  $
       CLR  R4          U    CLEAR PARITY FLAG
       COC  @QTYMSK,R3       IS IT REALLY 8 DATA BITS?
       JEQ  CHKPAR           IF SO, JUMP AND CHECK 1ST BIT
       ANDI R1,>007F         MAKE SURE 8TH BIT IS ZERO
CHKPAR EQU  $                ENTRY POINT TO CALC PARITY
       MOVB @FASTWS+3,R2 V   MAKE R1LSBYTE AFFECT STATUS
       JOP  CHKEVN           PARITY SET FOR ODD, SO JUMP
       INC  R4               SET LSBIT FOR ODD PARITY
CHKEVN EQU  $
       COC  @PARMK3,R3  W    CHECK EVEN/ODD PARITY?
       JEQ  PBIT             IF ODD PARITY, ITS SET UP OK
       XOR  @H0001,R4        INVERT PARITY BIT/MAKE IT EVEN
PBIT   EQU  $
       LI   R0,7        X    DEFAULT SHIFT FOR PARITY MASK
       CZC  @QTYMSK,R3       CHECK 7 OR 8 DATA BITS
       JEQ  PBIT2            JUMP IF 7 BITS
       INC  R0               ADJ SHIFT COUNT FOR 8 BITS
PBIT2  SLA  R4,0             POSITION PARITY BIT
       MOV  @H0001,R2        LOAD UP INITIAL MASK VALUE
       SLA  R2,0             LINE UP MASK BIT W/ PARITY BIT
       SZC  R2,R1            CLR OUT PARITY BIT, THEN...
       XOR  R4,R1            ...MASK IN CORRECT VALUE
RETPAR EQU  $
       SLA  R1,1        Y    MAKE ROOM FOR START BIT
       MOV  R6,R0            COPY NUMBER OF BITS TO SHIFT
       DEC  R0
       CZC  @STPSTS,R3       IS IT ONE OR TWO STOP BITS?
       JEQ  SHFIT2           JUMP IF ONLY ONE STOP BIT
       DEC  R0               2 STOP BITS, SO ADJ SHIFT CNT
SHFIT2 EQU  $
       LI   R2,>0003    Z    LOAD STOP BIT MASK
       SLA  R2,0             PUT STOP BITS IN PROPER PLACE
       SOC  R2,R1            NOW SET THE STOP BIT(S) UP
       RT
*
**********************************************************
*
*          BIT COUNT SUBROUTINE
*
*      CALLED BY: BL  @BITCNT
*
```

```
*      REGISTER USAGE:
*
*      R0 - DESTROYED
*      R6 - RETURNS TOTAL NUMBER OF BITS TO SEND
*
*      R6 = DATA LENGTH(7 OR 8) + 1 START BIT
*           + #STOP BITS(1 OR 2) + PARITY BITS(0 OR 1)
**********************************************************
*
BITCNT EQU  $
       MOV  @STATRS,R0  AA   COPY PARAMETER WORD
       LI   R6,9             LOAD DEFAULT BIT COUNT FOR
***                          1 STOP BIT,1 START BIT AND
***                          NO PARITY BIT, 7 DATA BITS
       COC  @STPSTS,R0  BB   IS IT ONE OR TWO STOP BITS?
       JNE  BITS10           JUMP IF ONE STOP BIT
       INC  R6               MAKE BIT COUNT=11
BITS10 EQU  $
       COC  @PARMK1,R0  CC   CHECK FOR PARITY
       JEQ  BITS11           JUMP IF NO PARITY SET
       INC  R6               ADJUST BIT CNT TO INCL PARITY
BITS11 EQU  $
       COC  @QTYMSK,R0  DD   CHECK FOR A 8 DATA BITS
       JNE  BITS12           IF DATA BITS=7, THEN JUMP
       INC  R6               ADJUST BIT CNT TO INCL 8 DBITS
BITS12 EQU  $
       RT               EE   RETURN TO CALLER
*
**********************************************************
*
*          GET BAUD RATE FROM TABLE
*
*      CALLED BY:  BL  @GETBD
*
*      REGISTER USAGE: R0,R3
*
*      RETURNS:  COARSE LOOP VALUE IN R0
*                FINE TUNING VALUE IN R3
**********************************************************
*
GETBD  EQU  $
       MOV  @STATRS,R3  FF   GET BAUD RATE INDEX FROM
       ANDI R3,>000F         TABLE AND MASK OFF UNEEDED BIT
       SLA  R3,1             MAKE MASKED VALUE WORD INDEX
       MOV  @BACNTR(R3),R3   GET BAUD COUNTER VALUE
       MOV  R3,R0            COPY BAUD RATE DATA
       SRL  R0,12            ADJUST SHIFT COUNT INTO R0
       ANDI R3,>0FFF
       RT
*
**********************************************************
*
*   SOFTWARE BIT TIMER VALUES
*
**********************************************************
*
BACNTR DATA 0*4096+970  GG   110    BAUD
       DATA 0*4096+710       150    BAUD
       DATA 0*4096+353       300    BAUD
       DATA 2*4096+174       600    BAUD
       DATA 12*4096+84       1200   BAUD
       DATA 3*4096+40        2400   BAUD
       DATA 13*4096+17       4800   BAUD
       DATA 10*4096+6        9600   BAUD
       DATA 2*4096+1         19200  BAUD
USERBD EQU  $                FIRST USER BAUD RATE(USER1)
ENDCNT EQU  $-BACNTR
       BSS  32-ENDCNT        SPACE FOR ADDITIONAL CUSTOM
*                            USER BAUD RATES
*
**********************************************************
*
*           SEND ONE CHARACTER
*
*      CALLED BY:  BL  @SENDIT
*
*   FAST LOOP TO TRANSMIT ONE CHARACTER THRU
*   THE JOYSTICK PORT
*
*   IF THE RECEIVING DEVICE IS BUSY, THE KEYBOARD IS
*   SCANNED APPR. EVERY 1/3 SEC FOR THE CLEAR KEY(FCTN 4),
*   WHICH IS ALSO THE BASIC 'BREAK' KEY.
*
*   IF THE CLEAR KEY IS PRESSED, CONTROL RETURNS TO BASIC
*   AND NO MORE CHARACTERS ARE SENT
*
*      REGISTER USAGE:
*
*      R0 - VARIABLE SHIFT COUNT FOR FINE TIMING CONTROL
*      R1 - CONTAINS DATA TO BE SHIFTED OUT
*      R2 - DUMMY REGISTER USED IN VARIABLE COUNT SHIFT
*      R3 - VALUE OF ONE BIT TIME WHICH IS PRESERVED
*      R4 - DECREMENT REGISTER FOR BIT TIME COUNTER
*      R5 - NUMBER OF BITS TO TRANSMIT
*
**********************************************************
*
BRKKEY BYTE >02        HH    CODE FOR 99/4 CLEAR KEY

       EVEN
SENDIT EQU  $
```

```
BUSYIN LI   R2,>4000        II    LOAD TIME BETWEEN BREAK CHECKS
BUSYLP DEC  R2                    COUNT DOWN ONE AT A TIME
       JNE  TESTIT                BREAK CHECK NOT READY SO JUMP
       MOV  @GPLWS+22,R2          SAVE GPL RETURN LINK
       LWPI GPLWS                 LOAD UP SYSTEM WORKSPACE
       MOVB @H00,@KUNIT           SCAN KEYBOARD ZERO
       BL   @>000E                GO TO CONSOLE ROM CODE
       LWPI FASTWS                RETURN TO RS232 WORKSPACE
       MOV  R2,@GPLWS+22          RESTORE SYSTEM RETURN ADDRESS
       MOV  R11,R13               SAVE SUBROUTINE LINK
       BL   @SETUP                TURN ON DEVICE BUSY GATE
       MOV  R13,R11               RESTORE SUBROUTINE LINK
       CB   @BRKKEY,@KCODE        WAS BREAK KEY PRESSED?
       JNE  BUSYIN                NO, SO CHECK BUSY LINE AGAIN
       B    @BRKRET               RETURN TO BASIC
TESTIT TB   -12                   SEE IF THE DEVICE IS BUSY
       JEQ  BUSYLP                IF SO, MAYBE NEED TO CHK BREAK
NXTBIT SRC  R1,1                  MOV ONE BIT INTO CARRY
       JOC  SETONE                IF BIT IS ONE, JUMP & OUTP 1
       SBZ  0                     BIT WAS ZER0, SO SET OUTP TO 0
       JMP  BITDLY                JUMP AND DELAY ONE BIT TIME
SETONE SBO  0                     SEND A '1' BIT
       JNC  $+2                   MAKE TIMING SAME BOTH WAYS
BITDLY MOV  R3,R4                 GET THE BAUD DECREMENTER COUNT
BITLP  DEC  R4                    BAUD COUNTER LOSES ONE...
       JNE  BITLP                 ...LOOP AGAIN IF NOT TIMED OUT
       SRC  R2,0                  THIS ALLOWS FINE TUNING OF
*                                  OF THE TIME WITH 2/3 US RESOL
       DEC  R5              JJ    BIT COUNTER LOSES ONE.
       JNE  NXTBIT                IF ALL BITS NOT OUTP., RE-LOOP
       RT
ENDADR EQU  $
*
*  MINI MEMORY USERS SHOULD ENTER THE FOLLOWING DATA IN THE
*  REF/DEF TABLE & 'MEMORY AVAILABLE' POINTERS
*
       AORG >7FF0
       EVEN                 KK
       BYTE 'J','O','U','T',' ',' '   NAME FOR REF/DEF TABLE
       DATA JOUT                  ADDRESS TO BRANCH ON NAME LINK
       BYTE 'J','S','E','T',' ',' '   PARAMETER SET ROUTINE
       DATA JSET                  ENTRY ADDRESS
*
       AORG >701C
       DATA ENDADR          LL    FIRST FREE ADDRESS IN MINI MEM
       DATA >7FF0                 BOTTOM OF REF/DEF TABLE
       DATA 0                     NO DEFAULT ENTRY ADDRESS
       DATA 0,0,0,0               DON'T RECOGNIZE MEMORY EXPANSN

       END
```

The following object code listing has two columns. The left-hand column has memory location addresses. Since the addresses given are all even hexadecimal numbers, they are *word* boundaries. The right-hand column contains the *contents* of that word in hexadecimal. Because EASYBUG's addressing increments by *bytes*, it only permits you to enter bytes. Thus, to enter the following data using EASYBUG, first access EASYBUG, then type M7118. Next, from the column opposite 7118, enter the *leftmost* two digits: 01. Pressing **[ENTER]** advances you to memory location 7119, the second byte of the word beginning at 7118. Now, from the column opposite 7118, type in the *rightmost* two digits: 00. Press **[ENTER]** again, 711A appears, and you repeat the process. The letters at the head of each section of this listing correspond to the letters on each grey section of Listing 1. This will allow you to compare the source code listing with the assembled object code.

JOYTALK
LISTING 2

	Addr.	Cont.
A	7118	0100
	711A	0080
	711C	0040
	711E	0020
	7120	0010
	7122	000F
	7124	0200
	7126	0400
	7128	0100
	712A	0001
	712C	0700
	712E	0001
	7130	00FF
B	7252	0062
C	7256	020C
	7258	0024
	725A	0202
	725C	0700
	725E	30C2
	7260	045B
D	7262	04C0
	7264	0201
	7266	0001
	7268	0202
	726A	7152
	726C	D4A0
	726E	7131
	7270	0420
	7272	604C
	7274	C1C2
	7276	D237
	7278	0988
	727A	045B
E	727C	0200
	727E	8300
	7280	0201
	7282	7132
	7284	1004
F	7286	0200
	7288	7132
	728A	0201
	728C	8300
	728E	0202
	7290	0020
	7292	CC70
	7294	0642
	7296	16FD
	7298	045B
G	729A	0300
H	729C	0800
I	729E	04C0
	72A0	0201
	72A2	0001
	72A4	0420
	72A6	6044
	72A8	0420
	72AA	601C
	72AC	1200
	72AE	9820
	72B0	729A
	72B2	8354
	72B4	1308
	72B6	8820
	72B8	834A
	72BA	729C
	72BC	1404
	72BE	C820
	72C0	834A
	72C2	7252
	72C4	045B
J	72C6	0200
	72C8	1300
	72CA	0420
	72CC	6050
K	72CE	0300
	72D0	0000
	72D2	C80B
	72D4	7254
	72D6	06A0
	72D8	7262
	72DA	06A0
	72DC	727C
	72DE	C807
	72E0	830E
	72E2	C808
	72E4	8310
	72E6	02E0
	72E8	8300
	72EA	06A0
	72EC	73B8
	72EE	06A0
	72F0	7256
L	72F2	D077
	72F4	0608
	72F6	1103
	72F8	06A0
	72FA	7316
	72FC	10FA
M	72FE	06A0
	7300	732C
N	7302	02E0
	7304	7152
	7306	06A0
	7308	7286
	730A	C2E0
	730C	7254
	730E	D820
	7310	7130
	7312	837C
	7314	045B
O	7316	C28B
	7318	0981
	731A	06A0
	731C	7354
P	731E	06A0
	7320	73DA
	7322	C146
	7324	06A0
	7326	7414
	7328	045A
Q	732A	0D0A
R	732C	C24B
	732E	C0E0
	7330	7252
	7332	24E0
	7334	7126
	7336	1606
	7338	D060
	733A	732A
	733C	06A0
	733E	7316
	7340	C0E0
	7342	7252
	7344	24E0
	7346	7124
	7348	1604
	734A	D060
	734C	732B
	734E	06A0
	7350	7316
S	7352	0459
T	7354	C0E0
	7356	7252
	7358	20E0
	735A	711A
	735C	1321
	735E	20E0
	7360	711C
	7362	1303
	7364	C103
	7366	0954
	7368	100F
U	736A	04C4
	736C	20E0
	736E	7120
	7370	1302
	7372	0241
	7374	007F
V	7376	D0A0
	7378	8303
	737A	1C01
	737C	0584
W	737E	20E0
	7380	711E
	7382	1302
	7384	2920
	7386	712E
X	7388	0200
	738A	0007
	738C	24E0
	738E	7120
	7390	1301
	7392	0580
	7394	0A04
	7396	C0A0
	7398	712E
	739A	0A02
	739C	4042
	739E	2844
Y	73A0	0A11
	73A2	C006
	73A4	0600
	73A6	24E0
	73A8	7118
	73AA	1301
	73AC	0600
Z	73AE	0202
	73B0	0003
	73B2	0A02
	73B4	E042
	73B6	045B
AA	73B8	C020
	73BA	7252
	73BC	0206
	73BE	0009
BB	73C0	2020
	73C2	7118
	73C4	1601
	73C6	0586
CC	73C8	2020
	73CA	711A
	73CC	1301
	73CE	0586
DD	73D0	2020
	73D2	7120
	73D4	1601
	73D6	0586
EE	73D8	045B
FF	73DA	C0E0
	73DC	7252
	73DE	0243
	73E0	000F
	73E2	0A13
	73E4	C0E3
	73E6	73F2
	73E8	C003
	73EA	09C0
	73EC	0243
	73EE	0FFF
	73F0	045B
GG	73F2	03CA
	73F4	02C6
	73F6	0161
	73F8	20AE
	73FA	C054
	73FC	3028
	73FE	D011
	7400	A006
	7402	2001
HH	7412	0200
II	7414	0202
	7416	4000
	7418	0602
	741A	1617
	741C	C0A0
	741E	83F6
	7420	02E0
	7422	83E0
	7424	D820
	7426	7130
	7428	8374
	742A	06A0
	742C	000E
	742E	02E0
	7430	8300
	7432	C802
	7434	83F6
	7436	C34B
	7438	06A0
	743A	7256
	743C	C2CD
	743E	9820
	7440	7412
	7442	8375
	7444	16E7
	7446	0460
	7448	7302
	744A	1FF4
	744C	13E5
	744E	0B11
	7450	1802
	7452	1E00
	7454	1002
	7456	1D00
	7458	1700
	745A	C103
	745C	0604
	745E	16FE
	7460	0B02
JJ	7462	0605
	7464	16F4
	7466	045B
KK	7FF0	4A4F
	7FF2	5554
	7FF4	2020
	7FF6	72CE
	7FF8	4A53
	7FFA	4554
	7FFC	2020
	7FFE	729E
LL	701C	7468
	701E	7FF0
	7020	0000
	7022	0000
	7024	0000
	7026	0000
	7028	0000