Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I requested permission to publish this from Texas Instruments (twice) but received no reply. It is presented here (for free) and this pdf version of the document is my copyright in much the same way as a photograph would be. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/Radar, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Radar). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

In order to generate this file, I need to scan the pages, split the double pages and remove any edge marks such as punch holes, clean up the pages, set the relevant pages to be all the same size and alignment. I then run Omnipage (OCR) to generate the searchable text and then generate the pdf file.

Hopefully after all that, I end up with a presentable file. If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you personally – I know that I would have liked to have found some of these files years ago – they would have saved me a lot of time !

Colin Hinson
In the village of Blunham, Bedfordshire.

TI RS232/PIO

DISASSEMBLED

COLIN HINSON
TIM MacEACHERN

INTERNATIONAL
SUPPLEMENT

IT
UG

TI
NS

FOREWORD

This is the first in what will be a continuing series of Supplements to INTERNATIONAL TI-LINES, monthly newsletter of the INTERNATIONAL TI USER GROUP (ITUG - formerly OXON TI USERS).

This Supplement is devoted entirely to two independently-produced disassemblies of the TI RS232/PIO DSR ROM (card version).

The first is by COLIN HINSON of ITUG, the second by TIM MacEACHERN of TINS (the TI USER GROUP of NOVA SCOTIA).

Both authors kindly consented to allow their work to be reproduced in this manner, but both point out that copyright of the original code still resides with TEXAS INSTRUMENTS, and that their work is intended only to assist study.

Why publish TWO disassemblies ?

Common sense dictates that if you want to learn about something, you consult more than one source of reference, to try and ensure that you obtain as complete a picture as possible. I believe that both Colin and Tim have provided us with as complete a picture as is possible under current circumstances, when information "from the horse's mouth" is so difficult to obtain.

This work will be of interest to a number of 99 owners for different reasons. If you have an interest in the operation of the RS232/PIO card for the purposes of direct control (perhaps for Comms work, or Robotics) or if you want to exercise better control over your printer (with the emulation of "true Centronics" in mind), then this Supplement will suit your purposes admirably.

If you are teaching yourself 99ØØ Assembler, then this is an excellent opportunity to examine a practical application. You can see how sections of 99ØØ Source code have been put together, with explanatory remarks from both authors indicating the purpose, or what is believed to be the purpose, of each particular section.

Finally, it is always worth examining the work of someone else, in order to learn by their example/mistakes (referring to TI's code), and even if you do not presently have an expanded system, you might consider putting this Supplement to one side against the day that you eventually get into this most fascinating area of operation.

ACKNOWLEDGEMENTS

I would like to express my thanks to PAUL MEADOWS, expatriot Brit and president of the TI USER GROUP of NOVA SCOTIA, who initially obtained permission from TIM MacEACHERN for me to publish his work here. Thanks are also due to a number of other individuals who contributed towards this Supplement in one way or another: MARTIN ROSS, RICHARD SIERAKOWSKI, GORDON PITT, RICHARD BLANDEN, and JOHN MATTHEWS (all ITUGers).

Last, but by no means least, thanks to the two authors, Colin and Tim, who have carried out a tremendous amount of work in our interest.

TIM MacEACHERN's disassembly originally appeared as a special supplement to the TINS Newsletter in August 1985.

Peter Brooks
Oxford
July 1986

A PUBLICATION OF THE INTERNATIONAL TI USER GROUP

Tel (02357) 2360
WESSEX PRESS
9-11 Mill Street Wantage, Oxon OX12 9AB

PETER G. Q. BROOKS
96 BANBURY ROAD
OXFORD OX2 6JT

COLIN HINSON

```
*
       TITL 'DEVICE SERVICE ROUTINES FOR RS232 INTERFACE CARD
*
*   GENERAL NOTES:
*
*   THIS SOURCE CODE WAS GENERATED USING MY OWN 'LABELLING'
*   DIS-ASSEMBLER. WHEN LABELS ARE GENERATED, THEY APPEAR
*   SIMPLY AS THE ADDRESS WITH AN 'A' ON THE FRONT OF THEM.
*   AS A RESULT OF THIS, ANY LABELS WITH THIS FORM SHOULD
*   APPEAR AT THE APPROPRIATE ADDRESS WHEN ASSEMBLED. LABELS
*   WITH A FORM OTHER THAN THAT ABOVE HAVE BEEN EDITED IN TO
*   MAKE THE CODE MORE READABLE.  THE DIS-ASSEMBLER DOES NOT
*   OF COURSE GENERATE COMMENTS, THESE BEING EDITED IN LATER
*   BY ME. THIS MEANS OF COURSE THAT WHILST THE CODE IS
*   CORRECT, THE COMMENTS MAY NOT BE, SO IF YOU FIND ANY THAT
*   AREN'T, PLEASE LET ME KNOW.  NOTE THAT THIS FILE HAS
*   BEEN RE-ASSEMBLED, AND THE RESULTING OBJECT VERIFIED
*   AGAINST THAT IN THE ORIGINAL DSR ROM.
*                                 COLIN HINSON  11/5/86
*
*       IN GENERAL, THE SAME ROUTINES ARE USED FOR BOTH THE
*    PIO AND THE RS232. A FLAG (R3) IS USED TO DIRECT THE
*    PROGRAM FLOW WHERE THIS DIFFERS BETWEEN THE TWO.  R3 IS
*    ZERO FOR RS232, AND NOT ZERO FOR PIO.
*
*       AS THE PERIPHERAL BOARDS WERE DESIGNED TO WORK WITH
*    FUTURE CONSOLES WHOSE INTERNAL SCRATCH PAD RAM MIGHT MOVE
*    FROM >83ØØ, NONE OF THE DSR'S MAKE DIRECT REFERENCE TO
*    RAM BEING AT >83ØØ.  ALL RAM IS REFERENCED WITH RESPECT
*    TO THE WORKSPACE POINTER (THIS BEING ASSUMED TO BE AT
*    >EØ WITH RESPECT TO THE SCRATCH PAD BASE ADDRESS). THE
*    REGISTER USED FOR THIS PURPOSE IS R4, ALL RAM BEING
*    ACCESSED VIA R4 AND THE APPROPRIATE INDEX.  IN ORDER TO
*    MAKE THE PROGRAM MORE READABLE, THE INDEXES ARE TAKEN
*    WITH RESPECT TO THE BASE ADDRESS IN A POSITIVE MANNER
*    I.E.  TO ADDRESS PAD+>2Ø (>832Ø IN A 99/4A), THE NORMAL
*    INSTRUCTION WOULD BE @>FF4Ø(R4).  IN THIS PROGRAM THIS
*    IS WRITTEN AS @PAD+>2Ø, WHERE PAD IS EQUATED TO ->EØ.
*    (IT ALL WORKS OUT IN THE END, AS THE ASSEMBLER TAKES CARE
*    OF IT)
*
*    ABBREVIATIONS USED:
*     TX = TRANSMIT              RX = RECEIVE
*     PTR = POINTER              C.RETN = CARRIAGE RETURN
*     LF  = LINE FEED            DC2 = DEVICE CONTROL 2 (>12)
*     CHAR. = CHARACTER          ADR.= ADDRESS
*     I/P = INPUT                O/P = OUTPUT
*     I/F = INTERFACE            REG = REGISTER
*     CRC = CYCLIC REDUNDANCY CHECK
*     SWS = SOFTWARE SWITCH
*
*   IT HAS BEEN ASSUMED THAT THE 'READER' IS AWARE OF THE
*   LAYOUT OF PAB'S FOR CALLING DSR'S, AND THE LAYOUT OF
*   ROM HEADERS ETC.
*
*  EQUATES
*
```

```
FF2Ø              PAD    EQU  ->EØ
ØØØØ              READ   EQU  Ø                  VDP READ MODE
4ØØØ              WRITE  EQU  >4ØØØ              VDP WRITE MODE BIT
                  *
FF6A              FAC    EQU  PAD+>4A
FF78              ECFLAG EQU  PAD+>58            ECHO CHARACTER FLAG
FF79              CRFLAG EQU  PAD+>59            CARRIAGE RETURN FLAG
FF7A              LFFLAG EQU  PAD+>5A            LINE FEED FLAG
FF7B              CHFLAG EQU  PAD+>5B            CHECK PARITY FLAG
FF7C              NUFLAG EQU  PAD+>5C            SEND NULLS AFTER CR FLAG
FF7D              CBFLAG EQU  PAD+>5D            CIRCULAR BUFFER FLAG
FF7E              PCOUNT EQU  PAD+>5E            PROGRAM FILE BYTE COUNT
FF8Ø              BCOUNT EQU  PAD+>6Ø            BLOCK BYTE COUNT
FF84              LEVEL1 EQU  PAD+>64            RETURN ADDRESS STORE 1
FF86              LEVEL2 EQU  PAD+>66            RETURN ADDRESS STORE 2
FF88              LEVEL3 EQU  PAD+>68            RETURN ADDRESS STORE 3
FF8A              LEVEL4 EQU  PAD+>6A            RETURN ADDRESS STORE 4
FF8C              LEVEL5 EQU  PAD+>6C            RETURN ADDRESS STORE 5
                  *
                  * START OF ROM AREA
                  *
4ØØØ                     AORG >4ØØØ
                  *
4ØØØ AA                  BYTE >AA                INDICATE VALID ROM
4ØØ1 Ø1                  BYTE >Ø1                VERSION Ø.1
4ØØ2 ØØØØ                DATA Ø                  NOT USED IN DSR ROMS
                  *
4ØØ4              BYTE4Ø EQU  $
4ØØ4 4Ø1Ø                DATA PWRLNK             POINTER TO POWER UP LINKS
4ØØ6 ØØØØ                DATA Ø                  NO LINKS FOR MENU
                  *
4ØØ8 4Ø16         X4ØØ8  DATA DSRLNK             POINTER TO DSR LINKS
4ØØA ØØØØ                DATA Ø
                  *
4ØØC 4Ø6C         X4ØØC  DATA INTLNK             POINTER TO INTERRUPT LINK(S)
4ØØE ØØØØ                DATA Ø                  NOT USED IN DSR ROMS
                  *
                  * POWER UP LINK
                  *
4Ø1Ø ØØØØ         PWRLNK DATA Ø                  NO FURTHER LINKS
4Ø12 4ØF4                DATA PWRUP
4Ø14 ØØØØ                DATA Ø
                  *
                  * DSR LINKS
                  *
4Ø16 4Ø2Ø         DSRLNK DATA DSRL1              POINTER TO NEXT LINK
4Ø18 416E                DATA RS232              ROUTINE POINTER
4Ø1A Ø5                  BYTE 5                  NAME LENGTH
4Ø1B 5253 3233           TEXT 'RS232'            DSR NAME
4Ø1F 32
                  *
4Ø2Ø 4Ø2C         DSRL1  DATA DSRL2              NEXT LINK
4Ø22 416E                DATA RS232              ROUTINE POINTER
4Ø24 Ø7                  BYTE 7                  NAME LENGTH
4Ø25 5253 3233           TEXT 'RS232/1'          DSR NAME
4Ø29 322F 31
                  *
4Ø2C 4Ø38         DSRL2  DATA DSRL3              NEXT LINK
```

```
4Ø2E 4174              DATA RS2322            ROUTINE POINTER
4Ø3Ø Ø7                BYTE 7                 NAME LENGTH
4Ø31 5253 3233         TEXT 'RS232/2'         DSR NAME
4Ø35 322F 32
                *
4Ø38 4Ø4Ø       DSRL3  DATA DSRL4             NEXT LINK
4Ø3A 415E              DATA PIO               ROUTINE POINTER
4Ø3C Ø3                BYTE 3                 NAME LENGTH
4Ø3D 5Ø49 4F           TEXT 'PIO'             DSR NAME
                *
4Ø4Ø 4Ø4A       DSRL4  DATA DSRL5             NEXT LINK
4Ø42 415E              DATA PIO               ROUTINE POINTER
4Ø44 Ø5                BYTE 5                 NAME LENGTH
4Ø45 5Ø49 4F2F         TEXT 'PIO/1'           DSR NAME
4Ø49 31
                *
4Ø4A 4Ø54       DSRL5  DATA DSRL6             NEXT LINK
4Ø4C 4164              DATA PIO2              ROUTINE POINTER
4Ø4E Ø5                BYTE 5                 NAME LENGTH
4Ø4F 5Ø49 4F2F         TEXT 'PIO/2'           DSR NAME
4Ø53 32
                *
4Ø54 4Ø6Ø       DSRL6  DATA DSRL7             NEXT LINK
4Ø56 418Ø              DATA RS2323
4Ø58 Ø7                BYTE 7                 NAME LENGTH
4Ø59 5253 3233         TEXT 'RS232/3'         DSR NAME
4Ø5D 322F 33
                *
4Ø6Ø ØØØØ       DSRL7  DATA Ø                 NO FURTHER DSR LINKS
4Ø62 417A              DATA RS2324            ROUTINE POINTER
4Ø64 Ø7                BYTE 7                 NAME LENGTH
4Ø65 5253 3233         TEXT 'RS232/4'         DSR NAME
4Ø69 322F 34
                *
                * INTERRUPT LINK
                *
4Ø6C ØØØØ       INTLNK DATA Ø                 NO FURTHER LINKS
4Ø6E 4ØØ2              DATA INTRPT            POINTER TO INT. ROUTINE
4Ø7Ø ØØØØ              DATA Ø                 NO NAME
                *
4Ø72 Ø8         BYTEØ8 BYTE 8
4Ø73 ØØ         BYTEØØ BYTE Ø
                *
4Ø74 Ø3Ø3       HXØ3Ø3 DATA >Ø3Ø3
                *
4Ø76 4543       SWSTB1  TEXT 'EC'              ECHO OFF
4Ø78 4512              DATA ECSWS             ROUTINE POINTER
                *
4Ø7A 4352              TEXT 'CR'              CAR. RETN./LINE FEED OFF
4Ø7C 4518              DATA CRSWS             ROUTINE POINTER
                *
4Ø7E 4C46              TEXT 'LF'              LINE FEED OFF
4Ø8Ø 451E              DATA LFSWS             ROUTINE POINTER
                *
4Ø82 4E55              TEXT 'NU'              NULLS
4Ø84 4524              DATA NUSWS
                *
4Ø86 4441              TEXT 'DA'              NUMBER OF DATA BITS
```

```
4Ø88 457Ø                   DATA DASWS
                     *
4Ø8A 4241            SWSTB2  TEXT 'BA'                BAUD RATE
4Ø8C 4536                   DATA BASWS
                     *
4Ø8E 5Ø41                   TEXT 'PA'                PARITY
4Ø9Ø 454Ø                   DATA PASWS
                     *
4Ø92 5457                   TEXT 'TW'                TWO STOP BITS
4Ø94 4596                   DATA TWSWS
                     *
4Ø96 4348                   TEXT 'CH'                CHECK PARITY
4Ø98 452A                   DATA CHSWS
4Ø9A ØØØØ                   DATA Ø                   NO FURTHER ROUTINES
                     *
                     * TABLE OF CPU CLOCK RATES
                     *
4Ø9C ØØ28 4ØB6       CLKTBL DATA >28,REGTB1          2.5MHZ
4ØAØ ØØ3Ø 4ØC4              DATA >3Ø,REGTB2          3.ØMHZ
4ØA4 ØØØØ                   DATA Ø                   END OF TABLE
                     *
4ØA1                 BYTE3Ø EQU  $-5
                     *
                     * BAUD RATE TABLE
                     *
4ØA6 ØØ6E Ø12C       BAUDS  DATA 11Ø,3ØØ,6ØØ,12ØØ,24ØØ,48ØØ,96ØØ
4ØAA Ø258 Ø4BØ
4ØAE Ø96Ø 12CØ
4ØB2 258Ø
4ØB4 ØØØØ                   DATA Ø                   END OF TABLE
                     *
4ØB3                 BYTE8Ø EQU  $-3                 (96ØØ = >258Ø)
                     *
                     * TABLE OF 99Ø2 TX/RX DATA RATE REGISTER VALUES
                     *
4ØB6 8563 8482       REGTB1 DATA >8563,>8482,>82Ø9,>15B,>8Ø82,>8Ø41,>2B
4ØBA 82Ø9 Ø15B
4ØBE 8Ø82 8Ø41
4ØC2 ØØ2B
4ØC4 85AA 849C       REGTB2 DATA >85AA,>849C,>8271,>1A1,>8Ø9C,>8Ø4E,>8Ø27
4ØC8 8271 Ø1A1
4ØCC 8Ø9C 8Ø4E
4ØDØ 8Ø27
                     *
                     *              INTERRUPT ROUTINE
                     * IT SHOULD BE NOTED THAT ENTRY TO HERE IS VIA A 'BL' AND
                     * THAT THE WORKSPACE IS THE GPLWKS (PAD+>EØ) AND NOT THE
                     *  INTERRUPT WORKSPACE AS WOULD BE EXPECTED.
                     *
4ØD2 Ø2A4            INTRPT STWP R4                  SAVE WORKSPACE POINTER
4ØD4 1DØ7                   SBO  7                   LIGHT THE L.E.D.
4ØD6 C14B                   MOV  R11,R5              SAVE THE RETURN ADDRESS
4ØD8 C18C                   MOV  R12,R6              SAVE CRU BASE ADDRESS
4ØDA Ø22C ØØ4Ø              AI   R12,>ØØ4Ø           POINT TO 99Ø2/1
4ØDE 1F1Ø                   TB   16                  RECEIVE INTERRUPT SET?
4ØEØ 1316                   JEQ  A41ØE               YES, DEAL WITH IT
4ØE2 1F1F                   TB   31                  THIS DEVICE INTERRUPTING?
4ØE4 13Ø6                   JEQ  RSTSIO              YES, RESET IT (SHOULDN'T BE)
```

```
4ØE6 Ø22C ØØ4Ø          AI   R12,>ØØ4Ø          POINT TO 99Ø2/2
4ØEA 1F1Ø               TB   16                 RECEIVE INTERRUPT SET?
4ØEC 131Ø               JEQ  A41ØE              YES, DEAL WITH IT
4ØEE 1F1F               TB   31                 THIS DEVICE INTERRUPTING?
4ØFØ 1632               JNE  A4156              NO, JUMP
4ØF2 C3Ø6        RSTSIO MOV  R6,R12             RETRIEVE CRU BASE ADR.
                 *
                 * POWER UP ROUTINE (& 99Ø2/PIO RESET)
                 *
4ØF4 C18C        PWRUP  MOV  R12,R6             SAVE CRU BASE ADDRESS
4ØF6 1DØ7               SBO  7                  LIGHT THE LED
4ØF8 1DØ2               SBO  2                  SET PIO STROBE TO '1'
4ØFA 1EØ1               SBZ  1                  ENABLE THE PIO O/P DEVICE
4ØFC Ø22C ØØ4Ø          AI   R12,>ØØ4Ø          POINT TO TMS99Ø2/1
41ØØ 1D1F               SBO  31                 RESET IT
41Ø2 Ø22C ØØ4Ø          AI   R12,>ØØ4Ø          POINT TO TMS99Ø2/2
41Ø6 1D1F               SBO  31                 RESET IT
41Ø8 C3Ø6               MOV  R6,R12             RESTORE THE CRU BASE
41ØA 1EØ7               SBZ  7                  TURN THE LED OFF
41ØC Ø45B               RT                      RETURN TO CALLER
                 *
                 *         CIRCULAR INTERRUPT INPUT BUFFER.
                 *
                 *  NOTE: THIS BUFFER CANNOT BE USED FROM T.I. BASIC DUE
                 *  TO 'COMMON' RAM USE (LOCATIONS PAD THRU PAD+4)
                 *
                 *  THIS OPTION IS AN OPCODE WHICH IS ONLY ALLOWED BY THE
                 *  RS232 DSR. IT IS ENABLED BY CALLING THE DSR WITH AN
                 *  OPCODE OF >8Ø (OPEN OPCODE + MS BIT SET).  THIS CAUSES
                 *  THE NORMAL 'OPEN' COMMAND TO BE EXECUTED, BUT ENABLES
                 *  THE 99Ø2 RECEIVER INTERRUPTS AS WELL. RAM USAGE IS AS
                 *  FOLLOWS:
                 *         PAD   (2 BYTES)   START OF VDP BUFFER AREA
                 *         PAD+2 (1 BYTE)    BUFFER END ADR. OFFSET
                 *         PAD+3 (1 BYTE)    CALLER'S READ ADDR.
                 *         PAD+4 (1 BYTE)    RS232 DSR WRITE ADR.
                 *  WHEN A RX INTERRUPT IS RX'D, THE DSR TRIES TO STORE THE
                 *  INCOMING BYTE AT A VDP MEMORY ADDRESS DETERMINED BY
                 *  THE START ADR. OF THE BUFFER (PAD) PLUS THE WRITE ADR.
                 *  (PAD+4). IF THIS CANNOT BE DONE (THE WRITE ADDRESS HAS
                 *  CAUGHT UP WITH THE READ ADDRESS), THEN A BYTE OF >FE
                 *  IS WRITTEN OVER THE LAST CHARACTER RECEIVED. IF A RX
                 *  ERROR (PARITY ETC) OCCURS, THEN A BYTE OF >FF IS WRITTEN
                 *  INTO THE BUFFER. WHEN THE WRITE OFFSET = THE BUFFER END
                 *  OFFSET, THEN THE WRITE OFFSET IS SET TO ZERO, AND THUS
                 *  A RE-CIRCULATING BUFFER IS FORMED.
                 *     WHEN READING FROM THE BUFFER, A CHECK SHOULD BE DONE
                 *  TO SEE IF THERE IS ANY DATA IN THE BUFFER BY COMPARING
                 *  THE READ OFFSET WITH THE WRITE OFFSET. IF THEY ARE NOT
                 *  THE SAME THEN AT LEAST ONE BYTE IS PRESENT IN THE BUFFER.
                 *  HAVING READ A BYTE,(FROM BUFFER START + READ OFFSET)
                 *  THE READ OFFSET SHOULD BE INCREMENTED BY ONE. IF THE
                 *  RESULTING OFFSET IS GREATER THAN THE BUFFER END OFFSET
                 *  THEN THE READ OFFSET SHOULD BE SET TO ZERO.
                 *  IT SHOULD BE NOTED THAT ONLY THE 'SOFTWARE SWITCH'
                 *  OPTIONS WHICH AFFECT THE HARDWARE (DATA BITS, BAUD RATE
                 *  ETC) WILL AFFECT THIS MODE OF INPUT. SUCH OPTIONS AS
                 *  ECHO, CR, ETC WILL NOT.
```

```
                *
410E Ø6AØ 4874  A41ØE  BL   @SRXRDY             SERIAL RX CHAR. READY?
4112 1621              JNE  A4156               NO, JUMP
4114 DØ64 FF24         MOVB @PAD+4(R4),R1       GET BUFFER WRITE OFFSET
4118 BØ6Ø 45F9         AB   @BYTEØ1,R1          INCREMENT IT
411C 99Ø1 FF22         CB   R1,@PAD+2(R4)       REACHED END OF BUFFER?
412Ø 12Ø1              JLE  A4124               NO, OK
4122 Ø4C1              CLR  R1                  ELSE RESET TO START
4124 99Ø1 FF23  A4124  CB   R1,@PAD+3(R4)       WRITE PTR=READ PTR?
4128 13Ø6              JEQ  A4136               YES, OVER-RUN
412A 36Ø7              STCR R7,8                GET THE RECEIVED CHAR.
412C 1FØ9              TB   9                   ANY RX ERRORS?
412E 16Ø7              JNE  A413E               NO, JUMP
4132            BYTEFF EQU  $+2
413Ø Ø2Ø7 FFØØ         LI   R7,>FFØØ            SET UP INVALID CHAR
4134 1ØØ4              JMP  A413E               WRITE IT INTO BUFFER
                *
4136 Ø2Ø7 FEØØ  A4136  LI   R7,>FEØØ            SET UP THE 'OVERRUN' CHAR
413A DØ64 FF24         MOVB @PAD+4(R4),R1       GET THE LAST WRITE ADR.
                *
413E D9Ø1 FF24  A413E  MOVB R1,@PAD+4(R4)       SET UP NEW WRITE ADR.
4142 Ø981              SRL  R1,8                OFFSET TO LSB
4144 AØ64 FF2Ø         A    @PAD(R4),R1         ADD THE BUFFER START
4148 Ø241 3FFF         ANDI R1,>3FFF            REMOVE SURPLUS BITS
414C Ø6AØ 484E         BL   @SETADR             SET UP TO WRITE TO VDP
415Ø 4ØØØ              DATA WRITE
4152 D8C7 FFFE         MOVB R7,@-2(R15)         WRITE BYTE TO THE VDP
4156 1Ø12       A4156  SBO  18                  ENABLE THE RX INTERRUPTS
4158 C3Ø6              MOV  R6,R12              RESTORE THE CRU BASE
415A 1EØ7              SBZ  7                   TURN THE LED OFF
415C Ø455              B    *R5                 RETURN TO CALLER
                *
                * PIO ROUTINE ENTRIES
                *
415E Ø2Ø6 ØØØ1  PIO    LI   R6,1                BOARD NO. 1
4162 1ØØ2              JMP  A4168
                *
4164 Ø2Ø6 ØØØ2  PIO2   LI   R6,2                BOARD NO. 2
4168 Ø7Ø3       A4168  SETO R3                  SET PARALLEL FLAG
416A Ø4C2              CLR  R2
416C 1Ø11              JMP  A419Ø
                *
                * RS232 ROUTINES ENTRIES
                *
416E Ø2Ø6 ØØØ1  RS232  LI   R6,1                BOARD NO. 1
4172 1ØØ8              JMP  A4184
                *
4174 Ø2Ø6 ØØØ1  RS2322 LI   R6,1                BOARD NO. 1
4178 1ØØ8              JMP  A418A
                *
417A Ø2Ø6 ØØØ2  RS2324 LI   R6,2                BOARD NO. 2
417E 1ØØ5              JMP  A418A
                *
418Ø Ø2Ø6 ØØØ2  RS2323 LI   R6,2                BOARD NO. 2
4184 Ø2Ø2 ØØ4Ø  A4184  LI   R2,>4Ø              CRU BASE OFFSET OF 99Ø2/1
4188 1ØØ2              JMP  A418E
                *
418A Ø2Ø2 ØØ8Ø  A418A  LI   R2,>8Ø              CRU BASE OFFSET OF 99Ø2/2
```

O

```
 418E Ø4C3       A418E  CLR  R3                   CLEAR PARALLEL FLAG
 419Ø Ø2A4       A419Ø  STWP R4                   SAVE THE PAD OFFSET FOR INDEXES
 4192 C9ØB FF84         MOV  R11,@LEVEL1(R4)      SAVE THE RETURN ADR.
 4196 8181              C    R1,R6                OPERATION FOR THIS BOARD?
 4198 13Ø2              JEQ  A419E                YES, JUMP
 419A Ø46Ø 448Ø         B    @A448Ø               ELSE EXIT AND TRY NEXT.
                 *
                 *  CLEAR THE SOFTWARE SWITCH FLAG AREA (TO DEFAULT STATE)
                 *
 419E C184       A419E  MOV  R4,R6                GET WORKSPACE ADDRESS
 41AØ Ø226 FF78         AI   R6,ECFLAG            POINT TO FLAG AREA
 41A7            BYTEØ6 EQU  $+3
 41A4 Ø2Ø5 ØØØ6         LI   R5,6                 6 WORDS TO CLEAR
 41A8 Ø4F6       A41A8  CLR  *R6+
 41AA Ø6Ø5              DEC  R5                   UPDATE COUNT
 41AC 16FD              JNE  A41A8                LOOP TILL DONE
                 *
 41AE 1DØ7              SBO  7                    LIGHT THE LED
 41BØ A3Ø2              A    R2,R12               POINT TO THE REQUIRED 99Ø2
 41B2 Ø6AØ 4842         BL   @SETPAB              SET UP TO READ THE PAB
 41B6 ØØØØ              DATA READ                  FROM THE VDP RAM
                 *
                 *   READ 1Ø BYTES FROM THE VDP TO PAD RAM (STARTING AT FAC)
                 *  (THESE 1Ø BYTES ARE THE PAB UP TO AND INCLUDING THE NAME
                 *   LENGTH BYTE.)
                 *
 41B8 Ø2Ø5 ØØØA         LI   R5,1Ø                SET COUNT TO 1Ø
 41BC C184              MOV  R4,R6                GET WORKSPACE ADDRESS
 41BE Ø226 FF6A         AI   R6,FAC               ADD THE DESTINATION OFFSET
 41C2 DDAF FBFE  A41C2  MOVB @>FBFE(R15),*R6+     READ A BYTE FROM THE VDP
 41C6 Ø6Ø5              DEC  R5                   COUNT THE BYTE
 41C8 16FC              JNE  A41C2                LOOP TILL DONE
                 *
 41CA 592Ø 46ØB         SZCB @BYTEEØ,@FAC+1(R4)  CLEAR THE ERROR BITS
 41CE FF6B
 41DØ 992Ø 4ØB3         CB   @BYTE8Ø,@FAC(R4)    OPEN CIRCULAR BUFFER?
 41D4 FF6A
 41D6 16Ø6              JNE  A41E4                NO, JUMP
 41D8 F92Ø 4132         SOCB @BYTEFF,@CBFLAG(R4)  SET THE CIRC. BUFFER FLAG
 41DC FF7D
 41DE 592Ø 4ØB3         SZCB @BYTE8Ø,@FAC(R4)    REMOVE THE MSB OF OPCODE
 41E2 FF6A
 41E4 9824 FF6A  A41E4  CB   @FAC(R4),@BYTEØ6    IS IT A VALID OPCODE?
 41E8 41A7
 41EA 12Ø2              JLE  A41FØ                YES, JUMP
 41EC Ø46Ø 445Ø         B    @ERROR3              ELSE ERROR IT
                 *
 41FØ Ø6AØ 449Ø  A41FØ  BL   @DOSWS               DO THE SOFTWARE SWITCHING
 41F4 D164 FF6A         MOVB @FAC(R4),R5          GET THE OPCODE FROM PAB
 41F8 Ø985              SRL  R5,8                 MOVE TO LSB
 41FA ØA15              SLA  R5,1                 WORD ALIGN
 41FC C165 42Ø2         MOV  @BTABLE(R5),R5       GET THE ROUTINE ADDRESS
 42ØØ Ø455              B    *R5                  AND BRANCH TO IT
                 *
                 * BRANCH TABLE
                 *
 42Ø2 421Ø 4464  BTABLE DATA OPENF,CLOSEF,READF,WRITEF,ERROR3,LOADF,SAVEF
 42Ø6 4236 42FA
 42ØA 445Ø 4338
 42ØE 43D2
                *
                *
                *             OPEN A FILE FOR THE I/O
                *
                *  THE ONLY INVALID ATTRIBUTE FOR 'OPEN' IS SPECIFYING
                *  A RELATIVE RECORD FILE.
                *
 421Ø DØA4 FF6E OPENF  MOVB @FAC+4(R4),R2      GET THE LOGICAL RECORD LENGTH
 4214 16Ø9             JNE  A4228              IF NOT ZERO, JUMP
                *                               (IF ZERO, USE DEFAULT OF 8Ø)
 4216 Ø6AØ 4842        BL   @SETPAB            SET UP TO WRITE TO PAB + 4
 421A 4ØØ4             DATA WRITE+4
 421C Ø2Ø2 5ØØØ        LI   R2,8Ø*256          DEFAULT TO RECORD LENGTH = 8Ø
 422Ø D9Ø2 FF6E        MOVB R2,@FAC+4(R4)      INSERT INTO CPU RAM PAB
 4224 DBC2 FFFE        MOVB R2,@-2(R15)        AND INTO VDP RAM PAB
 4228 DØ64 FF68 A4228  MOVB @FAC+1(R4),R1      GET THE FILE TYPE
 422C           BYTE2Ø EQU  $                  WHAT A BAD LABEL!
 422C 2Ø6Ø 43CA        COC  @HXØ1ØØ,R1         IS IT A SEQUENTIAL ILE?
 423Ø 1663             JNE  A42F8              YES, OK
 4232 Ø46Ø 444A        B    @ERROR2            CAN'T OPEN REL. REC FILE.
                *
                *        READ A FILE FROM THE I/O TO THE VDP BUFFER
                *
                * IF 'INTERNAL' TYPE FILE IS SPECIFIED, THEN THE FIRST
                * BYTE READ IS TAKEN TO BE THE BYTE COUNT OF THE REMAINING
                * BYTES TO BE READ. ONCE THIS BYTE IS READ, THE DSR
                * WILL FUNCTION AS IF 'FIXED' LENGTH HAD BEEN SPECIFIED.
                * THIS BYTE COUNT IS TRANSPARENT TO THE USER, N+1 BYTES
                * ARE RECEIVED, BUT ONLY THE LAST N BYTES ARE PASSED TO
                * THE CALLER.
                *
 4236 Ø743      READF  ABS  R3
 4238 592Ø 4132        SZCB @BYTEFF,@FAC+5(R4)  CLEAR THE CHARACTER COUNT
 423C FF6F
 423E D1E4 FF6E        MOVB @FAC+4(R4),R7      GET THE RECORD LENGTH
 4242 C264 FF6C        MOV  @FAC+2(R4),R9      GET THE DATA BUFFER ADR. (VDP)
 4246 Ø6AØ 474Ø        BL   @INTRNL            IS IT AN 'INTERNAL' FILE?
 424A 16Ø7             JNE  A425A              NO, JUMP
 424C Ø6AØ 463A        BL   @GETCHR            GET THE BYTE COUNT
 425Ø 9187             CB   R7,R6              IS THE COUNT REASONABLE?
 4252 14Ø2             JHE  A4258              YES, OK
 4254 Ø46Ø 4456        B    @ERROR4            IF NOT, ERROR IT
 4258 C1C6      A4258  MOV  R6,R7              RESET THE LENGTH
 425A Ø987      A425A  SRL  R7,8               MOVE TO LSB
 425C 1348             JEQ  A42EE              IF ZERO, NO CHARS - JUMP
 425E Ø6AØ 463A A425E  BL   @GETCHR            GET A CHAR FROM I/O (IN R6)
 4262 Ø6AØ 474Ø        BL   @INTRNL            INTERNAL DATA TYPE?
 4266 133A             JEQ  A42DC              YES, WRITE CHAR. TO BUFFER
 4268 DØ64 FF78        MOVB @ECFLAG(R4),R1     ECHO ON?
 426C 13Ø7             JEQ  A427C              YES, CHECK FOR C. RETN ETC
 426E Ø6AØ 474A        BL   @TSTFIX            FIXED RECORD LENGTH?
 4272 1334             JEQ  A42DC              YES, WRITE CHAR. TO BUFFER
 4274 Ø286 ØDØØ        CI   R6,>ØDØØ           CARRIAGE RETURN?
 4278 1631             JNE  A42DC              NO, WRITE CHAR TO BUFER
 427A 1Ø39             JMP  A42EE              YES, CLOSE FILE & EXIT
```

```
                  *
                  * CHARACTER RECEIVED, ECHO IS ON, FILE IS 'DISPLAY' TYPE
                  *
427C Ø286 ØDØØ    A427C  CI    R6,>ØDØØ          CARRIAGE RETURN RXD?
428Ø 1325                JEQ   A42CC             YES
4282 Ø286 7FØØ           CI    R6,>7FØØ          DEL RECEIVED?
4286 1312                JEQ   A42AC             YES, DELETE LAST CHAR
4288 Ø286 12ØØ           CI    R6,>12ØØ          DC2 RECEIVED?
428C 1625                JNE   A42D8             NO
                  *
                  *        A DC2 (PLAYBACK ON) HAS BEEN RECIEVED
                  *
                  *  THIS IS NORMALLY USED BY THE SENDER WHEN CHARACTERS
                  *  HAVE BEEN DELETED FROM THE BUFFER BY USE OF 'DEL'
                  *  AND THE SENDER IS NOT SURE WHAT REMAINS IN THE BUFFER.
                  *  RECEIPT OF THE 'DC2' CAUSES THE DSR TO O/P A C.RETN
                  *  LINE FEED, FOLLOWED BY THE CURRENT CONTENTS OF THE
                  *  RECEIVE BUFFER.
                  *
428E CØ64 FF6C           MOV   @FAC+2(R4),R1     GET THE VDP BUFFER START ADR.
4292 Ø6AØ 485Ø           BL    @SETRDA           SET VDP UP TO READ FROM THERE
4296 Ø6AØ 46EE           BL    @LINEND           SEND C.RETN, LF ETC
429A CØ89                MOV   R9,R2             GET THE END OF BUFFER PTR.
429C 6ØA4 FF6C           S     @FAC+2(R4),R2     COMPUTE NO. OF CHARS TO SEND
42AØ 1ØØ3                JMP   A42A8             AND GO SEND THEM
                  *
42A2 Ø6AØ 47DE    A42A2  BL    @TXVCHR           SEND A CHAR. FROM THE BUFFER
42A6 Ø6Ø2                DEC   R2                COUNT IT
42A8 16FC         A42A8  JNE   A42A2             LOOP TILL ALL SENT
42AA 1ØØ9                JMP   A425E             AND GO BACK TO RX
                  *
                  *     A DELETE (DEL = >7F) CHARACTER HAS BEEN RECEIVED
                  *
                  *  THE LAST CHAR IN THE BUFFER IS DELETED, AND THAT CHAR.
                  *  IS ECHOED TO THE SENDING TERMINAL.  IF MORE THAN ONE
                  *  DEL IS SENT, THEN THIS WILL RESULT IN THE CHARACTERS
                  *  APPEARING ON THE SENDING TERMINAL IN THE REVERSE ORDER
                  *  TO THAT IN WHICH THEY WERE SENT.  IF THE BUFFER IS
                  *  EMPTY, THEN NO ACTION IS TAKEN.
                  *
42AC 8264 FF6C    A42AC  C     @FAC+2(R4),R9     BUFFER EMPTY?
42BØ 13D6                JEQ   A425E             YES, GO BACK TO RX
42B2 Ø587                INC   R7                UN-COUNT THE CHARACTER
42B4 Ø6Ø9                DEC   R9                BACK OFF THE BUFFER PTR
42B6 CØ49                MOV   R9,R1             SET UP TO READ VDP RAM
42B8 Ø6AØ 485Ø           BL    @SETRDA
42BC Ø6AØ 47DE           BL    @TXVCHR           SEND THE CHAR. POINTED TO
42CØ Ø286 ØDØØ           CI    R6,>ØDØØ          WAS IT A C.RETN?
42C4 16CC                JNE   A425E             NO, GO RX NEXT CHAR
42C6 Ø6AØ 47ØØ           BL    @A47ØØ            YES, GO SORT OUT NULLS ETC
42CA 1ØC9                JMP   A425E             AND THEN RX NEXT CHAR
                  *
                  *  A CARRIAGE RETURN WAS RX'D, AND WE ARE IN ECHO MODE
                  *
42CC Ø6AØ 474A    A42CC  BL    @TSTFIX           FIXED RECORD LENGTH?
42DØ 13Ø3                JEQ   A42D8             YES, JUST ECHO IT
42D2 Ø6AØ 46EE           BL    @LINEND           NO, SORT OUT LF, NULLS ETC
42D6 1ØØB                JMP   A42EE             THEN FINISH OFF
```

```
                  *
                  * CHARACTER RECEIVED WHILST IN ECHO MODE
                  *
4208 Ø6AØ 47E6    A42D8  BL    @SENDR6             ECHO THE CHARACTER
42DC              BYTECØ EQU   $                   WHAT A BAD EQUATE!
                  *
                  *  CHARACTER RECEIVED, BUT NOT IN ECHO MODE
                  *
42DC CØ49         A42DC  MOV   R9,R1               GET CURRENT WRITE ADDRESS
42DE Ø6AØ 484E           BL    @SETADR             SET IT UP AS VDP ADR.
42E2 4ØØØ                DATA  WRITE
42E4 DBC6 FFFE           MOVB  R6,@-2(R15)         WRITE RX'D CHAR. TO BUFFER
42E8 Ø589                INC   R9                  UPDATE BUFFER PTR
42EA Ø6Ø7                DEC   R7                  COUNT THE CHARACTER
42EC 16B8                JNE   A425E               LOOP TILL ALL RECORD RX'D
                  *
42EE 6264 FF6C    A42EE  S     @FAC+2(R4),R9       COMPUTE ACTUAL QTY OF CHARS
42F2 ØA89                SLA   R9,8                MOVE TO LSB
42F4 D9Ø9 FF6F           MOVB  R9,@FAC+5(R4)       PASS INFO. BACK TO USER
42F8 1Ø1D         A42F8  JMP   A4334               CLOSE FILE & EXIT
                  *
                  *        WRITE A FILE TO THE I/O PORT REQUESTED
                  *
                  *  IF AN 'INTERNAL' DATA FILE HAS BEEN REQUESTED, THEN
                  *  THE FIRST BYTE TRANSMITTED IS A CHARACTER COUNT OF
                  *  THE ACTUAL NUMBER OF BYTES IN THE RECORD. AS A
                  *  RESULT OF THIS, N+1 BYTES ARE TX'S FOR A N BYTE
                  *  RECORD. (SEE ALSO NOTE AT BEGINNING OF 'READF'.
                  *
42FA CØC3         WRITEF MOV   R3,R3               IN SERIAL MODE?
42FC 13Ø1                JEQ   A43ØØ               YES, JUMP
42FE Ø7Ø3                SETO  R3                  SET PARALLEL FLAG NEGATIVE
43ØØ CØ64 FF6C    A43ØØ  MOV   @FAC+2(R4),R1       GET VDP BUFFER ADDRESS
43Ø4 Ø6AØ 485Ø           BL    @SETRDA             SET UP VDP TO READ FROM THERE
43Ø8 D1E4 FF6F           MOVB  @FAC+5(R4),R7       GET THE CHAR COUNT FROM PAB
43ØC Ø6AØ 474Ø           BL    @INTRNL             INTERNAL DATA FILE?
431Ø 16Ø3                JNE   A4318               NO, JUMP
4312 C187                MOV   R7,R6               CHAR. COUNT TO R6
4314 Ø6AØ 47E6           BL    @SENDR6             SEND CHAR. COUNT TO TERMINAL
4318 Ø987         A4318  SRL   R7,8                MOVE COUNT TO LSB
431A 13Ø4                JEQ   A4324               IF ZERO, JUMP
431C Ø6AØ 47ØE    A431C  BL    @TXVCHR             READ A CHAR AND SEND IT
432Ø Ø6Ø7                DEC   R7                  UPDATE COUNTER
4322 16FC                JNE   A431C               LOOP TILL ZERO
4324 Ø6AØ 474Ø    A4324  BL    @INTRNL             INTERNAL DATA FILE?
4328 13Ø5                JEQ   A4334               YES, JUMP
432A Ø6AØ 474A           BL    @TSTFIX             FIXED RECORD LENGTH?
432E 13Ø2                JEQ   A4334               YES, EXIT
433Ø Ø6AØ 46EE           BL    @LINEND             DEAL WITH LINE END (C.R. ETC)
4334 Ø46Ø 4464    A4334  B     @CLOSEF             CLOSE FILE AND EXIT
                  *
                  *    LOAD A FILE INTO THE VDP BUFFER FROM THE I/O
                  *            (ENTRY VIA BRANCH TABLE)
                  *
                  *  WHEN LOADing OR SAVEing THROUGH THE RS232 I/F OVER
                  *  MODEMS, HANDSHAKING IS INVOLVED. THIS DSR TAKES EACH
                  *  DATA BLOCK AND OUTPUTS THEM ONE BLOCK AT A TIME.  THE
                  *  HANDSHAKING STARTS WITH THE 'LOAD' PART SENDING A 'SYN'
                  *  (>16) EVERY 7 SECONDS TO THE SENDERS 'SAVE' PART WHICH
```

```
                *  WATCHES FOR THIS CHAR.  AS SOON AS THE 'SAVE' SEES THE
                *  SYN CHAR, IT STARTS TRANSMITTING THE BLOCKS AS FOLLOWS:
                *     2 BYTES PROGRAM BYTE COUNT
                *     2 BYTES CRC CHECK CODE (OF PROGRAM BYTE COUNT)
                *     N BYTES OF DATA BLOCK (UP TO 256)
                *     1 BYTE OF CRC CHECK FOR DATA BYTES IN BLOCK
                *  AS MANY 256 BYTE BLOCKS AS REQUIRED ARE TRANSMITTED IN
                *  THIS FORMAT UNTIL THE WHOLE FILE HAS BEEN SENT. (THE
                *  LAST BLOCK MAY BE OF VARIABLE LENGTH)
                *    IF THE 'LOAD' PART DOES NOT RECEIVE A GOOD CRC ON THE
                * BYTE COUNT, THEN IT SENDS A 'NAK' (>15), OTHERWISE IT
                * WAITS FOR THE DATA TO BE SENT.  AT THE END OF EACH DATA
                * BLOCK, THE 'LOAD' PART WILL RESPOND WITH EITHER A 'ACK'
                * (>06) OR A 'NAK', WITH THE 'SAVE' PART WAITING FOR THIS
                * RESPONSE BEFORE SENDING THE NEXT BLOCK.  IF A 'HANG-UP'
                * SHOULD OCCUR, THE CONDITION CAN BE CLEARED IN THE NORMAL
                * WAY BY PRESSING 'CLEAR' (FUNCTION 3). THIS WILL OF COURSE
                * ABORT THE WHOLE FILE I/O PROCESS AND PASS AN ERROR BACK
                * TO THE CALLER.
                *
                *  NOTE: THE EXIT FROM THIS ROUTINE IS VIA 'SETCNT' -
                *    WHEN THE 'PCOUNT' VALUE REACHES ZERO, IT EXITS
                *    VIA 'CLOSEF'.
                *
 4338 C024 FF70 LOADF  MOV  @FAC+6(R4),R0     GET THE CHARACTER COUNT
 433C 06A0 47E4 A433C  BL   @TXDATA           SEND SYNCRONIZATION BYTE
 4340 1600             DATA >1600
 4342 0205 0007        LI   R5,7              SET UP OUTER LOOP TO 7 SECS.
 4346 0201 C01C A4346  LI   R1,>C01C          SET UP INNER LOOP TO 1 SEC.
 434A 06A0 4870 A434A  BL   @CHARDY           CHAR. READY?
 434E 1307             JEQ  A435E             YES, EXIT LOOP
 4350 0601             DEC  R1                UPDATE INNER LOOP
 4352 16FB             JNE  A434A             NOT ZERO, LOOP BACK
 4354 06A0 4880        BL   @TSTCLR           TEST THE CLEAR KEY
 4358 0605             DEC  R5                UPDATE OUTER LOOP
 435A 16F5             JNE  A4346             NOT ZERO, LOOP BACK
 435C 10EF             JMP  A433C             IF ZERO, RE-SEND SYNCH BYTE
                *
                *    A CHARACTER HAS BEEN RECEIVED BY THE HARDWARE.
                *  THE FIRST TWO CHARS. SHOULD BE THE 16 BIT PROGRAM BYTE
                *  COUNT, FOLLOWED BY A 2 BYTE CYCLIC REDUNDANCY CHECK
                *  WORD (2 BYTES) FOR THE PROGRAM BYTE COUNT WORD.
                *
 435E 0709      A435E  SETO R9                PRE-SET FOR CRC START
 4360 06A0 45C6        BL   @CRXCRC           GET CHAR, DO C.R.C. CHECK
 4364 C1C6             MOV  R6,R7             SAVE CHAR.
 4366 06A0 45C6        BL   @CRXCRC           GET NEXT CHAR, DO C.R.C
 436A 0986             SRL  R6,8              MOVE TO LSB
 436C E1C6             SOC  R6,R7             ADD PREVIOUS CHAR TO MSB (R7)
 436E 06A0 45A0        BL   @GET2CH           GET BYTE COUNT CHECK (IN R8)
 4372 06A0 4684        BL   @WR7DEC           WRITE R7 IN DECIMAL TO SCREEN
 4376 8248             C    R8,R9             BYTE COUNT CRC = CHECK CODE?
 4378 1304             JEQ  A4382             YES, CONTINUE
 437A 06A0 47E4        BL   @TXDATA           NO, SEND 'NAK'
 437E 1500             DATA >1500
 4380 10EE             JMP  A435E             AND WAIT FOR NEXT BYTE COUNT
                *
                * BYTE COUNT HAS BEEN RECEIVED CORRECTLY (IT'S IN R7)
```

```
                   *
4382 81CØ          A4382  C     RØ,R7                IS THE BYTE COUNT REASONABLE?
4384 1A68                 JL    ERROR4               NO, ERROR IT
4386 Ø6AØ 47E4            BL    @TXDATA              YES, SEND 'ACK'
438A Ø6ØØ                 DATA >6ØØ
                   *
438C Ø6AØ 4686     A438C  BL    @SETCNT              SET UP BLOCK COUNT (IN R7)
439Ø Ø7Ø9          A439Ø  SETO R9                    PRESET CRC WORD
4392 CØ4A                 MOV   R1Ø,R1               GET CURRENT DATA BUFFER ADR.
4394 Ø6AØ 484E            BL    @SETADR              SET UP VDP TO WRITE THERE
4398 4ØØØ                 DATA WRITE
439A Ø6AØ 45C6     A439A  BL    @CRXCRC              GET A CHAR, DO CRC ON IT
439E DBC6 FFFE            MOVB R6,@-2(R15)           WRITE CHAR. TO VDP BUFFER
43A2 Ø6Ø7                 DEC   R7                   UPDATE BYTE COUNTER
43A4 16FA                 JNE   A439A                NOT ZERO, LOOP
43A6 Ø6AØ 45AØ            BL    @GET2CH              GET LAST 2 CHARS (CRC WORD)
43AA CØC3                 MOV   R3,R3                IN SERIAL MODE?
43AC 13Ø2                 JEQ   A43B2                YES, JUMP
43AE Ø6AØ 48A2            BL    @WAIT                WAIT
43B2 82Ø9          A43B2  C     R9,R8                CRC WORD = COMPUTED CRC?
43B4 13Ø6                 JEQ   A43C2                YES, GO SEND 'ACK'
43B6 C1E4 FF8Ø            MOV   @BCOUNT(R4),R7     ELSE GET THE START COUNT BACK
43BA Ø6AØ 47E4            BL    @TXDATA              SEND A 'NAK'
43BE 15ØØ                 DATA >15ØØ
43CØ 1ØE7                 JMP   A439Ø                AND TRY AGAIN.
                   *
43C2 Ø6AØ 47E4     A43C2  BL    @TXDATA              SEND 'ACK'
43C6 Ø6ØØ                 DATA >6ØØ
43CA               HXØ1ØØ EQU   $+2
43C8 Ø22A Ø1ØØ            AI    R1Ø,256              UPDATE VDP BUFFER ADDRESS
43CC C1E4 FF7E            MOV   @PCOUNT(R4),R7       GET THE NEW FILE LENGTH
43DØ 1ØDD                 JMP   A438C                AND GO GET NEXT BLOCK
                   *
                   *  SAVE A FILE TO THE I/O PORT AS REQUESTED
                   *        (ENTRY VIA BRANCH TABLE)
                   *
43D2 CØ4A          SAVEF  MOV   R1Ø,R1               GET VDP BUFFER ADDRESS
43D4 Ø6AØ 485Ø            BL    @SETRDA              SET VDP TO READ FROM THERE
43D8 Ø6AØ 463A     A43D8  BL    @GETCHR              GET CHAR FROM THE I/O
43DC Ø286 16ØØ            CI    R6,>16ØØ             IS IT A 'SYNC' CHAR?
43EØ 16F8                 JNE   A43D8                NO, LOOP
43E2 Ø7Ø9          A43E2  SETO R9                    PRESET THE CRC WORD
43E4 CØC3                 MOV   R3,R3                IN SERIAL MODE?
43E6 13Ø2                 JEQ   A43EC                YES, JUMP
43E8 Ø6AØ 48A2            BL    @WAIT                WAIT
43EC C1A4 FF7Ø     A43EC  MOV   @FAC+6(R4),R6        GET THE CHARACTER COUNT
43FØ Ø6AØ 45DØ            BL    @CTXCRC              SEND R6 MSB, DO CRC ON IT
43F4 Ø6C6                 SWPB R6                    GET LSB
43F6 Ø6AØ 45DØ            BL    @CTXCRC              SEND LSB, DO CRC ON IT
43FA Ø6AØ 45B4            BL    @SNDCRC              SEND THE CRC WORD
43FE Ø6AØ 463A            BL    @GETCHR              GET CHARACTER FROM I/O
44Ø2 Ø286 Ø6ØØ            CI    R6,>Ø6ØØ             IS IT AN 'ACK'
44Ø6 16ED                 JNE   A43E2                NO, TRY AGAIN
44Ø8 C1E4 FF7Ø            MOV   @FAC+6(R4),R7        GET THE PROGRAM BYTE COUNT
44ØC Ø6AØ 4686     A44ØC  BL    @SETCNT              SET UP THE COUNTERS
441Ø Ø7Ø9          A441Ø  SETO R9                    PRESET THE CRC WORD
4412 CØ4A                 MOV   R1Ø,R1               GET THE VDP BUFFER ADR.
4414 Ø6AØ 485Ø            BL    @SETRDA              SET UP TO READ FROM IT
```

```
4418 D1AF FBFE  A4418  MOVB @>FBFE(R15),R6     READ A BYTE FROM BUFFER
441C Ø6AØ 45ØØ         BL   @CTXCRC            SEND THE BYTE, DO CRC
442Ø Ø6Ø7              DEC  R7                 UPDATE BLOCK COUNTER
4422 16FA              JNE  A4418              LOOP TILL ZERO
4424 Ø6AØ 4584         BL   @SNDCRC            TRANSMIT THE COMPUTED CRC
4428 Ø6AØ 463A         BL   @GETCHR            GET A CHAR FROM THE I/O
442C Ø286 Ø6ØØ         CI   R6,>Ø6ØØ           IS IT A 'ACK'
443Ø 13Ø7              JEQ  A444Ø              YES, OK - JUMP
                *
4432 CØC3              MOV  R3,R3              IN SERIAL MODE?
4434 13Ø2              JEQ  A443A              YES, JUMP
4436 Ø6AØ 48A2         BL   @WAIT              WAIT
443A C1E4 FF8Ø  A443A  MOV  @BCOUNT(R4),R7     GET THE LAST BLOCK COUNT
443E 1ØE8              JMP  A441Ø              AND SEND THE BLOCK AGAIN
                *
444Ø Ø22A Ø1ØØ  A444Ø  AI   R1Ø,256            UPDATE THE BUFFER POINTER
4444 C1E4 FF7E         MOV  @PCOUNT(R4),R7     GET THE UPDATED FILE LENGTH
4448 1ØE1              JMP  A44ØC              AND SEND NEXT BLOCK
                *
                * ERROR ROUTINES.
                *
444A Ø2Ø1 4ØØØ  ERROR2 LI   R1,>4ØØØ           SET UP 'BAD OPEN ATTRIBUTE'
444E 1ØØ8              JMP  A446Ø
                *
                * A 'RESTORE' OPCODE COMES HERE (IT'S AN ERROR!)
                *
445Ø Ø2Ø1 6ØØØ  ERROR3 LI   R1,>6ØØØ           SET UP 'ILLEGAL OPERATION' CODE
4454 1ØØ5              JMP  A446Ø
                *
4456 Ø2Ø1 8ØØØ  ERROR4 LI   R1,>8ØØØ           SET UP 'OUT OF BUFFER SPACE'
445A 1ØØ2              JMP  A446Ø
                *
                * ENTRY TO HERE WHEN 'CLEAR' PRESSED DURING I/O
                *
445C Ø2Ø1 CØØØ  ERROR6 LI   R1,>CØØØ           SET UP 'DEVICE ERROR' CODE
446Ø F9Ø1 FF6B  A446Ø  SOCB R1,@FAC+1(R4)      ENTER ERROR CODE IN CPU RAM
                *
                * ENTRY VIA BRANCH TABLE
                *
4464 Ø6AØ 4842  CLOSEF BL   @SETPAB             SET UP TO WRITE
4468 4ØØ1              DATA WRITE+1            AT VDP PAB+1
446A DBE4 FF6B         MOVB @FAC+1(R4),@-2(R15)  WRITE STATUS BYTE
446E FFFE
447Ø Ø6AØ 4842         BL   @SETPAB             SET UP TO WRITE
4474 4ØØ5              DATA WRITE+5              AT VDP PAB+5
4476 DBE4 FF6F         MOVB @FAC+5(R4),@-2(R15)  CHARACTER COUNT TO PAB
447A FFFE
447C Ø5E4 FF84         INCT @LEVEL1(R4)
448Ø Ø24C FFØØ  A448Ø  ANDI R12,>FFØØ          ENSURE CRU POINTS AT DSR ROM
4484 C2E4 FF84         MOV  @LEVEL1(R4),R11
4488 1DØ2              SBO  2                  SET PIO STROBE TO 1
448A 1EØ1              SBZ  1                  ENABLE THE PIO O/P DEVICE
448C 1EØ7              SBZ  7                  TURN THE LED OFF
448E Ø45B              RT
                *
                * SET UP DEFAULT SOFTWARE SWITCH OPTIONS, THEN ALTER
                * THESE AS REQUIRED BY THE SWITCH OPTIONS IN THE PAB
                *
```

```
4490 C9Ø8 FF86  DOSWS  MOV  R11,@LEVEL2(R4)    SAVE RETURN
4494 Ø6AØ 473Ø         BL   @CHKLSV            IS IT A LOAD/SAVE OPCODE?
4498 13Ø5              JEQ  A44A4              YES, AVOID INVALID SWS'S
449A Ø2Ø8 4Ø76         LI   R8,SWSTB1          POINT TO START OF SWS TABLE
449E Ø2Ø1 B2ØØ         LI   R1,>B2ØØ           DEFAULT TO 1 STOP BIT
                *                               ODD PARITY, 7 DATA BITS
44A2 1ØØ4              JMP  A44AC
44A4 Ø2Ø8 4Ø8A  A44A4  LI   R8,SWSTB2          POINT TO SWS TABLE (2)
44A8 Ø2Ø1 83ØØ         LI   R1,>83ØØ           DEFAULT TO 1 STOP BIT
                *                               NO PARITY, 8 DATA BITS
44AC Ø2Ø5 Ø12C  A44AC  LI   R5,>Ø12C
44BØ C244              MOV  R4,R9              GET WORKSPACE ADDRESS
44B2 Ø229 FFFA         AI   R9,PAD+>DA         POINT TO SAVE AREA
44B6 D641              MOVB R1,*R9             SAVE THE DEFAULT OPTIONS
44B8 Ø6AØ 45F4         BL   @DOBAUD            SORT OUT THE BAUD RATES
44BC DØ24 FF73         MOVB @FAC+9(R4),RØ      GET NAME LENGTH
44CØ Ø98Ø              SRL  RØ,8               TO LSB
44C2 6Ø24 FF74         S    @PAD+>54(R4),RØ    SUBTRACT THE DSR NAME LENGTH
44C6 1217              JLE  SWSEND             IF ZERO FINISH
44C8 CØ64 FF76         MOV  @PAD+>56(R4),R1    GET POINTER TO SWS CHARS IN PAB
44CC Ø6AØ 485Ø         BL   @SETRDA            SET UP TO READ VDP FROM THERE
44DØ Ø7Ø6              SETO R6                 INIT COUNTER
                *
                * LOOP HERE WHILST DEALING WITH THE SOFTWARE SWITCH OPTIONS
                *
44D2 CØØØ       SWSLP  MOV  RØ,RØ              ANY CHARS LEFT IN PAB?
44D4 131Ø              JEQ  SWSEND             NO, FINISH OFF
44D6 Ø6AØ 4798         BL   @FINDCH            SEARCH FOR A DELIMITER ('.')
44DA 2EØØ              DATA '.'*256
44DC 13ØC              JEQ  SWSEND             NOT FOUND, FINISH OFF
44DE C1C8              MOV  R8,R7              GET TABLE POINTER
44EØ Ø986              SRL  R6,8               CHAR TO LSB
44E2 D1AF FBFE         MOVB @>FBFE(R15),R6     READ NEXT CHAR.
44E6 Ø6ØØ              DEC  RØ                 UPDATE CHARS LEFT IN PAB
44E8 Ø6C6              SWPB R6                 ALIGN CHARS. CORRECTLY
44EA CØ77       A44EA  MOV  *R7+,R1            GET ROUTINE ADR. FROM TABLE
44EC 1311              JEQ  ERR2LK             ZERO, END OF TABLE - ERROR
44EE CØ87              MOV  *R7+,R2            GET CHARS. FROM TABLE
44FØ 8181              C    R1,R6              SAME AS INPUT?
44F2 16FB              JNE  A44EA              NO, LOOP
44F4 Ø452              B    *R2                YES, BRANCH TO ROUTINE
                *
                * ALL SWITCH OPTIONS HAVE BEEN FOUND, LOAD UP CONTROL
                *  REGISTER IN 99Ø2 AS APPROPRIATE.
                *
44F6 DØ64 FF6A  SWSEND MOVB @FAC(R4),R1        IS IT AN 'OPEN' OPCODE?
44FA 13Ø7              JEQ  A45ØA              YES, SET UP CONTROL REGISTER
44FC Ø6AØ 473Ø         BL   @CHKLSV            IS IT LOAD/SAVE OPCODE?
45ØØ 16Ø6              JNE  A45ØE              NO, RETURN TO CALLER
45Ø2 Ø6AØ 4682         BL   @A4682             WRITE '255' TO SCREEN
45Ø6 C2A4 FF6C         MOV  @FAC+2(R4),R1Ø     GET BUFFER POINTER
45ØA Ø6AØ 4822  A45ØA  BL   @SETCTL            LOAD UP THE 99Ø2 CONTROL REG
45ØE 1Ø66       A45ØE  JMP  LVL2RT             AND RETURN TO CALLER
                *
451Ø 1Ø9C       ERR2LK JMP  ERROR2
                *
                * SOFTWARE SWITCH OPTION ENTRIES
                *
```

```
4512 Ø2Ø1 FF78  ECSWS  LI   R1,ECFLAG          POINT TO ECHO FLAG
4516 1ØØB              JMP  SETFGS
4518 Ø2Ø1 FF79  CRSWS  LI   R1,CRFLAG          POINT TO CR OFF FLAG
451C 1ØØ8              JMP  SETFGS
451E Ø2Ø1 FF7A  LFSWS  LI   R1,LFFLAG          POINT TO LF OFF FLAG
4522 1ØØ5              JMP  SETFGS
4524 Ø2Ø1 FF7C  NUSWS  LI   R1,NUFLAG          POINT TO NULLS FLAG
4528 1ØØ2              JMP  SETFGS
452A Ø2Ø1 FF7B  CHSWS  LI   R1,CHFLAG          POINT TO CHECK PARITY FLAG
452E AØ44       SETFGS A    R4,R1
453Ø F46Ø 4132         SOCB @BYTEFF,*R1        SET THE APPROPRIATE FLAG
4534 1Ø34              JMP  SWSLLK             AND LOOP FOR NEXT SWS
                *
4536 CØC3       BASWS  MOV  R3,R3              IN PARALLEL MODE?
4538 1632              JNE  SWSLLK             YES, DO NEXT SWS
453A Ø6AØ 45E2         BL   @A45E2             GET AND SET UP THE BAUD DATA
453E 1Ø2F              JMP  SWSLLK             AND DO NEXT SWS
                *
                *          PARITY SOFTWARE SWITCH ROUTINE
                *
                *  THE 99Ø2 CONTROL REGISTER BITS ARE STORED '*R9'. THE
                *  PARITY BITS ARE : XXPPXXXX WITHIN THAT BYTE, WHERE
                *  ØX=NO PARITY, 1Ø = EVEN, AND 11 = ODD PARITY.
                *
454Ø CØC3       PASWS  MOV  R3,R3              IN PARALLEL MODE?
4542 162D              JNE  SWSLLK             YES, DO NEXT SWS
4544 Ø6AØ 4798         BL   @FINDCH            GO FIND A '=' SIGN
4548 3DØØ              DATA '='*256
454A 13E2              JEQ  ERR2LK             NOT FOUND, EXIT
454C 566Ø 4ØA1         SZCB @BYTE3Ø,*R9        DEFAULT TO NO PARITY
455Ø Ø986              SRL  R6,8               CHARACTER TO LSB
4552 Ø286 ØØ4E         CI   R6,'N'             PARITY = 'N'ONE?
4556 1323              JEQ  SWSLLK             YES, DO NEXT SWS
4558 Ø286 ØØ45         CI   R6,'E'             PARITY ='E'VEN?
455C 13Ø6              JEQ  A456A              YES, JUMP
455E Ø286 ØØ4F         CI   R6,'O'             PARITY ='O'DD?
4562 16D6              JNE  ERR2LK             NO, ERROR IT
4564 F66Ø 4ØA1         SOCB @BYTE3Ø,*R9        SET ODD PARITY BITS
4568 1Ø1A              JMP  SWSLLK             AND DO NEXT SWS
456A F66Ø 422C  A456A  SOCB @BYTE2Ø,*R9        SET EVEN PARITY BITS
456E 1Ø17              JMP  SWSLLK             AND DO NEXT SWS
                *
                *   SET UP THE REQUESTED NUMBER OF DATA BITS
                *  NOTE: WHILST THE 99Ø2 CAN BE PROGRAMMED FOR 5
                *  6, 7 OR 8 DATA BITS, ONLY 7 AND 8 ARE ALLOWED
                *  IN THIS DSR ROM.
                *
457Ø CØC3       DASWS  MOV  R3,R3              IN PARALLEL MODE?
4572 1615              JNE  SWSLLK             YES, DO NEXT SWS
4574 Ø6AØ 4798         BL   @FINDCH            GO FIND AN '=' SIGN
4578 3DØØ              DATA '='*256
457A 13CA              JEQ  ERR2LK             NOT FOUND, ERROR IT
457C Ø6AØ 4754         BL   @ASCHEX            CONVERT DIGIT TO HEX
458Ø F66Ø 4Ø74         SOCB @HXØ3Ø3,*R9        DEFAULT TO DATA BITS = 8
4584 Ø225 FFF9         AI   R5,-7              SUBTRACT 7
4588 13Ø3              JEQ  A459Ø              IF NOW ZERO, SET UP FOR 7
458A Ø6Ø5              DEC  R5                 IF IT WAS GREATER THAN 8
458C 16C1              JNE  ERR2LK             ERROR IT
```

```
458E 1ØØ2                 JMP  A4594           ELSE SET UP
                *
459Ø 566Ø 45F9  A459Ø  SZCB @BYTEØ1,*R9      SET UP DATA BITS = 7
4594 1ØØ4       A4594  JMP  SWSLLK
                *
                *    SET UP THE REQUESTED NUMBER OF STOP BITS
                *
                *  NOTE: WHILST THE 99Ø2 CAN BE PROGRAMMED UP TO USE
                *   1 AND 1/2 STOP BITS, THIS IS NOT CATERED FOR HERE.
                *
4596 566Ø 42ØC  TWSWS  SZCB @BYTECØ,*R9      CLEAR STOP BIT DATA
459A F66Ø 4ØØ4         SOCB @BYTE4Ø,*R9      SET UP TO 2 STOP BITS
459E 1Ø99       SWSLLK JMP  SWSLP            AND LOOP FOR NEXT SWS
                *
                * GET 2 CHARACTERS FROM THE I/O INTO R8
                *
45AØ C9ØB FF86  GET2CH MOV  R11,@LEVEL2(R4)  SAVE RETURN
45A4 Ø6AØ 463A         BL   @GETCHR          GET A CHAR IN R6
45A8 C2Ø6              MOV  R6,R8            COPY
45AA Ø6AØ 463A         BL   @GETCHR          GET NEXT CHAR IN R6
45AE Ø6C6              SWPB R6               MOVE TO LSB
45BØ E2Ø6              SOC  R6,R8            ADD TO R8
45B2 1Ø14              JMP  LVL2RT           RETURN TO CALLER
                *
                * SEND THE CRC WORD (IN R9) TO THE I/O
                *
45B4 C9ØB FF86  SNDCRC MOV  R11,@LEVEL2(R4)  SAVE RETURN ADR.
45B8 C189              MOV  R9,R6            GET CRC WORD
45BA Ø6AØ 47E6         BL   @SENDR6          SEND MS BYTE
45BE Ø6C6              SWPB R6               ALIGN LSBYTE
45CØ Ø6AØ 47E6         BL   @SENDR6          SEND THAT TOO
45C4 1ØØB              JMP  LVL2RT           RETURN TO CALLER
                *
                * RECEIVE A CHARACTER, ADD IT TO THE CRC & RETURN
                *
45C6 C9ØB FF86  CRXCRC MOV  R11,@LEVEL2(R4)  SAVE RETURN
45CA Ø6AØ 463A         BL   @GETCHR          GET A CHAR. FROM I/O
45CE 1ØØ4              JMP  A45D8            DO CRC, AND RETURN
                *
                * TRANSMIT A CHARACTER, ADD IT TO THE CRC & RETURN
                *
45DØ C9ØB FF86  CTXCRC MOV  R11,@LEVEL2(R4)  SAVE RETURN
45D4 Ø6AØ 47E6         BL   @SENDR6          SEND THE CHARACTER
45D8 Ø6AØ 47CØ  A45D8  BL   @CRC             DO THE CRC ON IT
45DC C2E4 FF86  LVL2RT MOV  @LEVEL2(R4),R11  RETURN TO CALLER
45EØ Ø45B              RT
                *
                * FIND AN '=' SIGN, GET THE NUMBER FOLLOWING IT, CONVERT
                *  THIS NUMBER TO HEX, THEN SET UP THE BAUD RATE DATA
                *  FROM THAT NUMBER.
                *
45E2 C9ØB FF88  A45E2  MOV  R11,@LEVEL3(R4)  SAVE RETURN
45E6 Ø6AØ 4798         BL   @FINDCH          FIND AN '=' SIGN
45EA 3ØØØ              DATA '='*256
45EC 1391              JEQ  ERR2LK           NOT FOUND, ERROR IT
45EE Ø6AØ 4754         BL   @ASCHEX          CONVERT NUMBER TO HEX
45F2 1ØØ2              JMP  A45F8            GO SET UP BAUD RATE DATA
                *
```

```
                   *    THIS ROUTINE SORTS OUT WHAT HAS TO BE PROGRAMMED
                   *   INTO THE 99Ø2 RECEIVE AND TRANSMIT BAUD RATE
                   *   REGISTERS IN ORDER TO ACHIEVE THE REQUESTED BAUD
                   *   RATE.  THE DATA TO BE PROGRAMMED IS A COMBINATION
                   *   OF CONSOLE (99Ø2) CLOCK RATE AND THE REQUESTED
                   *   BAUD RATE.  WHILST THERE IS AN ALGORITHM FOR
                   *   COMPUTING ANY (YES ANY!) BAUD RATE DATA VALUE
                   *   FOR BOTH TRANSMIT AND RECEIVE SEPARATELY, IT
                   *   HAS BEEN CHOSEN TO DO THE OPERATION VIA TABLES.
                   *   AS A RESULT OF THIS, ONLY 7 BAUD RATES ARE USED
                   *   AND BOTH TRANSMIT AND RECEIVE OPERATE AT THE SAME
                   *   ONE.
                   *
                   *   THE ROUTINE SEARCHES THE 'BAUDTB' FOR THE INPUT RATE,
                   *   AND STORES THE OFFSET IN R2. IT THEN SEARCHES THE
                   *   'CLKTBL' FOR THE CONSOLE FREQUENCY, AND HOLDS THE
                   *   POINTER IN R1.(THERE ARE ONLY TWO OPTIONS!).  HAVING
                   *   DONE THIS, IT ADDS THE VALUE POINTED TO BY R1 TO
                   *   THE OFFSET IN R2 AND USES THIS (NOW A POINTER) TO
                   *   EXTRACT DATA FROM A TABLE 'REGTB1' OR 'REGTB2'
                   *
                   *
 45F4 C9ØB FF88    DOBAUD MOV  R11,@LEVEL3(R4)    SAVE RETURN
 45F9              BYTEØ1 EQU  $+1
 45F8 Ø2Ø1 4ØA6    A45F8  LI   R1,BAUDS           POINT TO BAUD TABLE
 45FC Ø4C2                CLR  R2                 CLEAR THE COUNTER
 45FE C2F1         A45FE  MOV  *R1+,R11           GET BAUD RATE FROM TABLE
 46ØØ 1387                JEQ  ERR2LK             IF END OF TABLE, ERROR IT
 46Ø2 82C5                C    R5,R11             EQUAL TO INPUT BAUD RATE?
 46Ø4 13Ø2                JEQ  A46ØA              YES, FIX UP BAUD RATE
 46Ø6 Ø5C2                INCT R2                 UPDATE COUNTER
 46Ø8 1ØFA                JMP  A45FE              AND LOOP BACK
                   *
 46ØB              BYTEEØ EQU  $+1                WHAT A BAD EQUATE!!
                   *
                   * THE INPUT BAUD RATE HAS BEEN FOUND IN THE TABLE
                   *
 46ØA D2EØ ØØØC    A46ØA  MOVB @>ØØØC,R11         GET THE CONSOLE CLOCK FREQ.
 46ØE Ø98B                SRL  R11,8              TO LSB
 461Ø Ø2Ø1 4Ø9C           LI   R1,CLKTBL          POINT TO CLOCK RATE TABLE
 4614 C171         A4614  MOV  *R1+,R5            GET THE RATE
 4616 1327                JEQ  ERR6LK             IF END OF TABLE, ERROR IT
 4618 82C5                C    R5,R11             SAME AS TABLE ENTRY?
 461A 13Ø2                JEQ  A462Ø              YES, OK, JUMP
 461C Ø5C1                INCT R1                 UPDATE POINTER
 461E 1ØFA                JMP  A4614              AND LOOP BACK
                   *
                   * CLOCK FREQUENCY OF CONSOLE WAS FOUND IN TABLE
                   *
 462Ø AØ91         A462Ø  A    *R1,R2             ADD POINTER TO OFFSET
 4622 CØ52                MOV  *R2,R1             GET REGISTER DATA FROM TABLE
 4624 15Ø5                JGT  A463Ø              IF MSB NOT SET, JUMP
 4626 F66Ø 4Ø72           SOCB @BYTEØ8,*R9        SET 99Ø2 CLK TO DIVIDE BY 4
 462A Ø241 7FFF           ANDI R1,>7FFF           REMOVE THE MSBIT
 462E 1ØØ2                JMP  A4634              SAVE THE BAUD RATE DATA
 463Ø 566Ø 4Ø72    A463Ø  SZCB @BYTEØ8,*R9        SET 99Ø2 CLK TO DIVIDE BY 3
 4634 C9Ø1 FFFE    A4634  MOV  R1,@PAD+>DE(R4)    SAVE THE BAUD RATE DATA
 4638 1Ø23                JMP  LVL3RT             AND RETURN TO CALLER
```

```
                *
                **** CALLED BY BL ****
                *
463A C9ØB FF88  GETCHR MOV   R11,@LEVEL3(R4)   SAVE THE RETURN ADR.
463E Ø6AØ 487Ø  A463E  BL    @CHARDY           IS THERE A CHAR. READY?
4642 13Ø3              JEQ   A464A             YES, JUMP
4644 Ø6AØ 488Ø         BL    @TSTCLR           CHECK FOR CLEAR KEY
4648 1ØFA              JMP   A463E             AND LOOP BACK
                *
464A CØC3       A464A  MOV   R3,R3             IN PARALLEL MODE?
464C 16ØE              JNE   A466A             YES, JUMP
464E Ø4C6              CLR   R6                FOR BYTE OPERATION
465Ø 36Ø6              STCR  R6,8              GET THE RECEIVED CHAR.
4652 1E12              SBZ   18                RESET THE BUFFER LOAD SIGNAL
4654 1FØB              TB    11                OVER-RUN ERROR?
4656 13Ø7              JEQ   ERR6LK            YES, JUMP
4658 1FØC              TB    12                FRAMING ERROR?
465A 13Ø5              JEQ   ERR6LK            YES, JUMP
465C D2E4 FF78         MOVB  @CHFLAG(R4),R11   ARE WE CHECKING PARITY?
466Ø 13ØF              JEQ   LVL3RT            NO, JUMP
4662 1FØA              TB    1Ø                PARITY ERRORS?
4664 16ØD              JNE   LVL3RT            NO, JUMP
4666 Ø46Ø 445C  ERR6LK B     @ERROR6           ELSE ERROR IT
                *
466A 1DØ1       A466A  SBO   1                 DISABLE THE PIO O/P DEVICE
466C 1EØ2              SBZ   2                 SET STROBE (NOW 'BUSY') TO Ø
466E 1FØ2       A466E  TB    2                 WAIT FOR A STROBE (PIN 1Ø)
467Ø 16Ø3              JNE   A4678             JUMP WHEN FOUND ('Ø')
4672 Ø6AØ 488Ø         BL    @TSTCLR           ELSE TEST FOR THE CLEAR KEY
4676 1ØFB              JMP   A466E             AND LOOP BACK
4678 Ø4C6       A4678  CLR   R6                READY FOR BYTE MOVE
467A D1AØ 5ØØØ         MOVB  @>5ØØØ,R6         READ THE CHARACTER FROM INPUT
467E 1DØ2              SBO   2                 ACKNOWLEDGE THE CHAR.
468Ø C2E4 FF88  LVL3RT MOV   @LEVEL3(R4),R11   RETRIEVE THE RETURN ADR.
4684 Ø45B              RT                      RETURN TO CALLER
                *
                * THIS ROUTINE SORTS OUT THE BLOCK LENGTHS TO BE TX'D
                * OR RX'D. ON ENTRY, R7 CONTAINS THE FILE LENGTH, FROM
                * WHICH (IF POSSIBLE) 256 IS SUBTRACTED, AND THAT NO.
                * IS SAVED IN PCOUNT (PROGRAM BYTE COUNT). THE LENGTH
                * OF THE CURRENT BLOCK TO BE DEALT WITH IS HELD IN
                * 'BCOUNT' (BLOCK BYTE COUNT).
                *
4686 C9ØB FF88  SETCNT MOV   R11,@LEVEL3(R4)   SAVE RETURN
468A D1C7              MOVB  R7,R7             LESS THAN 256 BYTES TO GO?
468C 13Ø9              JEQ   A46AØ             YES, JUMP
468E Ø6AØ 4684         BL    @WR7DEC           WRITE BYTE COUNT TO SCREEN
4692 Ø227 FFØØ         AI    R7,-256           SUBTRACT NEXT BLOCK LENGTH
4696 C9Ø7 FF7E         MOV   R7,@PCOUNT(R4)    SAVE AS 'NEW COUNT'
469A Ø2Ø7 Ø1ØØ         LI    R7,256            SET UP NEXT BLOCK LENGTH
469E 1ØØ6              JMP   A46AC             SAVE IT AND EXIT
                *
                * THERE ARE LESS THAN 256 BYTES TO DEAL WITH
                *
46AØ C1C7       A46AØ  MOV   R7,R7             BYTES LEFT = Ø?
46A2 16Ø2              JNE   A46A8             NO, JUMP
46A4 Ø46Ø 4464         B     @CLOSEF           ELSE CLOSE AND EXIT
                *
```

```
46A8 Ø4E4 FF7E   A46A8  CLR   @PCOUNT(R4)        SET FILE COUNT TO ZERO
46AC C9Ø7 FF8Ø   A46AC  MOV   R7,@BCOUNT(R4)     SAVE REMAINING COUNT
46BØ 1ØE7               JMP   LVL3RT             AND RETURN TO CALLER
                 *
                 *  SET R7 MSB TO 255, THEN WRITE IT TO SCREEN IN DECIMAL
                 *
46B2 Ø7Ø7        A46B2  SETO R7
                 *
                 * THIS ROUTINE STARTS AT THE TOP OF THE SCREEN AND
                 * WRITES 14 SPACES, THEN WRITES THE MSB OF R7 TO THE
                 * SCREEN IN DECIMAL, THEN WRITES 14 MORE SPACES.
                 *
46B4 C9ØB FF8A   WR7DEC MOV   R11,@LEVEL4(R4)    SAVE RETURN
46B8 Ø4C1               CLR   R1                 POINT TO START OF SCREEN
46BA Ø6AØ 484E          BL    @SETADR            SET UP VDP ADR IN WRITE MODE
46BE 4ØØØ               DATA WRITE
46CØ Ø6AØ 485A          BL    @SPAC14            SEND 14 SPACES TO SCREEN
46C4 CØ87               MOV   R7,R2              GET FIRST 2 CHARS RXD
46C6 Ø982               SRL   R2,8               MOVE 1ST CHAR TO LSB
46C8 Ø2Ø6 ØØ64          LI    R6,1ØØ             SET DIVISOR TO 1ØØ
46CC Ø4C1        A46CC  CLR   R1                 CLEAR MOST SIGNIFICANT WORD
46CE 3C46               DIV   R6,R1              DIVIDE BY DIVISOR
46DØ Ø221 ØØ3Ø          AI    R1,>ØØ3Ø           ADD ASCII TO RESULT
46D4 ØA81               SLA   R1,8               MOVE TO MSB
46D6 BØ64 FF72          AB    @PAD+>52(R4),R1    ADD THE SCREEN OFFSET
46DA DBC1 FFFE          MOVB R1,@-2(R15)         WRITE TO SCREEN
46DE Ø4C5               CLR   R5                 CLEAR MS WORD
46EØ 3D6Ø 4796          DIV   @TEN,R5            DIVIDE THE DIVISOR BY 1Ø
46E4 C185               MOV   R5,R6              MOVE QUOTIENT TO DIVISOR
46E6 16F2               JNE   A46CC              IF NOT ZERO, LOOP
46E8 Ø6AØ 485A          BL    @SPAC14            14 MORE SPACES TO SCREEN
46EC 1Ø1E               JMP   LVL4RT             RETURN TO CALLER
                 *
                 *  THIS ROUTINE DEALS WITH THE END OF THE LINE, AND ADDS
                 *   CARRIAGE RETURN, LINE FEED AND NULLS AS REQUESTED
                 *   BY THE SOFTWARE SWITCH OPTIONS SET UP BY THE USER
                 *
46EE C9ØB FF8A   LINEND MOV   R11,@LEVEL4(R4)    SAVE RETURN ADR.
46F2 D2E4 FF79          MOVB @CRFLAG(R4),R11     CR FLAG ON?
46F6 1619               JNE   LVL4RT             YES, EXIT (SUPPRESS)
46F8 Ø6AØ 47E4          BL    @TXDATA            SEND A CARRIAGE RETURN
46FC ØDØØ               DATA >DØØ
46FE 1ØØ2               JMP   A47Ø4              AND GO CHECK THE 'NULL' FLAG
                 **** CALLED BY BL ****
47ØØ C9ØB FF8A   A47ØØ  MOV   R11,@LEVEL4(R4)    SAVE THE RETURN ADDRESS
47Ø4 DØ64 FF7C   A47Ø4  MOVB @NUFLAG(R4),R1      IS THE 'NULLS' FLAG SET?
47Ø8 13Ø7               JEQ   A4718              NO, BYPASS NULL ROUTINE
                 *
                 * SEND 6 NULLS AFTER A CARRIAGE RETURN
                 *
47ØA Ø2Ø5 ØØØ6          LI    R5,6               SET COUNTER TO 6
47ØE Ø6AØ 47E4   NULOOP BL    @TXDATA
4712 ØØØØ               DATA Ø                   SEND A NULL (>ØØ)
4714 Ø6Ø5               DEC   R5                 UPDATE LOOP COUNT
4716 16FB               JNE   NULOOP             LOOP TILL ZERO
                 *
                 *
                 *
```

```
4718 DØ64 FF79  A4718  MOVB @CRFLAG(R4),R1     IS THE 'CR' FLAG SET
471C 16Ø6              JNE  LVL4RT             YES, DON'T ADD LINE FEED
471E DØ64 FF7A         MOVB @LFFLAG(R4),R1     IS THE 'LF' FLAG SET?
4722 16Ø3              JNE  LVL4RT             YES, DONT ADD LINE FEED
4724 Ø6AØ 47E4         BL   @TXDATA            SEND A LINE FEED
4728 ØAØØ              DATA >AØØ
472A C2E4 FF8A  LVL4RT MOV  @LEVEL4(R4),R11    RETRIEVE THE RETURN ADR.
472E Ø45B              RT                      AND RETURN TO CALLER
                *
                *  CHECK IF THE OPCODE IS A LOAD OR SAVE (SET 'EQU' IF SO)
                *
473Ø DØ64 FF6A  CHKLSV MOVB @FAC(R4),R1        GET THE OPCODE
4734 Ø981              SRL  R1,8               TO LSB
4736 Ø221 FFFB         AI   R1,-5              SUBTRACT 5
473A 13Ø1              JEQ  A473E              IF LOAD OPCODE, JUMP
473C Ø6Ø1              DEC  R1                 IF SAVE, ZERO REMAINS
473E Ø45B       A473E  RT                      RETURN TO CALLER
                *
                * CHECK IF FILE IS INTERNAL DATA TYPE (SET 'EQU' IF SO)
                *
474Ø DØ64 FF6B  INTRNL MOVB @FAC+1(R4),R1
4744 2Ø6Ø 4Ø72         COC  @BYTEØ8,R1
4748 Ø45B              RT
                *
                * CHECK IF FILE IS FIXED RECORD LENGTH (SET 'EQU' IF SO)
                *
474A DØ64 FF6B  TSTFIX MOVB @FAC+1(R4),R1      GET THE FLAG BYTE
474E Ø241 1ØØØ         ANDI R1,>1ØØØ           LEAVE ONLY THE VARIABLE BIT
4752 Ø45B              RT                      RETURN TO CALLER
                *
                * CONVERT ASCII NUMBERS IN PAB TO A HEX WORD
                *
4754 C9ØB FF8A  ASCHEX MOV  R11,@LEVEL4(R4)    SAVE RETURN
4758 Ø4C1              CLR  R1                 CLEAR THE TOTAL
475A Ø4CB              CLR  R11                CLEAR THE BYTE COUNT
475C 1ØØ3              JMP  A4764
                *
475E D1AF FBFE  A475E  MOVB @>FBFE(R15),R6     READ NEXT CHAR FROM PAB
4762 Ø6ØØ              DEC  RØ
4764 C1C6       A4764  MOV  R6,R7              GET CHARACTER
4766 Ø987              SRL  R7,8               MOVE TO LSB
4768 Ø227 FFDØ         AI   R7,->3Ø            REMOVE THE ASCII
476C 11ØC              JLT  A4786              IF NEGATIVE, POSSIBLE ERROR
476E Ø287 ØØØ9         CI   R7,9               IS IT A NUMBER?
4772 18Ø9              JH   A4786              NO, POSSIBLE ERROR
4774 Ø58B              INC  R11                COUNT THE CHAR
4776 386Ø 4796         MPY  @TEN,R1            MULTIPLY PRESENT TOTAL BY 1Ø
477A CØ41              MOV  R1,R1              ANSWER MORE THAN 16 BITS?
477C 16Ø6              JNE  A478A              YES, ERROR IT
477E AØ87              A    R7,R2              ADD IN NEW NUMBER
478Ø CØ42              MOV  R2,R1              SET UP MULTIPLICAND
4782 CØØØ              MOV  RØ,RØ              HAVE WE FINISHED?
4784 16EC              JNE  A475E              NO, LOOP
4786 C2CB       A4786  MOV  R11,R11            HAVE WE FOUND ANY NUMBERS?
4788 16Ø2              JNE  A478E              YES, JUMP
478A Ø46Ø 444A  A478A  B    @ERROR2            ELSE ERROR IT
478E C141       A478E  MOV  R1,R5              HEX CONVERSION OF INPUT TO R5
479Ø C2E4 FF8A         MOV  @LEVEL4(R4),R11    RETURN ON LEVEL 4
```

```
4794 Ø45B                 RT
                  *
4796 ØØØA         TEN     DATA 1Ø
                  *
                  *  SEARCH FOR THE CHARACTER IN THE MS BYTE OF THE DATA
                  *  STATEMENT FOLLOWING THE CALL.  START THE SEARCH WITH
                  *  THE LAST BYTE READ (IT'S IN R6) THEN CONTINUE FROM
                  *  THEN CURRENT READ ADR. IN THE VDP. THE MAXIMUM NO.
                  *  OF CHARACTERS LEFT TO BE READ FROM THE VDP IS PASSED
                  *  IN RØ.  IF THE CHARACTER IS FOUND, THEN THE 'EQU'
                  *  BIT IN THE 99ØØ STATUS REGISTER IS CLEARED.  IE
                  *  A 'JNE' INSTRUCTION WOULD RESULT IN THE JUMP BEING
                  *  TAKEN IF THE CHARACTER WAS FOUND.
                  *     IF THE CHARACTER IS FOUND, THE FOLLOWING CHAR IS
                  *  READ AND PASSED TO THE CALLER IN R6.
                  *
4798 C17B         FINDCH MOV  *R11+,R5           GET DATA STATEMENT
479A 9185                CB   R5,R6              SEARCH BYTE = LAST BYTE?
479C 13Ø7                JEQ  A47AC              YES, JUMP
479E D1AF FBFE    A479E  MOVB @>FBFE(R15),R6     READ A BYTE FROM THE VDP
47A2 Ø6ØØ                DEC  RØ                 COUNT IT
47A4 9185                CB   R5,R6              SAME AS SEARCH BYTE?
47A6 13Ø2                JEQ  A47AC              YES, JUMP
47A8 CØØØ                MOV  RØ,RØ              ANY CHARS. LEFT?
47AA 16F9                JNE  A479E              YES, LOOP
47AC CØØØ         A47AC  MOV  RØ,RØ              ANY CHARS LEFT?
47AE 13Ø7                JEQ  A47BE              NO, EXIT ('EQU' SET)
47BØ Ø4C6                CLR  R6                 FOR BYTE MOV
47B2 D1AF FBFE           MOVB @>FBFE(R15),R6     READ NEXT BYTE
47B6 Ø6ØØ                DEC  RØ                 COUNT IT
47B8 Ø286 2ØØØ           CI   R6,' '*256         SPACE READ?
47BC 13F7                JEQ  A47AC              YES, LOOP
47BE Ø45B         A47BE  RT
                  *
                  *  COMPUTE A CYCLIC REDUNDANCY CHECK WORD FROM CHARACTER
                  *         IN MSB OF R6. RESULT IS STORED IN R9.
                  *
                  * THE POLYNOMIAL USED FOR THE CRC IS :
                  *
                  *   16   12   5
                  *  X  + X  + X + 1
                  *
47CØ CØ46         CRC    MOV  R6,R1
47C2 Ø241 FFØØ           ANDI R1,>FFØØ
47C6 2A41                XOR  R1,R9
47C8 CØ49                MOV  R9,R1
47CA Ø941                SRL  R1,4
47CC 2849                XOR  R9,R1
47CE Ø241 FFØØ           ANDI R1,>FFØØ
47D2 Ø941                SRL  R1,4
47D4 2A41                XOR  R1,R9
47D6 Ø871                SRC  R1,7
47D8 2A41                XOR  R1,R9
47DA Ø6C9                SWPB R9
47DC Ø45B                RT
                  *
                  * READ A CHARACTER FROM VDP BUFFER, THEN SEND TO I/O
                  *
```

```
47DE D1AF FBFE  TXVCHR MOVB @>FBFE(R15),R6
47E2 1ØØ1              JMP  SENDR6
                *
                *    ENTRY TO HERE (VIA BL) WILL RESULT IN THE CHARACTER
                *  CODE IN THE MSB OF THE DATA STATEMENT BEING TRANSMITTED.
                *
47E4 C1BB       TXDATA  MOV  *R11+,R6
                *
                *  TRANSMIT THE MSB OF R6 DOWN THE I/O
                *
47E6 C9ØB FF8C  SENDR6 MOV  R11,@LEVEL5(R4)
47EA CØC3       A47EA  MOV  R3,R3               IN PARALLEL MODE?
47EC 16ØD              JNE  A48Ø8               YES, JUMP
47EE 1D1Ø              SBO  16                  ENABLE 'RTS' TO O/P
47FØ 1F1B              TB   27                  DATA SET READY?
47F2 16Ø2              JNE  A47F8               NO, JUMP
47F4 1F16              TB   22                  TRANSMIT BUFFER REG EMPTY?
47F6 13Ø3              JEQ  A47FE               YES, GO SEND A CHARACTER
47F8 Ø6AØ 488Ø  A47F8  BL   @TSTCLR             ELSE TEST FOR CLEAR KEY
47FC 1ØF6              JMP  A47EA               AND LOOP BACK
                *
47FE 32Ø6       A47FE  LDCR R6,8                SEND THE CHARACTER
48ØØ 1E1Ø              SBZ  16                  'RTS' TO '1' WHEN TX EMPTY
48Ø2 C2E4 FF8C  A48Ø2  MOV  @LEVELS(R4),R11     RETRIEVE THE RETURN ADR.
48Ø6 Ø45B              RT                       RETURN TO CALLER
                *
48Ø8 1EØ1       A48Ø8  SBZ  1                   ENABLE THE PIO O/P DEVICE
48ØA 1FØ2              TB   2                   'BUSY' LOW?
48ØC 13F5              JEQ  A47F8               NO, JUMP
48ØE D8Ø6 5ØØØ         MOVB R6,@>5ØØØ           YES, SEND THE CHAR TO PIO
4812 1EØ2              SBZ  2                   SET THE STROBE LOW
4814 1FØ2       A4814  TB   2                   'BUSY' HIGH?
4816 13Ø3              JEQ  A481E               YES, JUMP
4818 Ø6AØ 488Ø         BL   @TSTCLR              NO
481C 1ØFB              JMP  A4814
481E 1DØ2       A481E  SBO  2                   SET STROBE HIGH
482Ø 1ØFØ              JMP  A48Ø2               AND EXIT TO CALLER
                *
                * SET UP THE 99Ø2 CONTROL REGISTER (IF IN SERIAL MODE)
                *  OR THE PIO FOR O/P IF IN PARALLEL MODE.
                *
4822 CØC3       SETCTL MOV  R3,R3               IN SERIAL MODE?
4824 13Ø3              JEQ  A482C               YES, JUMP
4826 1DØ2              SBO  2                   SET STROBE TO '1'
4828 1EØ1              SBZ  1                   ENABLE THE PIO O/P DEVICE
482A Ø45B              RT                       RETURN TO CALLER
                *
                *  THE DATA FOR THE 99Ø2 CONTROL REGISTER HAS BEEN
                *  COMPUTED FROM THE SOFTWARE SWITCH OPTIONS AND IS
                *  STORED IN LOCATION PAD+>DA. LIKEWISE, DATA FOR
                *  THE TX AND RX BAUD RATE REGISTERS HAS BEEN COMPUTED
                *  AND IS STORED AT PAD+>DE
                *
482C 1D1F       A482C  SBO  31                  RESET THE 99Ø2
482E 3224 FFFA         LDCR @PAD+>DA(R4),8      LOAD THE CONTROL REGISTER
4832 1EØD              SBZ  13                  DON'T LOAD THE INTERVAL REG
4834 3324 FFFE         LDCR @PAD+>DE(R4),12     LOAD THE BAUD REGISTERS
```

```
4838 DØ64 FF7D         MOVB @CBFLAG(R4),R1    CIRCULAR BUFFER REQUESTED?
483C 13Ø1              JEQ  A484Ø             NO, EXIT
483E 1D12              SBO  18                ELSE NABLE RX INTERRUPTS
484Ø Ø45B       A484Ø  RT                     RETURN TO CALLER
                *
                *   COMPUTE THE START ADDRESS OF THE PAB IN VDP RAM, AND
                *   PUT THE RESULT IN R1 SO THAT THIS ADR. CAN BE
                *   SET UP IN THE VDP.
                *
4842 CØ64 FF76  SETPAB MOV  @PAD+>56(R4),R1   GET NAME LENGTH POINTER
4846 6Ø64 FF74         S    @PAD+>54(R4),R1   SUBTRACT DSR NAME LENGTH
484A Ø221 FFF6         AI   R1,-1Ø            SUB. 1Ø TO POINT TO PAB START
                *
                *  ENTRY TO HERE (VIA BL) WRITES R1 TO THE VDP
                *  IN THE MODE DETERMINED BY THE DATA STATEMENT
                *
484E AØ7B       SETADR A    *R11+,R1          ADD DATA STATEMENT
                *
                *  ENTRY TO HERE (VIA BL) WRITES R1 TO THE VDP
                *          AS THE VDP READ ADDRESS
                *
4850 D7E4 ØØØ3  SETRDA MOVB @3(R4),*R15       WRITE R1 LSB TO VDP ADDRESS
4854 1ØØØ              NOP                    WAIT FOR VDP
4856 D7C1              MOVB R1,*R15           WRITE R1 MSB TO VDP ADDRESS
4858 Ø45B              RT                     RETURN TO CALLER
                *
                *  WRITE 14 SPACES STARTING AT THE CURRENT VDP WRITE ADR.
                *
485A Ø2Ø1 2Ø2Ø  SPAC14 LI   R1,'  '           SET UP SPACES
485E BØ64 FF72         AB   @PAD+>52(R4),R1   ADD THE SCREEN OFFSET BYTE
4862 Ø2Ø2 ØØØE         LI   R2,14             14 SPACES REQUIRED
4866 DBC1 FFFE  A4866  MOVB R1,@-2(R15)       WRITE A SPACE
486A Ø6Ø2              DEC  R2                COUNT IT
486C 16FC              JNE  A4866             LOOP TILL ZERO
486E Ø45B              RT                     RETURN TO CALLER
                *
                *  TEST THE RS232 AND THE PIO TO SEE IF A CHARACTER HAS
                *  BEEN RECIEVED.
                *    ON RETURN TO CALLER, SET 'EQU' STATUS IF A CHAR. HAS
                *  BEEN RECEIVED.
                *
4870 CØC3       CHARDY MOV  R3,R3             IN PARALLEL MODE?
4872 16Ø4              JNE  A487C             YES, JUMP
                *
                * ENTRY TO HERE CHECKS ONLY TO SEE IF A CHARACTER HAS
                * BEEN RECEIVED BY THE RS232 PORTS.
                *
4874 1F1B       SRXRDY TB   27                DATA SET READY?
4876 16Ø1              JNE  A487A             NO, EXIT
4878 1F15              TB   21                RX BUFFER REGISTER LOADED?
487A Ø45B       A487A  RT                     -----EXIT (TEST ON RETURN)
487C 1FØ2       A487C  TB   2                 BUSY HIGH?
487E Ø45B              RT                     ---- EXIT (TEST ON RETURN)
                *
                * TEST TO SEE IF THE 'CLEAR' KEY IS PRESSED. IF IT IS THEN
                *  ABORT THE I/O OPERATION AND EXIT THE DSR VIA ERROR6.
                *  IF THE KEY IS NOT PRESSED, THEN RETURN TO THE CALLING
                *  ROUTINE WITH NO CHANGE IN CONDITIONS.
```

```
                         *
  488Ø C Ø4C             TSTCLR MOV  R12,R1              SAVE THE I/O CRU BASE
  4882 Ø2ØC ØØ24                LI   R12,>ØØ24           POINT TO KEYBOARD
  4886 3ØEØ 4Ø73                LDCR @BYTEØØ,3           POINT TO KBD, COLUMN 1
  488A 1FF5                     TB   -11                 FUNCTION KEY DOWN?
  488C 13Ø4                     JEQ  A4896               NO, EXIT
  488E 3ØEØ 4Ø74                LDCR @HXØ3Ø3,3           POINT TO KBD, COLUMN 3
  4892 1FF5                     TB   -11                 '4' (CLEAR) KEY DOWN?
  4894 16Ø2                     JNE  A489A               YES, JUMP
  4896 C3Ø1              A4896  MOV  R1,R12              RESTORE I/O CRU BASE
  4898 Ø45B                     RT                       RETURN TO CALLER
                         * CLEAR KEY WAS PRESSED
  489A C3Ø1              A489A  MOV  R1,R12              RESTORE I/O CRU BASE
  489C Ø46Ø 445C                B    @ERROR6
                         *
  48AØ ABCD                     DATA >ABCD
                         *
                         *   WAIT.
                         *
  48A2 ØB8Ø              WAIT   SRC  RØ,8
  48A4 ØB8Ø                     SRC  RØ,8
  48A6 ØB8Ø                     SRC  RØ,8
  48A8 ØB8Ø                     SRC  RØ,8
  48AA ØB8Ø                     SRC  RØ,8
  48AC ØB8Ø                     SRC  RØ,8
  48AE Ø45B                     RT
```

```
41ØE   A41ØE   4124   A4124   4136   A4136   413E   A413E
4156   A4156   4168   A4168   4184   A4184   418A   A418A   418E   A418E
419Ø   A419Ø   419E   A419E   41A8   A41A8   41C2   A41C2   41E4   A41E4
41FØ   A41FØ   4228   A4228   4258   A4258   425A   A425A   425E   A425E
427C   A427C   42A2   A42A2   42A8   A42A8   42AC   A42AC   42CC   A42CC
42D8   A42D8   42DC   A42DC   42EE   A42EE   42F8   A42F8   43ØØ   A43ØØ
4318   A4318   431C   A431C   4324   A4324   4334   A4334   433C   A433C
4346   A4346   434A   A434A   435E   A435E   4382   A4382   438C   A438C
439Ø   A439Ø   439A   A439A   43B2   A43B2   43C2   A43C2   43D8   A43D8
43E2   A43E2   43EC   A43EC   44ØC   A44ØC   441Ø   A441Ø   4418   A4418
443A   A443A   444Ø   A444Ø   446Ø   A446Ø   448Ø   A448Ø   44A4   A44A4
44AC   A44AC   44EA   A44EA   45ØA   A45ØA   45ØE   A45ØE   456A   A456A
459Ø   A459Ø   4594   A4594   45D8   A45D8   45E2   A45E2   45F8   A45F8
45FE   A45FE   46ØA   A46ØA   4614   A4614   462Ø   A462Ø   463Ø   A463Ø
4634   A4634   463E   A463E   464A   A464A   466A   A466A   466E   A466E
4678   A4678   46AØ   A46AØ   46A8   A46A8   46AC   A46AC   46B2   A46B2
46CC   A46CC   47ØØ   A47ØØ   47Ø4   A47Ø4   4718   A4718   473E   A473E
475E   A475E   4764   A4764   4786   A4786   478A   A478A   478E   A478E
479E   A479E   47AC   A47AC   47BE   A47BE   47EA   A47EA   47F8   A47F8
47FE   A47FE   48Ø2   A48Ø2   48Ø8   A48Ø8   4814   A4814   481E   A481E
482C   A482C   484Ø   A484Ø   4866   A4866   487A   A487A   487C   A487C
4896   A4896   489A   A489A   4754   ASCHEX  4536   BASWS   4ØA6   BAUDS
FF8Ø   BCOUNT  42Ø2   BTABLE  4Ø73   BYTEØØ  45F9   BYTEØ1  41A7   BYTEØ6
4Ø72   BYTEØ8  422C   BYTE2Ø  4ØA1   BYTE3Ø  4ØØ4   BYTE4Ø  4ØB3   BYTE8Ø
42DC   BYTECØ  46Ø8   BYTEEØ  4132   BYTEFF  FF7D   CBFLAG  487Ø   CHARDY
FF7B   CHFLAG  473Ø   CHKLSV  452A   CHSWS   4Ø9C   CLKTBL  4464   CLOSCF
47CØ   CRC     FF79   CRFLAG  4518   CRSWS   45C6   CRXCRC  45ØØ   CTXCRC
```

```
457Ø    DASWS    45F4    DOBAUD   449Ø    DOSWS    402Ø    DSRL1    4Ø2C    DSRL2
4Ø38    DSRL3    4Ø4Ø    DSRL4    4Ø4A    DSRL5    4Ø54    DSRL6    4Ø6Ø    DSRL7
4Ø16    DSRLNK   FF78    ECFLAG   4512    ECSWS    451Ø    ERR2LK   4666    ERR6LK
444A    ERROR2   445Ø    ERROR3   4456    ERROR4   445C    ERROR6   FF6A    FAC
4798    FINDCH   45AØ    GET2CH   463A    GETCHR   43CA    HXØ1ØØ   4Ø74    HXØ3Ø3
4Ø6C    INTLNK   474Ø    INTRNL   4ØD2    INTRPT   FF84    LEVEL1   FF86    LEVEL2
FF88    LEVEL3   FF8A    LEVEL4   FF8C    LEVEL5   FF7A    LFFLAG   451E    LFSWS
46EE    LINEND   4338    LOADF    45DC    LVL2RT   468Ø    LVL3RT   472A    LVL4RT
FF7C    NUFLAG   47ØE    NULOOP   4524    NUSWS    421Ø    OPENF    FF2Ø    PAD
454Ø    PASWS    FF7E    PCOUNT   415E    PIO      4164    PIO2     4Ø1Ø    PWRLNK
4ØF4    PWRUP    ØØØØ    RØ       ØØØ1    R1       ØØØA    R1Ø      ØØØB    R11
ØØØC    R12      ØØØD  * R13      ØØØE  * R14      ØØØF    R15      ØØØ2    R2
ØØØ3    R3       ØØØ4    R4       ØØØ5    R5       ØØØ6    R6       ØØØ7    R7
ØØØ8    R8       ØØØ9    R9       ØØØØ    READ     4236    READF    4Ø86    REGTB1
4ØC4    REGTB2   416E    RS232    4174    RS2322   418Ø    RS2323   417A    RS2324
4ØF2    RSTSIO   43D2    SAVEF    47E6    SENDR6   484E    SETADR   4686    SETCNT
4822    SETCTL   452E    SETFGS   4842    SETPAB   485Ø    SETRDA   4584    SNDCRC
485A    SPAC14   4874    SRXRDY   44F6    SWSEND   459E    SWSLLK   44D2    SWSLP
4Ø76    SWSTB1   4Ø8A    SWSTB2   4796    TEN      488Ø    TSTCLR   474A    TSTFIX
4596    TWSWS    47E4    TXDATA   47ØE    TXVCHR   48A2    WAIT     4684    WR7DEC
4ØØØ    WRITE    42FA    WRITEF   4ØØ8  * X4ØØ8    4ØØC  * X4ØØC
```

TIM MacEACHERN

```
       TITL 'TI RS232 & PIO DSR ROM'
*
* Decoded and commented by:
*    Tim MacEachern
*    PO Box 1105
*    Dartmouth, NS
*    Canada  B2Y 4B8
*
* Note: All rights to this code belong to Texas Instruments.
*       This listing is provided only as tutorial material.
*
       AORG >4ØØØ   Standard peripheral ROM address
*
* Standard header table
*
       BYTE >AA     Header table flag
       BYTE 1       Version number (also used as level)
       DATA Ø       Autorun code address
       DATA INITLS  Initialization list pointer
BYTE4Ø EQU  $-2     Convenient byte value >4Ø for use in code
       DATA Ø       Normal subroutine list pointer
       DATA DEVL1   Device name list
       DATA Ø       DSR subroutine list
       DATA INTRLS  Interrupt service routine list
       DATA Ø       (unused)
*
* Standard list format is:
*      DATA next-pointer  points to next element in list (Ø for done)
*      DATA code-pointer  points to code to do the function
*      BYTE length        length of name (may be Ø if no name is needed)
*      TEXT 'name'        name of device, routine, etc.
*      EVEN               note: all devices below happen not to need this
*


*
* Initialization code list
*
INITLS DATA Ø,INIT        all initialization code starts at INIT
       BYTE Ø,Ø
*
* Device name list
*
DEVL1  DATA DEVL2         list entry for RS232 device
       DATA RS232
       BYTE 5
       TEXT 'RS232'
*
DEVL2  DATA DEVL3
       DATA RS232         treat RS232/1 as same as RS232
       BYTE 7
       TEXT 'RS232/1'
*
DEVL3  DATA DEVL4         RS232/2 device
       DATA RS2322
       BYTE 7
       TEXT 'RS232/2'
*
DEVL4  DATA DEVL5         parallel port
       DATA PIO
       BYTE 3
       TEXT 'PIO'
*
```

```
DEVL5  DATA DEVL6         alternate name for PIO
       DATA PIO
       BYTE 5
       TEXT 'PIO/1'
*
DEVL6  DATA DEVL7         PIO/2 -- defined to allow for future offerings
       DATA PIO2          (i.e. not currently supported)
       BYTE 5
       TEXT 'PIO/2'
*
DEVL7  DATA DEVL8         RS232/3 -- defined to allow for future offerings
       DATA RS2323
       BYTE 7
       TEXT 'RS232/3'
*
DEVL8  DATA Ø             last device
       DATA RS2324        RS232/4 -- defined to allow for future offerings
       BYTE 7
       TEXT 'RS232/4'
*
* Interrupt service routine list
*
INTRLS DATA Ø             no more entries
       DATA INTR          interrupt code
       BYTE Ø,Ø           name not needed
*
* Miscellaneous data bytes
*
BYTEØ8 BYTE >Ø8           Byte used to check bit 3 of words (internal flag)
KFUNCT BYTE Ø             Keyboard row number for FCTN, enter, shift, space,
KROW4  BYTE 3             Keyboard row number for MJU74FRV
BYTEØ3 EQU  $-1           Convenient byte value >Ø3 for use in code
       BYTE 3             Unused byte (value is an artifact of the assembler
*
* Legal parameters for OPEN calls
*
PARMOP TEXT 'EC'          echo on/off
       DATA PARMEC        pointer to code
*
       TEXT 'CR'          carriage return, line feed inhibit
       DATA PARMCR
*
       TEXT 'LF'          line feed inhibit
       DATA PARMLF
*
       TEXT 'NU'          nulls transmitted for timing
       DATA PARMNU
*
       TEXT 'DA'          number of data bits (7 or 8)
       DATA PARMDA
*
* Parameters legal for both OPEN and SAVE/LOAD
*
PARMSV TEXT 'BA'          baud rate
       DATA PARMBA
*
       TEXT 'PA'          parity
       DATA PARMPA
*
       TEXT 'TW'          stop bits
       DATA PARMTW
*
       TEXT 'CH'          check parity
       DATA PARMCH
       DATA Ø             end of parameters
*
* Machine speed table - used to determine 99Ø2 clock rate
* (99/4A clock rate is stored in ROM byte >ØØØC)
*
SPEEDS DATA >28           2.5 Megahertz (used on some old machines)
       DATA SRATES        pointer to slow data rate table
*
       DATA >3Ø           3 Megahertz (standard value for 99/4As)
BYTE3Ø EQU  $-1           Convenient byte value >3Ø for use in code
       DATA FRATES        pointer to fast data rate table
*
       DATA Ø             end of table
*
* Baud rate table
*
BAUDS  DATA 11Ø
       DATA 3ØØ
       DATA 6ØØ
       DATA 12ØØ
       DATA 24ØØ
       DATA 48ØØ
       DATA 96ØØ
BYTE8Ø EQU  $-1           96ØØ in hex is >258Ø. The >8Ø byte is used for tests
*                         in later code (I DIDN'T WRITE THIS!)
*
       DATA Ø             end of table
*
* 99Ø2 countdown rates for each baud rate
* values are found by using corresponding offset from baud rate table
* see 99Ø2 documentation for more details
*
* Sample calculation: 3ØØ baud on a 3 Mhz machine - table value is:
*     1                  = divide clock frequency by 4 before counting
*                          (if Ø, clock frequency is divided by 3)
*      ØØØ Ø             = (unused)
*           1            = divide count rate by 8 before counting
*                          (if Ø, count rate is used without further dividing)
*            ØØ 1ØØ1 11ØØ = count to >9C or 156
*    ---- ---- ---- ----
*       8    4    9    C  = transfer rate table datum
* Frequency used is 3ØØ = (3,ØØØ,ØØØ)/(4x8x156)/2 -- the last '2' is a constant
*
* This table is for slow (2.5 Mhz) machines
*
SRATES DATA >8563         for 11Ø baud
       DATA >8482         for 3ØØ baud
       DATA >82Ø9         for 6ØØ baud
       DATA >Ø158         for 12ØØ baud
       DATA >8Ø82         for 24ØØ baud
       DATA >8Ø41         for 48ØØ baud
       DATA >ØØ28         for 96ØØ baud
*
* This table is for fast (3 Mhz) machines
*
FRATES DATA >85AA         for 11Ø baud
       DATA >849C         for 3ØØ baud
       DATA >8271         for 6ØØ baud
       DATA >Ø1A1         for 12ØØ baud
       DATA >8Ø9C         for 24ØØ baud
       DATA >8Ø4E         for 48ØØ baud
       DATA >8Ø27         for 96ØØ baud
```

```
************************
* RS232/2 - second file *
************************
*
* Executable code
*
****************************************
*
* Interrupt service routine - called whenever a character is received
* after the RS232 port has been opened in interrupt-driven mode
* To get to interrupt driven mode open with I/O command op >8Ø
*
* Workspace used is >83EØ - GPL workspace
* Register contents on entry:
*    R2  = pointer to interrupt list entry
*    R11 = return address
*    R12 = CRU base (>13ØØ for CRU pin base >98Ø)
*    R13 = >98ØØ (GROM read address - also used to set read address)
*    R14 = >Ø1xx (Constant 1 byte plus machine state flags)
*    R15 = >8CØ2 (VDP write address location - also used on VDP reads and writes
*
* Contents of RØ, R12, R13, R14, R15 must be retained on exit
*
INTR   STWP R4              Store workspace pointer (>83EØ) in R4
*                           used so this peripheral could work in a machine with
*                           a oifferent memory map. All scratchpad references are
*                           made relative to the calling workspace pointer.
* General housekeeping
       SBO  7               Turn on the light (device is operating)
       MOV  R11,R5          Save the return address
       MOV  R12,R6          Save the CRU base address
* Test RS232/1 for a pending character
       AI   R12,>4Ø         Offset the CRU base to RS232/1's 99Ø2 CRU space (>9AØ)
       TB   16              Is the receiver buffer full (that is, is there a char)
       JEQ  INTGTC          Yes, get the char & transfer to the VDP buffer
* Test RS232/1 for some other (spurious) interrupt
       TB   31              Test general 99Ø2 interrupt bit
       JEQ  INTRST          Spurious interrupt - reset device
* Test RS232/2 for the interrupt (code is as above)
       AI   R12,>4Ø         Advance to RS232/2 (>9CØ)
       TB   16
       JEQ  INTGTC
       TB   31
       JNE  INTRT           Return from the interrupt - wasn't us
*
INTRST MOV  R6,R12          Restore original CRU base & flow into reset code
*
* Initialize devices
*
INIT   MOV  R12,R6          Save the CRU base
       SBO  7               Turn on the light
* Reset PIO
       SBO  2               Turn off the data strobe
       SBZ  1               Switch the parallel port to write mode
* Reset RS232/1
       AI   R12,>4Ø         Advance to RS232/1's 99Ø2 CRU base (>9AØ)
       SBO  31              Reset the 99Ø2
* Reset RS232/2
       AI   R12,>4Ø         Advance to RS232/2's 99Ø2 CRU base (>9CØ)
       SBO  31              Reset the 99Ø2
       MOV  R6,R12        Restore the CRU base
       SBZ  7             Turn off the light
       RT                 Return to power-up, or from the interrupt handler
*
* Get the character that caused the interrupt
*
INTGTC BL   @QSRRDY       Check DSR and RBRL - is there a char?
       JNE  INTRT         No character - return
* Figure out where to put the character
       MOVB @>FF24(R4),R1 Get the buffer IN pointer from >83Ø4
       AB   @BYTEØ1,R1    Add 1 to the pointer
       CB   R1,@>FF22(R4) Compare the pointer to the buffer size (>83Ø2)
       JLE  INTCIR        Jump if still okay, not past the end of the buffer
       CLR  R1            Else reset to the start of the buffer
* Test for buffer overflow (IN pointer catches up to OUT pointer)
INTCIR CB   R1,@>FF23(R4) Compare IN to the OUT pointer (>83Ø3)
       JEQ  INTOVR        Jump if overflow error has occurred
* Get the character, replace with error char if necessary
       STCR R7,8          Read the character from the 99Ø2
       TB   9             Was there a receive error? (parity, framing, etc.)
       JNE  INTSAV        No, received OK. Skip to storing
* Character was received poorly
       LI   R7,>FFØØ      Replace the bad character with the DEL char, ASCII 127
*                         (parity bit is set on as well)
BYTEFF EQU  $-2           A convenient source for an >FF masking byte
       JMP  INTSAV        Skip to store
* Overflow of buffer
INTOVR LI   R7,>FEØØ      Load ' ' to indicate buffer overflow
       MOVB @>FF24(R4),R1 Restore the previous IN pointer,
*                         i.e. overwrite the last valid character in the buffer
* Save the new character in the VDP buffer
INTSAV MOVB R1,@>FF24(R4) Save the new IN pointer
       SRL  R1,8          Make it a 16-bit integer
       A    @>FF2Ø(R4),R1 Add the buffer start address (from >83ØØ)
       ANDI R1,>3FFF      Make sure it is inside the VDP space
       BL   @SETVDP       Set the VDP to write to that address
       DATA >4ØØØ         Code to indicate setting for write, not read
       MOVB R7,@>FFFE(R15) Write the character to the VDP RAM byte
*
* Return from the interrupt
*
INTRT  SBO  18            clear the 99Ø2's interrupt
       MOV  R6,R12        restore the original CRU base value
       SBZ  7             turn the light off
       B    *R5           return (return address was saved in R5)
*
****************************************
* Device code entry points
*
* Each device is encoded as:
*                          RS232/1  RS232/2  RS232/3  RS232/4  PIO/1  PIO/2
* Reg.      Contents       -------  -------  -------  -------  -----  -----
*  R6   Port set number       1        1        2        2       1      2
*  R3   Serial or parallel    Ø        Ø        Ø        Ø      -1     -1
*  R2   CRU base offset      >4Ø      >8Ø      >4Ø      >8Ø      Ø      Ø
*
* Workspace used is >83EØ - GPL workspace
* Register contents on entry:
*    R1   = 1
*    R2   = pointer to interrupt list entry
*           (also saved in >83D2)
*     R9  = code address (i.e. we got here by B *R9)
*     R11 = return address
*     R12 = CRU base (>13ØØ for CRU pin base >98Ø)
*           (also stored in >83DØ)
*     R13 = >98ØØ (GROM read address - also used to set read address)
*     R14 = >Ø1xx (Constant 1 byte plus machine state flags)
*     R15 = >8CØ2 (VDP write address location - also used on VDP reads and writes
*
* Also, the device name length is in word >8354 (e.g. 5 for RS232.PA=N)
* and the VDP pab pointer is in >8456 (points to the period after
* the device name, or just past the device name if there is no period)
*
* Contents of R12, R13, R14, R15 must be retained on exit
*
PIO    LI   R6,1
       JMP  PIOSET
*
PIO2   LI   R6,2
PIOSET SETO R3
       CLR  R2
       JMP  DEVGEN
*
RS232  LI   R6,1
       JMP  RSDEV1
*
RS2322 LI   R6,1
       JMP  RSDEV2
*
RS2324 LI   R6,2
       JMP  RSDEV2
*
RS2323 LI   R6,2
RSDEV1 LI   R2,>4Ø
       JMP  RSFIN
*
RSDEV2 LI   R2,>8Ø
RSFIN  CLR  R3
*
* General device handling code
*
DEVGEN STWP R4              Store the scratchpad WS address. This value is used
*                           to allow all references to be relative to the
*                           existing hardware. For 99/4As the value in R4 is >83EØ
       MOV  R11,@>FF84(R4) Save return address in >8364
       C    R1,R6           Check that level 2 devices are not used (the check is
*                           made against the serial number from >4ØØ1, which was
*                           read in DSRLNK and put into R1.
       JEQ  NOTDV2          OK, device is actually there. Skip to good code
       B    @ERRDSR         Leave DSRLNK without recognizing the device
*
* Clear >8358 to >8363 to store device parameters
* Bytes used are:
*   >8358 - echo option                 Ø = echo on
*   >8359 - CRLF option                 Ø = CRLF on
*   >835A - LF option                   Ø = LF on
*   >835B - check parity option         Ø = parity check off
*   >835C - transmit nulls option       Ø = nulls off
*   >835D - interrupt driven RS232 port Ø = not interrupt driven
*
NOTDV2 MOV  R4,R6           Workspace pointer
       AI   R6,>FF78        Pointer to >8358
       LI   R5,6            Number of words to clear (12 bytes)
BYTEØ6 EQU  $-1             The byte >Ø6 used for testing elsewhere
```

```
DEVLPA CLR  *R6+           Clear a word
       DEC  R5             Finished yet?
       JNE  DEVLPA         Repeat if needed
*
       SBO  7              Turn on the light
       A    R2,R12         Compute the CRU base address - >98Ø for PIO,
*                          >9AØ for RS232/1, >9CØ for RS232/2
* Move the PAB to FAC area, >834A through >8353
       BL   @VDPPAB        Set VDP to read starting at byte Ø of the PAB
       DATA Ø              Read, byte Ø
       LI   R5,1Ø          1Ø bytes to read
       MOV  R4,R6          Get workspace pointer
       AI   R6,>FF6A       Make R6 point to >834A (FAC)
RDPAB  MOVB @>FBFE(R15),*R6+ Read a byte from the VDP, put it in the PAB copy
       DEC  R5             Finished yet?
       JNE  RDPAB          Continue until done
       SZCB @BYTEEØ,@>FF6B(R4) Clear the error flag bits, byte 1 of the PAB
*
* Check for interrupt-driven open call (I/O op code >8Ø)
*
       CB   @BYTE8Ø,@>FF6A(R4) Check >834A for interrupt-driven open
       JNE  NOTINT             If not, skip next 2 lines
       SOCB @BYTEFF,@>FF7D(R4) Store >FF in >835D to remember
       SZCB @BYTE8Ø,@>FF6A(R4) Clear the flag from the PAB copy
NOTINT CB   @>FF6A(R4),@BYTEØ6 Check the op code
       JLE  CODEOK             Skip (OK) if 6 or less. Codes are Ø=OPEN, 1=CLOSE
*                              2=READ, 3=WRITE, 4=RESTORE, 5=LOAD, 6=SAVE
       B    @ERROP             Error if op>6 : delete, etc.
*
* Branch to code that will decode the optional parameters
* and set the device to the proper protocol
*
CODEOK BL   @GETPAR
*
* Branch to code to perform each operation
*
       MOVB @>FF6A(R4),R5 Get the op code from >834A
       SRL  R5,8           Make the code into a 16-bit number
       SLA  R5,1           Double the code: now OPEN=Ø, CLOSE=2, etc.
       MOV  @OPTABL(R5),R5 Read the code pointer for this operation
       B    *R5            Execute the code
*
* Jump table for defined operations
*
OPTABL DATA OPEN           Address of OPEN code
       DATA CLOSE          Close processing is also used to exit normal ops
       DATA READ           Read a record
       DATA WRITE          Write a record
       DATA ERROP          RESTORE is illegal: return illegal operation code
       DATA LOAD           Program LOAD
       DATA SAVE           Program SAVE
*
****************************************
* Code to perform an OPEN on the device
*
* Set the record length
*
OPEN   MOVB @>FF6E(R4),R2 Get the desired record length from >834E
       JNE  ORECLN         Skip if not zero, i.e. has been specified 1 - 255
       BL   @VDPPAB        Set VDP to write into the PAB
       DATA >4ØØ4          starting at byte number 4 (returned record length)
       LI   R2,>5ØØØ       Load default record length byte (8Ø)
       MOVB R2,@>FF6E(R4) Save the default record length in the PAB copy
       MOVB R2,@>FFFE(R15) Write it to the VDP PAB as well
```

```
ORECLN MOVB @>FF68(R4),R1 Get the file type byte from >8348
* If file organization is relative, signal error, else return
       COC  @MFIXED,R1    Check to see whether the file is sequent. or relative
BYTE2Ø EQU  $-4           Convenient byte value >2Ø for use in code
       JNE  RDRTN         If variable, return (uses jump to READ return jump)
       B    @ERROPN       Else signal bad open attribute -- relative file
*
**************************************
* Code for reading a record
*
READ   ABS  R3            If parallel set the flag to +1 (-1 means write access)
       SZCB @BYTEFF,@>FF6F(R4) Clear the number of characters received byte
* Get the buffer pointer and record length
       MOVB @>FF6E(R4),R7 Get the maximum record length
       MOV  @>FF6C(R4),R9 Get the VDP buffer address from the PAB copy
       BL   @QINTER       Check to see if the file is internal type
       JNE  READLN        Skip if display (internal record has count byte first)
       BL   @READCH       Read a character from the port for use as a count
       CB   P7,R6         Check to see if the record is too long for the buffer
       JHE  RECINT        Skip if the length is valid
       B    @ERRBUF       Signal error - insufficient buffer space
RECINT MOV  R6,R7         Transfer the count to the record length
READLN SRL  R7,8          Make the count into a full word integer
       JEQ  RDEXIT        If no more room is free, skip to end of the operation
* Handle each character in the record
RDLOOP BL   @READCH       Get a character from the port
       BL   @QINTER       Test to see if the file is internal
       JEQ  TAKECH        No editing is done on internal characters
       MOVB @>FF78(R4),R1 Check the echo setting
       JEQ  RDEDIT        If echo is on, allow edits on the incoming record
* When the echo is off, just check for the end of the record
       BL   @QFIXED       Check to see if the file is fixed type
       JEQ  TAKECH        If fixed accept the character as is (including CR)
       CI   R6,>ØDØØ      Check for carriage return
       JNE  TAKECH        Echo off, variable - accept char if not CR
       JMP  RDEXIT        Take CR as meaning the end of the record
*
* Display data, echo on - perform editing if requested
*
RDEDIT CI   R6,>ØDØØ      Check for carriage return to end record
       JEQ  RDFIN         On CR handle end of line echoing
* Use DEL to perform backspacing
       CI   R6,>7FØØ      Check for delete character
       JEQ  RDDEL         If so, jump to do it
* Use CONTROL-R to rewrite the line
       CI   R6,>12ØØ      Check for CONTROL-R
       JNE  RDFNFX        If not requesting rewrite, simply echo the character
* Rewrite the line as requested
       MOV  @>FF6C(R4),R1 Get the starting buffer address
       BL   @SETVDR       Set the VDP to read from this address
       BL   @DOEOL        Perform end of line processing (CR, nulls, LF)
       MOV  R9,R2         Get the current buffer in pointer
       S    @>FF6C(R4),R2 Subtract the original buffer address, giving a count
       JMP  REWRLX        Skip to end of loop (in case the line was empty)
* Copy the characters
REWRLP BL   @WRITEV       Transfer one char from the VDP buffer to the device
       DEC  R2            Decrement the line length count
REWRLX JNE  REWRLP        Repeat for each character
       JMP  RDLOOP        Go back to get the next character
*
* Perform deletion of the last character
*
RDDEL  C    @>FF6C(R4),R9 Is the buffer empty?
       JEQ  RDLOOP        Yes - ignore the delete character
```

```
* Delete this character
       INC  R7             Allow one more character to the record
       DEC  R9             Go back one character space in the buffer
       MOV  R9,R1          Get VDP address of the character in R1
       BL   @SETVDR        Set VDP for read of the character
       BL   @WRITEV        Read the character from the VDP then echo it
       CI   R6,>ØDØØ       Was it a carriage return?
       JNE  RDLOOP         Go back to read the next character if it wasn't a CR
       BL   @DOLF          Send nulls and a line feed
       JMP  RDLOOP         Get the next character
*
* Finish processing of the line
*
RDFIN  BL   @QFIXED        Check to see if the file is fixed type
       JEQ  RDFNFX         Handle fixed lines differently
       BL   @DOEOL         Output CR, nulls and LF if needed
       JMP  RDEXIT         Finish read operation
RDFNFX BL   @WRITER        Write the carriage return back out to end the line
*
* Accept the character
*
TAKECH MOV  R9,R1          Get the buffer pointer address
BYTECØ EQU  $-2            Convenient byte value >CØ for use in code
       BL   @SETVDP        Set VDP to write, starting at PAB buffer
       DATA >4ØØØ          Code to do the above
       MOVB R6,@>FFFE(R15) Write the character into the buffer
       INC  R9             Increment the buffer address pointer
       DEC  R7             One less space to put character into
       JNE  RDLOOP         Get the next character if there is still room for it
*
* Finish up the read operation
*
RDEXIT S    @>FF6C(R4),R9 Figure out how many characters were read
       SLA  R9,8           Move the count to the top byte of R9
       MOVB R9,@>FF6F(R4) And save the byte count in the PAB copy
*
* Skip to code that will copy the PAB flags back to VDP
*
RDRTN  JMP  WRRTN          Skip to write return
*
****************************************
* Code to perform write operation
*
WRITE  MOV  R3,R3          Check the serial/parallel flag
       JEQ  WRITES         Skip if writing to a serial port (RS232s)
       SETO R3             Set code to -1 for parallel write (read is +1)
* Set the VDP to read from the buffer and get the record length
WRITES MOV  @>FF6C(R4),R1 Get the VDP buffer address pointer
       BL   @SETVDR        Set the VDP to read from the buffer
       MOVB @>FF6F(R4),R7 Get the record length
       BL   @QINTER        Check to see if the file is internal
       JNE  WRDISP         Skip if display
       MOV  R7,R6          Precede the record with the count byte
       BL   @WRITER        Write tne count byte out from R6 to the port
* Copy each character in the record out
WRDISP SRL  R7,8           Make the record length into a 16-bit count value
       JEQ  WREOL          Skip to end of line processing if the line is empty
WRLOOP BL   @WRITEV        Get the next char from VDP and write it to the port
       DEC  R7             Decrement count
       JNE  WRLOOP         And continue until the whole line is done
*
* End of line processing for writes
```

```
*
WREOL  BL   @QINTER       Check - is the file internal
       JEQ  WRRTN         If internal no CR/LF is needed - skip to exit
       BL   @QFIXED       Check for fixed type file
       JEQ  WRRTN         If fixed no CR/LF is added
       BL   @DOEOL        Output carriage return, nulls and line feed as needed
WRRTN  B    @CLOSE        Skip to CLOSE processing, copying the PAB flags back
*************************
* RS232/3 - third file *
*************************
*
* Code to load a program file from the device
*
LOAD   MOV  @>FF7Ø(R4),RØ Get the maximum buffer size
LCTRLV BL   @WRITEX       Send a SYN (Control-V) to sender to signal ready
       DATA >16ØØ         to receive the file
       LI   R5,7          Send the prompt up to seven times
LDRSPL LI   R1,>CØ1C      Wait delay count (4918Ø decimal)
* Wait for a response to the prompt
LDCHKC BL   @QREADY       Check to see if there is a character ready to be read
       JEQ  LDBEGN        Begin accepting file if response is received
       DEC  R1            Decrease delay count
       JNE  LDCHKC        Continue checking for a response
* Check clear occasionally
       BL   @QCLEAR       Check for the pressing of the CLEAR key to abort loop
       DEC  R5            Decrement count (if CLEAR was pressed exits automatic)
       JNE  LDRSPL        Continue looking for a response
       JMP  LCTRLV        Send the Control-V again
* Data received on the line - try receiving the records
LDBEGN SETO R9            Preset the checksum
* Read the number of bytes to transfer plus its checksum and set the screen
       BL   @RDWCRC       Read top byte of the buffer length, updating checksum
       MOV  R6,R7         Save the character
       BL   @RDWCRC       Read the rest of the block length
       SRL  R6,8          Move the bottom byte of the block length to R6 bottom
       SOC  R6,R7         Or in the bottom byte, giving a 16-bit count
       BL   @RDCRC        Read in the checksum byte(s) to R8
       BL   @SCRNBK       Put the top byte of the buffer length (the block
*                         number) on the top line of the screen
       C    R8,R9         Compare the checksum read to what we think it is
       JEQ  LRCRC1        Skip if the checksum is okay
* Error in record length checksum - ask for it again
       BL   @WRITEX       Write out a NAK (Control-U) to ask for re-send
       DATA >15ØØ         Code for negative acknowledge
       JMP  LDBEGN        Start read over again
* Check that the file will fit
LRCRC1 C    RØ,R7         Check to see if the file will fit in the buffer
       JL   ERRBUF        If not signal 'Out of buffer space' error
* Read in the next block of at most 256 characters
       BL   @WRITEX       Send an ACK (Control-F) accepting the record length
       DATA >Ø6ØØ         Code for acknowledge
LRDBLK BL   @BLKCMP       Update the block count, giving no. of chars to receive
*                         (exits if the operation is complete)
*                         also updates the block number on the screen
LREREA SETO R9            Preset checksum
       MOV  R1Ø,R1        Get a working copy of the current block address
       BL   @SETVDP       Set the VDP to write to the buffer
       DATA >4ØØØ         Flag to indicate write mode
LRDCRS BL   @RDWCRC       Read a character, updating the checksum
       MOVB R6,@>FFFE(R15) Write it into the buffer
       DEC  R7            Decrement the block count
       JNE  LRDCRS        Continue reading the block
```

```
* Check the block for transmission errors
       BL   @RDCRC        Read in the checksum for comparison
       MOV  R3,R3         Check device - serial or parallel
       JEQ  LSERIA        Skip if serial
       BL   @TURNAR       Wait for the line to turn around for parallel
LSERIA C    R9,R8         Compare the computer checksum to that read
       JEQ  LACCPT        Skip if okay
* Reject record because of checksum error
       MOV  @>FF8Ø(R4),R7 Reread the number of bytes in the block
       BL   @WRITEX       Write NAK to signal bad record
       DATA >15ØØ         Negative acknowledge character
       JMP  LREREA        Reread the record
* Accept the record and go on to the next one
LACCPT BL   @WRITEX       Send ACK to signal acceptance of the record
       DATA >Ø6ØØ         Acknowledge character
       AI   R1Ø,>1ØØ      Go on to the next block (buffer address)
MFIXED EQU  $-2           Convenient byte value >Ø1 used as mask for fixed bit
       MOV  @>FF7E(R4),R7 Reset the file size count
       JMP  LRDBLK        Go on to read the next block
*
* Code to save a program file over the device
*
SAVE   MOV  R1Ø,R1        Move a copy of the buffer address to R1
       BL   @SETVDR       Set VDP to read from the buffer
* Wait until a Control-V comes from the receiving computer
SVSYNL BL   @READCH       Read a character from the port
       CI   R6,>16ØØ      Check for a Control-V (SYN) character to start sending
       JNE  SVSYNL        Wait until we get a Control-V from the receiver
* Send out the buffer length
SVSNDL SETO R9            Preset the checksum to >FFFF
       MOV  R3,R3         Serial or parallel operation?
       JEQ  SVSER1        Skip some code if serial operation
       BL   @TURNAR       If parallel, allow time for line turnaround
SVSER1 MOV  @>FF7Ø(R4),R6 Get the number of bytes to send from >835Ø
       BL   @WRWCRC       Write out the top byte of the buffer length, updating
       SWPB R6            the checksum. Switch halves of the buffer length
       BL   @WRWCRC       Write out the least significant byte of the length.
       BL   @WRTCRC       Write out the checksum value
       BL   @READCH       Read a character, looking for accept from the receive
       CI   R6,>Ø6ØØ      Check the character for Control-F (ACK) - acknowledge
       JNE  SVSNDL        Repeat until the receiver acknowledges
* Send a block of at most 256 characters
       MOV  @>FF7Ø(R4),R7 Get number of bytes left to send
SVSNDB BL   @BLKCMP       Compute no. of bytes to send this block and no. left
*                         also writes out progress to the screen
SVRSND SETO R9            Preset checksum value
       MOV  R1Ø,R1        Get start of block address (as set by BLKCMP)
       BL   @SETVDR       Set VDP to read from the block
SVSNDC MOVB @>FBFE(R15),R6 Get the next char from the buffer (from VDP)
       BL   @WRWCRC       Write out the character, updating the checksum
       DEC  R7            Decrement the block count
       JNE  SVSNDC        Jump if more characters are to be written
       BL   @WRTCRC       Write out the checksum value for error testing
       BL   @READCH       Read the response to this block
       CI   R6,>Ø6ØØ      Check for Control-F (ACK) acknowledging correctness
       JEQ  SVNXTB        Skip if okay to code for next block
       MOV  R3,R3         Check if device is serial or parallel
       JEQ  SVSER2        Skip code if serial
       BL   @TURNAR       For parallel device, wait for the line to turnaround
SVSER2 MOV  @>FF8Ø(R4),R7 Get a new copy of the current block length from >836Ø
       JMP  SVRSND        Send the block again
```

```
* Move on to the next block
SVNXTB AI   R1Ø,256       Skip to the start of the next block
       MOV  @>FF7E(R4),R7 Get the number of characters left from >835E
       JMP  SVSNDB        Go on to the next block
*
* Error exit addresses
*
ERROPN LI   R1,2*>2ØØØ    Error 2 - bad open attribute
       JMP  SETERR        Put error code in PAB
*
ERROP  LI   R1,3*>2ØØØ    Error 3 - operation not supported
       JMP  SETERR        Put error code in PAB
*
ERRBUF LI   R1,4*>2ØØØ    Error 4 - out of buffer or table space
       JMP  SETERR        Put error code in PAB
*
ERRDEV LI   R1,6*>2ØØØ    Error 6 - device error
*
* Put error value in the PAB and exit
SETERR SOCB R1,@>FF6B(R4) Or the error code into >834B - flag byte of PAB copy
*
* CLOSE code - also used for normal exit processing and error processing
*
CLOSE  BL   @VDPPAB       Set the VDP to write to the PAB flag byte address
       DATA >4ØØ1         Code to signal write to the flag byte
       MOVB @>FF6B(R4),@>FFFE(R15) Write the new flag byte to VDP
       BL   @VDPPAB       Set the VDP to write to the record length in the PAB
       DATA >4ØØ5         Code to signal write to the record length byte
       MOVB @>FF6F(R4),@>FFFE(R15) Write the new record length byte to VDP
       INCT @>FF84(R4)    Increment the return address to signal device found
*                         (DSRLNK treats a direct return as device not found)
*
* Exit, resetting the CRU base and the device
*
ERRDSR ANDI R12,>FFØØ     Reset the CRU base to >98Ø (*2)
*                         (alternate entry also used for device not present,
*                         which is signalled by a direct return to DSRLNK - see
*                         the code above at CLOSE for normal return)
       MOV  @>FF84(R4),R11 Get the return address
       SBO  2             Make sure the parallel data strobe is off
       SBZ  1             Reset parallel device to output mode
       SBZ  7             Turn off the light
       RT                 Return to DSRLNK code
*
* Code to read parameters following the device name and to
* set the 99Ø2 to use those parameters for I/O
*
GETPAR MOV  R11,@>FF86(R4) Save the return address
* Get table address for legal parameters
       BL   @QSVLD        Test the op code for save/load
       JEQ  PRSVLD        Use a different table if save or load
       LI   R8,PARMOP     Get pointer to table of options for open/read/write
*
* Load default options for 99Ø2 control register. Bits used are:
*    xx - Stop Bits: ØØ=1+1/2, Ø1=2, 1x=1 Stop Bit
*      xx - Parity: Øx=none, 1Ø=even, 11=odd
*         x - CLK4M, clock rate divisor: Ø->3, 1->4. See explanation at BAUDS
*          x - unused
*           xx - Data bits: ØØ=5, Ø1=6, 1Ø=7, 11=8 Data Bits
*    xxxx xxxx - This value is stored temporarily at >83ØA
*                 Transfer rate settings are stored at >83ØE,F
*
```

```
       LI   R1,>82ØØ       Default options: 1 stop bit, odd parity, 7 data bits
       JMP  PRESBD         Skip to baud rate processing
* Table address and default parameters for save/load
PRSVLD LI   R8,PARMSV      Get pointer to table of parms allowed for save/load
       LI   R1,>83ØØ       Default options: 1 stop bit, no parity, 8 data bits
* Preset baud rate and default values
PRESBD LI   R5,3ØØ         Load default baud rate
       MOV  R4,R9          Get copy of workspace register (for relocatable code)
       AI   R9,>FFFA       Subtract 6 - uses >830A to >83DF for temporary storage
       MOVB R1,*R9         Save the bit settings in >830A
       BL   @BAUDRT        Calculate baud rate settings, leave in >830A, >83DE,F
       MOVB @>FF73(R4),RØ Get the length of the device specifier string
       SRL  RØ,8           Turn it into a 16-bit number
       S    @>FF74(R4),RØ Subtract the number of chars used on the device name
       JLE  PARXIT         Skip if there are no parameters to look at
* Get the next parameter and process it
       MOV  @>FF76(R4),R1 Get the pointer to the rest of the device specifier
       BL   @SETVDR        Set the VDP to read from the device specifier
       SETO R6             Preset R6 to an unused character
NXTPAR MOV  RØ,RØ          Check to see if there are more chars in the specifier
       JEQ  PARXIT         No more characters - exit
       BL   @SKIPCH        Skip past the next period (as follows)
       BYTE '.',Ø          Character to be scanned to
       JEQ  PARXIT         Exit if a period could not be found
* Find the next parameter in the table
       MOV  R8,R7          Get a copy of the table pointer for legal parameters
       SRL  R6,8           Move the first char of the parameter to R6's bottom
       MOVB @>FBFE(R15),R6 Get the second char of the parameter name
       DEC  RØ             Decrement device specifier length count
       SWPB R6             Swap R6 to get the parameter 2 character code name
PRLOOK MOV  *R7+,R1        Get the next parameter name from the table
       JEQ  PARERR         If table is exhausted, signal parameter error
       MOV  *R7+,R2        Get the code address for this parameter
       C    R1,R6          Compare this name to the table name
       JNE  PRLOOK         Continue looking for the parameter name
       B    *R2            Execute the code associated with this parameter
* Exit parameter processing
PARXIT MOVB @>FF6A(R4),R1 Get the operation code
       JEQ  PRXOPN         Skip if an open call
       BL   @QSVLD         Test to see if it is a save or load operation
       JNE  PRXNSV         Skip if not save/load
       BL   @SCRNPS        For save/load: preset the screen with block no. = 255
       MOV  @>FF6C(R4),R1Ø Move the buffer address to R1Ø
PRXOPN BL   @SETDEV        Set up the device characteristics
PRXNSV JMP  CRCEX          Return
* Parameter name not recognised
PARERR JMP  ERROPN         Signal error on open if a parameter is bad
*
* Code to handle each of the parameters
*
PARMEC LI   R1,>FF78       User selected echo off parameter - set >8358 non-zero
       JMP  PONOFF         Skip to on/off code
*
PARMCR LI   R1,>FF79       User selected CR option (CRLF suppressed) set in >8359
       JMP  PONOFF         Skip to on/off code
*
PARMLF LI   R1,>FF7A       User selected LF option (LF suppressed) set in >835A
       JMP  PONOFF         Skip to on/off code
*
PARMNU LI   R1,>FF7C       User selected null option - set in >835C
       JMP  PONOFF         Skip to on/off code
*
```

```
PARMCH LI   R1,>FF7B        User selected check parity option - set in >835B
* Turn the selected parameter on or off
PONOFF A    R4,R1           Add the workspace offset to the pointer address
       SOCB @BYTEFF,*R1     Move >FF to the parameter value table
       JMP  PARMFN          Skip to finish interpretation for this parameter
*
* Baud rate parameter selected
*
PARMBA MOV  R3,R3           Check to see if the device is serial or parallel
       JNE  PARMFN          If parallel, skip the parameter
       BL   @RDBAUD         Read the baud rate and set the configuration to i1
       JMP  PARMFN          Finish parameter processing
*
* Parity parameter selected
*
PARMPA MOV  R3,R3           Check to see if the device is serial or parallel
       JNE  PARMFN          If parallel, skip the parameter
       BL   @SKIPCH         Skip past the equals sign which precedes the valu(
       BYTE '=',Ø           Data value for the equals sign
       JEQ  PARERR          Signal error if an equals sign was not found
       SZCB @BYTE3Ø,*R9     Clear the parity option bits in the configuration
       SRL  R6,8            Shift the value character (N, E or O) to R6 bottor
       CI   R6,'N'          Check for no parity desired
       JEQ  PARMFN          If no parity, setting is right already - exit
       CI   R6,'E'          Check for even parity desired
       JEQ  EVENPA          If so, jump to save the setting
       CI   R6,'O'          Check for odd parity
       JNE  PARERR          If not legal selection, signal error
       SOCB @BYTE3Ø,*R9     Set both bits to configure for odd parity
       JMP  PARMFN          Exit parameter processing
EVENPA SOCB @BYTE2Ø,*R9     Set configuration to even parity option
       JMP  PARMFN          Exit parameter processing
*
* Number of data bits selected
*
PARMDA MOV  R3,R3           Check the device - serial or parallel
       JNE  PARMFN          If the device is parallel, skip the parameter
       BL   @SKIPCH         Skip past the equals sign
       BYTE '=',Ø           Data value for the skip routine
       JEQ  PARERR          Signal error if the equals sign is not found
       BL   @NUMBER         Read in a numeric parameter value
       SOCB @BYTEØ3,*R9     Preset the number of data bits to 8 (11=8, 1Ø=7)
       AI   R5,-7           Check to see if the value desired is 7
       JEQ  DATAB7          Jump if 7 data bits wanted
       DEC  R5              Ensure that if not 7, 8 was selected
       JNE  PARERR          If not 7 or 8 signal error
* 8 data bits wanted
       JMP  PARMDX          Exit data bit setting
DATAB7 SZCB @BYTEØ1,*R9     Change data bit setting to 1Ø to select 7 bits
PARMDX JMP  PARMFN          Finish processing for this parameter
*
* Two stop bits selected
*
PARMTW SZCB @BYTECØ,*R9     Clear stop bit configuration (Ø1=2 bits, 1x=1 bit
       SOCB @BYTE4Ø,*R9     Set for 2 stop bits
PARMFN JMP  NXTPAR          Go on to the next parameter in the device specifi
*
* Read the checksum value from the port
*
```

```
RDCRC  MOV  R11,@>FF86(R4) Save the return address
       BL   @READCH        Read the first character of the checksum
       MOV  R6,R8          Save the top byte
       BL   @READCH        Read the bottom byte of the checksum
       SWPB R6             Move the bottom byte to the bottom of R6
       SOC  R6,R8          Or together the two checksum bytes to get a 16-bit CRC
       JMP  CRCEX          Exit reading of the checksum value
*
* Write the checksum value to the port
*
WRTCRC MOV  R11,@>FF86(R4) Save the return address
       MOV  R9,R6          Get a copy of the CRC checksum
       BL   @WRITER        Write the top byte out to the port
       SWPB R6             Switch bytes
       BL   @WRITER        Write the bottom byte out to the port
       JMP  CRCEX          Exit writing of the checksum value
*
* Read a character from the port, updating the checksum
*
RDWCRC MOV  R11,@>FF86(R4) Save the return address
       BL   @READCH        Read in the character
       JMP  UPDCRC         Skip to update the checksum
*
* Write a character to the port, updating the checksum
*
WRWCRC MOV  R11,@>FF86(R4) Save the return address
       BL   @WRITER        Write the character out to the port
UPDCRC BL   @CRCALC        Add the character to the checksum calculation
CRCEX  MOV  @>FF86(R4),R11 Restore the return address
       RT                  Return (no tests performed)
*
* Read the baud rate setting parameter value
*
RDBAUD MOV  R11,@>FF88(R4) Save the return address
       BL   @SKIPCH        Skip past the equals sign
       BYTE '=',Ø          Data value (equals sign) for skip routine
       JEQ  PARERR         Signal error if an equals sign was not found
       BL   @NUMBER        Read in the baud rate from the string
       JMP  BAUDRE         Flow into baud rat3 setting code
*
* Figure out the speed settings for the selected baud rate
*
BAUDRT MOV  R11,@>FF88(R4) Save the return address
BAUDRE LI   R1,BAUDS       Load pointer to the top of the baud rate table
BYTEØ1 EQU  $-3            Convenient byte with value >Ø1
       CLR  R2             Clear offset count (how far down in the table?)
BAUDLP MOV  *R1+,R11       Get the next baud rate
       JEQ  PARERR         Signal parameter error if no rates are left
       C    R5,R11         Compare this rate with the desired rate
       JEQ  BAUDLK         Skip if the rate was found
       INCT R2             Increment the offset into the baud rate tables
       JMP  BAUDLP         Continue through the baud rate table
BAUDLK MOVB @12,R11        Get the computer clock rate from >ØØØC
BYTEEØ EQU  $-3            Convenient byte value >EØ for use in other code
       SRL  R11,8          Make the cycle rate a 16-bit number
       LI   R1,SPEEDS      Load a pointer to the clock rate speed table
BAUDL2 MOV  *R1+,R5        Get the next clock speed
       JEQ  CERROR         Signal error if clock rate could not be found
       C    R5,R11         Compare this rate to the machine's rate
       JEQ  BAUDEX         If the rate was found, skip to use it
       INCT R1             Skip the 99Ø2 rate setting table pointer
       JMP  BAUDL2         Continue looking for the clock speed
```

```
BAUDEX A    *R1,R2        Add the 99Ø2 setting table address to the offset
       MOV  *R2,R1        Get the settings for the 99Ø2 transmit speed
       JGT  BCLK4M        Skip if the clock divisor is to be 3 rather than 4
       SOCB @BYTEØ8,*R9   Set the CLK4M bit to indicate division of clock by 4
       ANDI R1,>7FFF      Get the countdown rate for transmit/receive
       JMP  BAUDCS        Skip to save the countdown rate
BCLK4M SZCB @BYTEØ8,*R9   Clear the CLK4M bit to select division of clock by 3
BAUDCS MOV  R1,@>FFFE(R4) Save the countdown rate register in >830E,F
       JMP  CHEXIT        Return (no test performed)
***********************
* RS232/4 - last file *
***********************
*
* Read a character from the port
* Returns with the character in the top byte of R6
*
READCH MOV  R11,@>FF88(R4) Save the return address
READC1 BL   @QREADY       Test to see if there is a character ready to be read
       JEQ  READIT        If a character is available, jump to read it
       BL   @QCLEAR       Check to see if the CLEAR key is pressed
       JMP  READC1        Keep trying to read the character
* Get the character
READIT MOV  R3,R3         Test the port - serial or parallel
       JNE  READPL        If parallel, jump to the code for PIO
* Read a character from the serial port
       CLR  R6            Clear the receiving register
       STCR R6,8          Read the character from the 99Ø2
       SBZ  18            Reset the receiver buffer register full flag
       TB   11            Check for receiver overrun (chars too frequent)
       JEQ  CERROR        If so, signal error
       TB   12            Check for framing error (data bit setting incorrect)
       JEQ  CERROR        If so, signal error
       MOVB @>FF7B(R4),R11 Check to see if parity check is enabled (CH option)
       JEQ  CHEXIT        If not, accept the character as received
       TB   1Ø            Check for parity error
       JNE  CHEXIT        If no error, jump to normal exit
* Character was received incorrectly
CERROR B    @ERRDEV       Signal device error
* Code to read from the parallel port
READPL SBO  1             Set the port to read data
       SBZ  2             Turn the data strobe on, signalling ready to receive
READPW TB   2             Check the BUSY/ACK line
       JNE  READPC        If a character is ready, jump to read it
       BL   @QCLEAR       Test to see if the CLEAR key is pressed
       JMP  READPW        Wait until the character arrives
READPC CLR  R6            Clear the receiving register
       MOVB @>5ØØØ,R6     Read the character to the top of R6
       SBO  2             Turn the data strobe off
CHEXIT MOV  @>FF88(R4),R11 Restore the return address
       RT                 Return
*
* Compute the next block number and number of bytes to transfer
* Block length is left in R7 and also in >836Ø
* Number of chars left to go is left in >835E
*
BLKCMP MOV  R11,@>FF88(R4) Save the return address
       MOVB R7,R7         Check the number of blocks left in the buffer (R7 top
       JEQ  BLKZER        If on the last block, jump to code to handle it
       BL   @SCRNBK       Write the block number out to the screen
       AI   R7,-256       Allocate 256 bytes in this block
       MOV  R7,@>FF7E(R4) Put the now buffer length in >835E
       LI   R7,256        Reload R7 as block length
       JMP  BLKCEX        Jump to exit code
* Code for the last block - block length less than 256
BLKZER MOV  R7,R7         Check the block length
       JNE  BLKZRA        Continue if there are more bytes to process
       B    @CLOSE        Finished the save/load: No more bytes
BLKZRA CLR  @>FF7E(R4)    Clear the number of chars to transfer after this block
BLKCEX MOV  R7,@>FF8Ø(R4) Move the block length to >836Ø
       JMP  CHEXIT        Use the preceding exit code to return
*
* Write out the screen with the block number
*
SCRNPS SETO R7            For initial screen, use block number 255
SCRNBK MOV  R11,@>FF8A(R4) Save the return address
       CLR  R1            Set R1 to point to the start of the video display (Ø)
       BL   @SETVDP       Set VDP to write to the video display
       DATA >4ØØØ         Code number for write, starting address Ø
       BL   @WBLNKS       Write 14 blanks out to the display
       MOV  R7,R2         Get a copy of the remaining buffer length
       SRL  R2,8          Calculate the block number
       LI   R6,1ØØ        Start output at the hundreds digit
* Put out one digit of the number
PUTDIG CLR  R1            Clear the top of the 32-bit dividend
       DIV  R6,R1         Divide the block number by 1ØØ
       AI   R1,'Ø'        Convert the digit to an ASCII character
       SLA  R1,8          Shift the char to the top byte of the register
       AB   @>FF72(R4),R1 Add the screen offset (>6Ø for Basic) from the PAB
       MOVB R1,@>FFFE(R15) Move tne character to the screen area of VDP
       CLR  R5            Clear the top register of the units
       DIV  @NUM1Ø,R5     Divide the units factor by 1Ø to move to the next unit
       MOV  R5,R6         Move the new unit factor to R6
       JNE  PUTDIG        Put out the next digit
* Exit
       BL   @WBLNKS       Write out 14 blanks (to clear the line)
       JMP  NULXIT        Return (using other code)
*
* End an output line with CR/nulls/LF (if needed)
*
DOEOL  MOV  R11,@>FF8A(R4) Save the return address
       MOVB @>FF79(R4),R11 Get the CRLF option from >8359
       JNE  NULXIT        Exit if CR/LF is not desired
       BL   @WRITEX       Write out a carriage return
       DATA >ØDØØ         Carriage return character
       JMP  DOLFA         Flow into the nulls/LF code
*
* End an output line with nulls/LF (if selected)
*
DOLF   MOV  R11,@>FF8A(R4) Save the return address
DOLFA  MOVB @>FF7C(R4),R1 Check the NULLS option
       JEQ  DONONL        Skip if no nulls are desired
* Write out six nulls to allow time for a carriage return
       LI   R5,6          Get the number of nulls to put out
DONULL BL   @WRITEX       Write out a null (ASCII character Ø)
       DATA >ØØØØ         ASCII character NUL (Ø)
       DEC  R5            Decrement the character count
       JNE  DONULL        Continue writing nulls until six are sent
* Write a line feed if selected
DONONL MOVB @>FF79(R4),R1 Check the CRLF option
       JNE  NULXIT        Skip if CRLF was specified (i.e. not desired)
       MOVB @>FF7A(R4),R1 Check the LF option
       JNE  NULXIT        Skip if the LF option was specified (not desired)
       BL   @WRITEX       Write out a line feed
       DATA >ØAØØ         ASCII line feed character LF - 1Ø
```

```
NULXIT MOV  @>FF8A(R4),R11 Restore the return address
       RT                  Return
*
* Get the operation code & subtract 6, testing for save/load
* EQ flag is set if the operation is save or load
*
QSVLD  MOVB @>FF6A(R4),R1 Get the operation code from the PAB copy
       SRL  R1,8           Make it a 16-bit number
       AI   R1,-5          Subtract five
       JEQ  >473E          Return if load code
       DEC  R1             If save code, make it zero
       RT                  Return EQ if save/load
*
* Test to see if the file is internal or display
* EQ flag is set if the file is internal
*
QINTER MOVB @>FF6B(R4),R1 Get the file flag byte
       COC  @BYTEØ8,R1     Check to see if the internal bit is set
       RT                  Return EQ if internal type file
*
* Test to see if the file is fixed or variable
* EQ flag is set if the file is fixed
*
QFIXED MOVB @>FF6B(R4),R1 Get the file flag byte
       ANDI R1,>1ØØØ       Check the proper bit
       RT                  Return EQ if the file is fixed
*
* Read a number from the parameter string
* Value is returned in R5
*
NUMBER MOV  R11,@>FF8A(R4) Save the return address
       CLR  R1             Preset the number as zero
       CLR  R11            Use R11 as a flag to indicate if there was a no.
       JMP  RDDIGT         Use the first digit (already in R6)
* Add the next digit to the number
RDNXTD MOVB @>FBFE(R15),R6 Get the next digit from the VDP
       DEC  RØ             Decrement the remaining length of the file speci
RDDIGT MOV  R6,R7          Move the digit to R7 for temporary use (may be p
* Convert the ASCII character to a digit
       SRL  R7,8           Turn the ASCII character into a 16-bit number
       AI   R7,-'Ø'        Subtract the code for Ø, to make it a real digit
       JLT  RDNOTD         Jump if not a digit ( < 'Ø' )
       CI   R7,9           Check to see if it is not too big for a digit
       JH   RDNOTD         Jump if not a digit
       INC  R11            Increment R11 - non-zero if at least one digit f
* Add the new digit to the number
       MPY  @NUM1Ø,R1      Multiply the old value by 1Ø (goes to R2)
       MOV  R1,R1          Check for overflow (past a 16-bit number)
       JNE  RDOVRF         Signal error if overflow occurred
       A    R7,R2          Add the new digit to the number
       MOV  R2,R1          Move the number back to R1
       MOV  RØ,RØ          Check the file specifier length
       JNE  RDNXTD         If more characters available, get the next digit
* When a non-digit is reached, check for valid end of a number
RDNOTD MOV  R11,R11        Check to see if any digits were found
       JNE  RDCHKP         Yes, there were digits - skip to end code
RDOVRF B    @ERROPN        Signal error in open parameters
RDCHKP MOV  R1,R5          Move the number to R5 (where the answer is expec
       MOV  @>FF8A(R4),R11 Restore the return address
       RT                  Return with answer in R5
NUM1Ø  DATA 1Ø
*
```

```
* First byte after the call is the character to scan to
* EQ flag is set if the character is not found
*
SKIPCH MOV  *R11+,R5      Get the desired character (increment return address)
       CB   R5,R6         Check to see if already there
       JEQ  SKIPEX        If already there, exit
SKIPNX MOVB @>FBFC(R15),R6 Get the next character from the string
       DEC  RØ            Decrement file specifier length
       CB   R5,R6         Check to see if this is the character we want
       JEQ  SKIPEX        Found - skip to the exit code
       MOV  RØ,RØ         Check the number of chars left in the file specifier
       JNE  SKIPNX        If there are more, continue skipping
* Found the desired character
SKIPEX MOV  RØ,RØ         Check the number of chars left in the specifier
       JEQ  GTCHRX        Return EQ if out of chars
*
* Get the next non-blank character from the parameter string
*
GETCHR CLR  R6            Clear the receiving register
       MOVB @>FBFE(R15),R6 Read the next byte from VDP (the file specifier)
       DEC  RØ            Decrement the count of characters left
       CI   R6,>2ØØØ      Check to see if the character is a blank space
       JEQ  SKIPEX        If blank, go back and get another character
GTCHRX RT                 Return
*
* Perform CRC-CCITT cyclical redundancy checksum, a 16-bit checksum
* This code adds one character (top R6) to the checksum (R9)
* The polynomial is described as x**16+x**12+x**5+1
*
CRCALC MOV  R6,R1         Move tne character to R1 for working storage
       ANDI R1,>FFØØ      Mask off the character
       XOR  R1,R9         XOR the character into the checksum
       MOV  R9,R1         Use another copy of the new checksum for further work
       SRL  R1,4          Shift for x**12 operation
       XOR  R9,R1         XOR in the x**12 op
       ANDI R1,>FFØØ      Mask off the top byte
       SRL  R1,4          Shift another 4 bits
       XOR  R1,R9         XOR into checksum again
       SRC  R1,7          Generate the x**5 element
       XOR  R1,R9         XOR the x**5 element in
       SWPB R9            Swap bytes of the result
       RT                 Return with checksum updated in R9
*
* Write a character to tne port - entries for VDP, register, ROM source
*
* Read a character from VDP then write it to the port
WRITEV MOVB @>FBFE(R15),R6 Read the character from VDP
       JMP  WRITER        Use the 'write from register' code
* Read the character following the call and write it to the port
WRITEX MOV  *R11+,R6      Read the character desired (increment return address)
WRITER MOV  R11,@>FF8C(R4) Save the return address
WRITDV MOV  R3,R3         Check the device - is it serial or parallel?
       JNE  WRITEP        Skip if it is parallel
* Write a character to a serial port
       SBO  16            Turn on the RTS (Request To Send) line
       TB   27            Test for DSR (Data Set Ready)
       JNE  WRITWT        Jump if set - ready to receive a character
       TB   22            Test to see if the transmit buffer register is empty
       JEQ  WRITS2        If it is empty, skip to write the character
```

```
WRITWT BL   @QCLEAR        Test to see if the clear key is pressed
       JMP  WRITDV         Wait until the character can be sent
WRITS2 LDCR R6,8           Write the character to the port
       SBZ  16             Turn off RTS. The 99Ø2 keeps it on until the char
WRITXT MOV  @>FF8C(R4),R11 Restore the return address
       RT                  Return
* Write a character to a parallel port
WRITEP SBZ  1              Set the port for writing
       TB   2              Test the BUSY/ACK line
       JEQ  WRITWT         Wait until the port is ready to receive
* Port is ready to accept the character
       MOVB R6,@>5ØØØ      Write the character to the data lines
       SBZ  2              Set the DATA STROBE
WRITPW TB   2              Test the BUSY/ACK line
       JEQ  WRITPF         If acknowledged, skip to exit
       BL   @QCLEAR        Check to see if the CLEAR key is pressed
       JMP  WRITPW         Wait until the character is acknowledged
WRITPF SBO  2              Turn the DATA STROBE off
       JMP  WRITXT         Return
*
* Set the devices as directed by the parameters
*
SETDEV MOV  R3,R3          Check to see if the device is serial or parallel
       JEQ  SETSER         Skip to code for serial devices if needed
* Preset a parallel port
       SBO  2              Turn the DATA STROBE off
       SBZ  1              Preset the device for writing
       RT                  Return
* Preset a serial port (99Ø2)
SETSER SBO  31             Reset the 99Ø2
       LDCR @>FFFA(R4),9   Load the control register (data bits, parity, etc.
       SBZ  13             Clear loading of the interval register
       LDCR @>FFFE(R4),12  Load both the transmit and receive data rate regis
       MOVB @>FF7D(R4),R1  Test to see whether the device is interrupt-driver
       JEQ  SETSRX         If not, skip
       SBO  18             Interrupt-driven: set receiver interrupts on
SETSRX RT                  Return
*
* Routines to set VDP to read or write from an address
*
* Set VDP to read from a byte in the PAB (byte follows in-line)
VDPPAB MOV  @>FF76(R4),R1  Get the pointer to the end of the device string
       S    @>FF74(R4),R1  Subtract the number of character in the device nam
       AI   R1,-1Ø         Subtract 1Ø to get to the start of the PAB
* Set VDP to read/write as directed by code following
SETVDP A    *R11+,R1       Add the code after the call to the register pointe
* Set VDP to read/write as selected in the register
SETVDR MOVB @3(R4),*R15    Move the bottom of R1 to the VDP address register
       NOP                 Wait
       MOVB R1,*R15        Move the top of R1 to the VDP address register
       RT                  Return
*
* Write 14 blanks to VDP
*
WBLNKS LI   R1,'  '        Load a blank code
       AB   @>FF72(R4),R1  Add the character offset (byte 8 of PAB)
       LI   R2,14          Load number of chars to write
WBLNK  MOVB R1,@>FFFC(R15) Write one blank out to VDP
       DEC  R2             Decrement the count
       JNC  WBLNK          Continue until all blanks have been written
       RT                  Return
*
```

```
* Check to see whether the port has a character ready to be read
* EQ bit is set if there is a character ready to be read
*
QREADY MOV  R3,R3          Test the port - is it serial or parallel
       JNE  QREADP         Jump for parallel test
* Test the 99Ø2 for a character
QSRRDY TB   27             Check the DSR (Data Set Ready) line
       JNE  QSRRDX         Return if not set
       TB   21             Check the Receive Buffer Register Full flag
QSRRDX RT                  Return EQ if a character is available to be read
* Test the parallel port for a character
QREADP TB   2              Test the BUSY/ACK line
       RT                  Return EQ if a character is available
*
* Test to see if the CLEAR key is pressed
* (If it is pressed a device error will be raised)
*
QCLEAR MOV  R12,R1         Save the CRU address
       LI   R12,>24        Load the keyboard select CRU base address
       LDCR @KFUNCT,3      Set the keyboard to set function key row active
       TB   -11            Test CRU bit 7 (>24/2-11) for FCTN key pressed
       JEQ  QCLEAX         Skip if FCTN is 1 - not pressed
       LDCR @KROW4,3       Set the keyboard to set key row with 4 in it active
       TB   -11            Test to see if 4 key is pressed (Ø)
       JNE  CLEARX         CLEAR pressed - exit processing
QCLEAX MOV  R1,R12         Restore the CRU base
       RT                  Return only if CLEAR is not pressed
* Exit all processing if CLEAR is pressed
CLEARX MOV  R1,R12         Restore the CRU base
       B    @ERRDEV        Signal device error
*
* Test data - used to find the end of the ROM
*
       DATA >ABCD          Test data only
*
* Wait to allow time for the parallel port to switch from output to input
* This is used to ensure timing is okay for save/load
*
TURNAR SRC  RØ,8           Wait 3Ø cycles (2 extra for memory access to ROM)
       SRC  RØ,8           This waits 1Ø microseconds
       SRC  RØ,8           As above
       SRC  RØ,8           As above
       SRC  RØ,8           As above
       SRC  RØ,8           As above
       RT                  Return after waiting 6Ø microseconds
```

THE TIHOME SOFTWARE COLLECTION

Pricing for TSC disks has been reorganised, although the full Collection is still offered at £35 inclusive.

Check the TSC Catalogue (available free to ITUG subscribers on request) to find out what the TSC entries stand for. Make sure that you indicate clearly what your choice is, specifying the name of the disk/s you want.

GAMES:	TSC ENTRIES:
TSC-DSK-A	GA0001 - GA0011
TSC-DSK-B	GA0012 - GA0022
TSC-DSK-C	GA0023 - GA0032
TSC-DSK-D	GA0033 - GA0043
TSC-DSK-E	GA0044 - GA0055
TSC-DSK-F	GA0056 - GA0066
EDUCATIONAL:	
TSC-DSK-G	ED0001 - ED0009
TSC-DSK-H	ED0010 - ED0017
DEMONSTRATION:	
TSC-DSK-I	DE0001 - DE0006
MUSIC:	
TSC-DSK-J	MU0001 - MU0009
TSC-DSK-K	MU0010 - MU0017
TSC-DSK-L	MU0018 - MU0026
UTILITIES:	
TSC-DSK-M	UT0001 - UT0011
TSC-DSK-N	UT0012 - UT0022
TSC-DSK-O	UT0023 - UT0033
TSC-DSK-P	UT0034 - UT0044

PRICING

The number of programs on a disk can vary between about 7 and 11, dependent upon the sizes of the programs in terms of sectors used.

The exception is the DEMONSTRATION disk, which alone is offered at £2.95 inclusive of post and packing

The rest are priced as follows:

£ 3.95	for ANY	1 DISK	
£ 6.90	for ANY	2 DISKS	(SAVE £ 1)
£ 9.85	for ANY	3 DISKS	(SAVE £ 2)
£12.80	for ANY	4 DISKS	(SAVE £ 3)
£14.75	for ANY	5 DISKS	(SAVE £ 5)
£16.70	for ANY	6 DISKS	(SAVE £ 7)
£18.65	for ANY	7 DISKS	(SAVE £ 9)
£20.60	for ANY	8 DISKS	(SAVE £11)
£22.55	for ANY	9 DISKS	(SAVE £13)
£24.50	for ANY	10 DISKS	(SAVE £15)
£26.45	for ANY	11 DISKS	(SAVE £17)
£28.40	for ANY	12 DISKS	(SAVE £19)
£30.35	for ANY	13 DISKS	(SAVE £21)
£32.30	for ANY	14 DISKS	(SAVE £23)
£34.25	for ANY	15 DISKS	(SAVE £25)
£35.00	ENTIRE		(SAVE £28.20)

All programs are recorded on Single-sided disks.

You may elect to be supplied with software on Double-sided disks, when you should deduct £1 from the prices shown above.

If you prefer, you may send in your own disks, in which case deduct 50p for each disk from the prices shown above.

For example, 8 disks supplied by you would work out at £20.60 - (8 x 50p = £4) = £16.60 nett.

All the above prices are inclusive of post and packing.

The pricing for programs recorded on cassette remains £1 per program, with an overall charge of 65p for post and packing.

PLEASE MAKE ALL CHEQUES PAYABLE TO "PETER BROOKS"