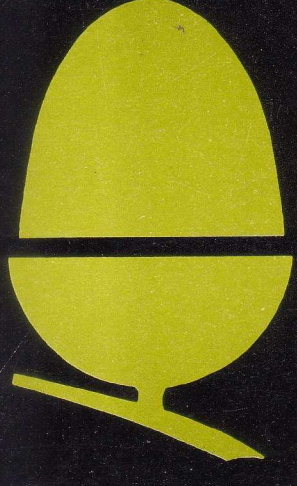
ACORN 6809
User's Manual



ACORN 6809

TECHNICAL AND PROGRAMMING *MANUAL*

© COPYRIGHT ACORN COMPUTER LTD 1980

ISSUE 1 FEB 1980

1.0 INTRODUCTION

The 6809 monitor is designed to operate at two different levels. At one level it provides all the commands necessary for the efficient writing, and debugging, of machine-code programs, and commands for saving and loading programs to and from cassette tapes. At a second level, the monitor has been designed with future expansion in mind, so that it will form the kernel of a more sophisticated operating system.

1.1 Monitor Commands

The monitor commands are all single-letter mnemonics followed by a number of parameters which are optional, and when these parameters are ommitted then default values are assumed. Commands are looked up in a table, and this table may be supplemented, or replaced, by external tables supplied by the user. Thus user-written commands can be linked in with the standard monitor commands to increase the capabilities of the monitor as desired; examples will be given later.

1.2 Device Drivers

The monitor program includes driver routines for console input/output (i. e. keyboard and display), cassette input/output, and printer output. The addresses of these routines are held in RAM and all calls to them are made via these addresses; thus user-written driver routines can be substituted for any of the monitor routines to enable the monitor to be used with different devices.

1.3 Keyboard

The keyboard in the monitor is interrupt driven, making operation of the keyboard independent of the operation of other programs. Characters

entered at the keyboard are displayed on the screen and buffered in memory, even if a user program is running. Up to 80 characters may be typed ahead, and the console input routine will automatically supply successive characters from the buffer. Two modes of operation are provided for: in 'buffered' mode, the mode in which the monitor normally operates, characters cannot be read from the buffer until the line being typed has been terminated by 'return'. In this mode of operation mistakes can be erased by typing 'rubout'. In the 'unbuffered' mode, characters can be read from the buffer as soon as they are typed at the keyboard.

1.4 Display

The display is driven by software, and includes automatic scrolling and a flashing-underline cursor. The interpretation of control characters is performed by a look-up table which may be extended, or replaced, by the user. The codes for 'return', 'line-feed', 'rubout', and 'form-feed' (clear screen), which are already implemented, can thus be added to by the user. The graphics facilities of the VDU can be used with the monitor's routines just like the standard alphanumeric characters.

1.5 Command Passing

Although the monitor normally reads commands from the keyboard buffer, routines are provided so that a line of commands can be passed to the monitor, as a text string, by a user program. It is thus possible for user programs to use the full facilities of the monitor in a very simple manner.

i.6 Interrupts

Interrupts are handled by the monitor in a very flexible manner.

All interrupts are vectored via addresses held in RAM. These addresses can be replaced by the user to redefine some, or all, of the interrupt service routines. Thus all the interrupts are potentially available for user applications with a minimal overhead.

1.7 DISK Operating System

The Acorn 6809 card is fully supported by a range of memory and interface cards, and so may be expanded into a complete computer system. With future expansion in mind, the 6809 monitor has been provided with all the routines necessary for loading a disk operating system from a minifloppy disk drive; thus it will, without modification, form the basis of a much larger system.

2.0 MONITOR OPERATION

Connect power and press reset. If all is well an asterisk will appear in the top left-hand corner of the screen, followed by a flashing bar. The asterisk is the monitor prompt; it indicates that the monitor is in control, and is waiting for input. The flashing bar is a cursor indicating where the next character will appear on the screen.

Type in the letters "ABCDEF". The letters will be echoed on the screen, but nothing else will happen. Now type 'return'; the monitor will reply:

What is:A

The convention will be used that what is output by the monitor will be underlined in the examples in this manual.

The message 'What is:A' has been given because the first character of the line, A, was not recognised by the monitor as a valid command. This example illustrates a more important fact: the monitor only acts on a command when 'return' is typed. Before 'return' is typed the line is just held in memory, and it can be changed by typing 'rubout' to erase mistakes. In all the examples which follow it is assumed that every line typed in is terminated with 'return'; otherwise, nothing will happen.

2.1 Entering a Program

The 'M' (Modify) command is used to examine, and modify, the contents of memory. All numbers are entered, and displayed, in hexadecimal.
In the examples 'XX' indicates that any two hexadecimal digits might be displayed, depending on the previous contents of the memory.

Type in the following:

*M200

0200 XX 30,8D,00,04

0204 XX BD,F8,EC

0207 XX 39

```
0208 XX  08,8D,4F,4B,21,0D,0A
020F XX  08,8D,4F,4B,21,0D,0A
0216 XX  00 ;
*
```

This has entered the following simple program which calls a routine in the monitor, STRING, to output a string of characters to the display:

```
0200 308D 0004      PROG    LEAX  STR,PCR          GET STRING ADDRESS
0204 BD F8EC                JSR   STRING           MONITOR ROUTINE
0207 39                     RTS                    RETURN

0208 08             STR     FCB    $08,$8D,$4F,$4B,$21,$0D,$0A
020E 08                     FCB    $08,$8D,$4F,$4B,$21,$0D,$0A
0216 00                     FCB    $00             TERMINATOR
```

To execute the program use the 'G' (Go) command. Type: *G200
The program will display OK! in double-height flashing letters, and return to the monitor's prompt. The program can be re-executed by simply typing 'G' since the command remembers the last address used.

2.2 Cassette Calibration

The next step is to save the program on cassette. The playback level of the cassette is quite critical, and so it is first necessary to find the optimum playback level for the particular cassette recorder being used. First, enter the following program which will record a stream of 'X's on the tape:

```
*MO

0000 XX  86,58,BD,FD,25,20,FB ;

*
```

This corresponds to the following program:

```
0000 86 58          CALIB    LDA    £'X
0002 BD FD25        CALIB2   JSR    MCASOP   OUTPUT A TO CASSETTE
0005 20 FB                   BRA    CALIB2   LOOP FOR EVER
```

Execute the program by typing:

*GO

and record for a minute or so. Stop the program by pressing 'reset', then enter the following program which will read characters from the cassette and store them in the display area of memory:

*M10

0010 XX 8E,04,00,BD,FD,53,A7,80

0018 XX 8C,08,00,26,F6,20,F1 ;

This corresponds to the following program:

```
0010   BE    0400  READ   LDX    E$400       DISPLAY AREA START
0013   BD    FD53  READ2  JSR    MCASIN      INPUT A FROM CASSETTE
0016   A7    80          STA    ,X+
0018   8C    0800        CMPX   £$800       DISPLAY AREA END
0018   26    F6          BNE    READ2
001D   20    Fl          BRA    READ
```

Execute at 10 and adjust the playback level so that a stream of 'X's appear on the screen.

2.3 Store and Load

To save a program the 'S' command is used. Programs are identified on the tape by a filename consisting of up to 6 letters. Thus, several programs can be stored on one tape, and the load routine will search for the one with the required name. For example, calling the above program PROG, it can be saved with:

*S200 216 PROG

where the two numbers are the start and end addresses of the program. It can then be loaded with:

*L PROG

Load displays the last address of each block loaded. Alternatively, typing:

*L PROG G200

will load the program and execute it as soon as it is loaded.

2.4 User's Registers

The contents of the registers are saved in memory when the user's program returns to the monitor; these values are loaded into the registers when a program is run using the 'G' command. To look at the saved values of the registers, type 'R' (Registers). For example, with the above program loaded, typing:

*G200 R

will display:

```
CC    A  B DP     X     Y     U    PC     S
04   00 00 00  0217  0000  0000  FBEE  034D
PC-XX XX XX XX XX
```

The user's program can also be terminated by a SWI instruction (3F), in which case the registers will be displayed automatically on return to the monitor.

The memory area, where the user's register values are saved, can be accessed automatically by typing 'MR'. The first location displayed corresponds to the CC register, and the other registers follow in the order displayed in the 'R' command. By modifying the contents of these locations the initial contents of the registers can be specified before running a program.

The program counter (or PC) is also saved in the register area after a SWI instruction, and can be modified with the 'MR' command along with the other registers.

A program may be executed from the saved PC address by means of the 'P' (Proceed) command.

2.5 Breakpoints

The simplest way to debug a program is to examine the contents of the registers at various points during the program's execution.

This is achieved by inserting a breakpoint at the desired point is a SWI instruction which will cause a return to the monitor a display of the register contents. The monitor will insert and remove one breakpoint automatically; for example, to insert a breakpoint after the first instruction in the above example program:

*V204

Now, executing the program with:

*G200

will display:

```
CC   A  B DP     X     Y     U    PC     S
80  00 00 00  0208  D000  0000  0204  034D
PC+BD  F8 EC 39 08
```

Note that the X register contains the start address of the string, 0208, as required. The PC contains 0204, the breakpoint address.

The third line of the display shows the five memory bytes following the program counter; i.e. the next instruction of the program.

To continue execution of the program, type:

*P1

where the optional number after the Proceed command indicates the number of breakpoints to be ignored; 1 in this case.

Alternatively we could type:

* VP

The V command without any address cancels the breakpoint.

Any number of additional breakpoints can be inserted using the M command.

Suppose the breakpoint, inserted at 204 as described above, needs to be moved to 207. One way is to type:

* V207

Alternatively the 'MV' command can be used. Type:

```
* MV
0204 BD
0205 F8
0206 EC
0207 39  ;
```

The breakpoint address is moved to the last address displayed.

2.6 Trace Facility

An important debugging facility provided by the monitor is the ability to execute a program an instruction at a time, displaying the register contents after each instruction. To turn the trace facility on, type:

```
*T1
```

where the parameter, 1, indicates how many instructions are to be executed on each trace. Then, the '.' command will execute just one instruction, from the previous halt address, and return to the monitor after displaying the registers.

For example, with the example program loaded, set a breakpoint at the start of the program and enable tracing with:

```
*V200 T1
```

Then start execution at 200 with:

```
*G200
```

Successively typing '.' will step through the program as described.

2.7 Error Messages

Whenever the monitor reads a character, such as 'X', that it cannot understand, it prints:

What is:X

The rest of the line is ignored.

An unrecognized interrupt will give the message:

I-Err

For example, attempting to execute an SWI2 instruction (10 3F) without first redefining the interrupt vector ISWI2 will give this error. The only way to exit from this error is by 'reset'.

When the 'M' command is used to modify memory, a check is made that the stored value is correct. If a discrepancy is found, the message:

Rom?

will be given to warn the user.

The cassette-tape format includes a checksum byte at the end of each block of data. The load routine checks this, and if an error is found, the message:

XXXX -Err

is given, where XXXX is the last address of the block containing the error. The F (Finish) command can be used to finish loading a tape which contains errors; it will load without searching for a filename.

Finally, errors during the disk bootstrap command 'D' are of the form:

Err XX

where XX specifies the error number. Err FF means that the disk did not contain a valid boot file; other error codes are those generated by the disk controller.

3.0 SUMMARY OF MONITOR COMMANDS

Some commands are followed by optional parameters. These will be specified in quotation marks in the following list: e.g. 'name'. If any parameters are omitted then sensible defaults will be assumed.

All numbers to be input are in hexadecimal, and leading zeroes are ignored. Numbers may have leading spaces, and the number ends on the first non-hex character. A comma is treated as the last character of the previous number if no spaces intervene. A comma or carriage return with no digits will become a default value. Note then that "123," is a single number, but "123 ," is two numbers.

Commands may be strung together on one line, and no separators are required unless the line is ambiguous in which case comma or space should be used. Ambiguity can only arise with the commands C, D, and F, since these are also hex digits.

Commands

Modify Memory

M 'address' MODIFY

The contents of the given address will be printed. Numbers entered will be stored at successive memory locations starting at that address. The stored result is checked, and if different the user will be queried.

A comma will move up one location, and a minus will move down one location. A carriage return with no data will also move up one location. The M command is exited with a semicolon, and the last address accessed will be saved as the default address.

MG

Modify memory starting at the saved Go address. The Go address will be changed to the address last accessed when the modify command is exited.

MR

Modify registers. The first location opened is the CC register. The registers follow in the sequence:

CC, A, B, DP, X (high), X (low), Y (high), Y (low),

U (high), U (low), PC (high), PC (low).

MV

Modify memory starting at breakpoint address. The breakpoint address is moved to the address last accessed when the modify command is exited.

MP

Modify memory starting at saved program counter address. The next P command will cause execution of the user program to resume at the address last accessed when the modify command is exited.

Program Execution

G 'address' GO

Go to address specified. Registers are loaded from user register area. Will return to the monitor on an RTS or SWI instruction.

P 'number' PROCEED Proceed after hitting a breakpoint. Execution begins at the saved program counter address. If a number is entered after the P, this number of breakpoints will be ignored by the monitor. On hitting a breakpoint the monitor is entered with an automatic R command.

Program Debugging

R REGISTERS

Registers; prints contents of user registers, and contents of the five memory locations pointed to by the saved program counter.

V 'address' BREAKPOINT

Breakpoint insert/remove. If an address is specified, a breakpoint will be inserted at this address. Any previous breakpoint is removed. The breakpoint will not be inserted until the program is executed using G or P.

If no address is specified the breakpoint is cancelled.

T 'number' TRACE

Trace facility. The command is followed by a number indicating the number of instructions to be executed on each "." command. If no number is given the trace will be turned off, and "." will be ignored. The monitor starts with trace switched off.

DO TRACE

Do Trace. The number of instructions set by the T command will be executed, and the monitor will return with an automatic R command to display the registers.

Cassette Interface

S 'start address' 'end address' 'file name' SAVE

Saves an area of memory between the specified addresses. The file name may have up to 6 characters not including space, comma, or carriage return. The name is padded to 6 characters, if needed, with spaces on the right. The file is saved as a name header block followed by data blocks of up to 256 bytes, and terminated by a terminator block.

A minus sign following the filename will inhibit output of a terminator block, thus allowing non-contiguous blocks of data to be saved as a single file.

All three parameters of the S command will default to the last values used.

L 'file name' 'offset' LOAD

Load a file into memory from cassette. All input data will be ignored until the required header block is found.

The character "?" in the file name specified will match any character in the input file name; e.g. "DATA??" will match "DATA01", " DATA02", etc. The name "??????" will match any file name.

The optional offset will be added to the start and end addresses from the input file, thus enabling a file to be relocated to an address in memory different from where it was saved from. An offset can also be specified to load a file back into an unused part of the address space to verify that it was saved correctly without destroying the original version in memory.

The Load will print out the last address of each data block loaded. Each data block includes a checksum; if an error is found the message " -Err° will be printed after the last address of the block causing the error.

During loading the keyboard interrupt is disabled to prevent errors caused by typing at the keyboard.

F

FINISH

Finish loading without searching for a name header block.

Can be used after an error to load the remainder of the file.

Printer Interface

C '+' COPY

The printer is switched on to echo all data shown on the VDU. If the next character is not a "+" then the printer is switched off.

Disc Interface

D DISK

Loads a bootstrap program from a minifloppy disk drive, thus entering the disk operating system.

4.0 MONITOR EXPANSION

This section describes how to take advantage of the expandability of the monitor.

4.1 Extra Rom

The monitor occupies address space between F800 and FFFF. A second ROM may be added in the space F000 to F7FF. The monitor checks for the presence of the extra ROM, and calls it as a subroutine if it is present. The extra ROM should contain:

```
F7FC SUBR
F7FE $A55A
```

where SUBR is the address of the subroutine called by the monitor. This subroutine can be used to reassign the command-table addresses and thus add commands to the existing monitor.

4.2 Adding to Monitor Commands

The following example shows how a user-written command can be linked in to the monitor so that it behaves as if it were another monitor command. The command described is one to list an area of memory on the screen, eight bytes per line, both in hexadecimal and in ASCII. Thus a typical line would appear:

```
0200 01 44 00 07 FF FD 52 BD -D----R-
```

showing the eight bytes from 200 to 207. The program lists 24 lines at a time, since these fit conveniently on the screen. The command is assigned the name 'D' (Dump), and is followed by the start address of the memory area to be dumped. The command calls five routines which are provided in the monitor; these handle the input and output of numbers and characters.

```
                    * DUMP COMMAND
                    *
                    * LINKED TO MONITOR COMMAND SET
                    *
0200  01             AUXTAB  FCB     1               NO. OF COMMANDS IN TABLE
0201  44                     FCB     'D              COMMAND NAME
0202  0007                   FDB     DUMP-AUXTAB     OFFSET TO ROUTINE
0204  FF                     FCB                     GO TO NEXT TABLE
0205  FF51                   FDB     CMNDS           MONITOR COMMAND TABLE
                    *
0207  BD   FB6C     DUMP    JSR     NUMB          GET HEX NO. IN D
020A  1F   02               TFR     D,Y           PUT IN Y
020C  86   18               LDA     £24           NO. OF LINES
020E  3402         PSHS     A       SAVE COUNTER
0210  C608 GO      LDB      £8      BYTES PER LINE
0212  34   24               PSHS    Y,B
0214  1F   21               TFR.    Y,X           X = ADDRESS
0216  BD   FA75             JSR     OPXREG        PRINT X IN HEX + SPACE
0219  A6   AO       D1      LDA     ,Y+           GET BYTE
0218  BD   FA97             JSR     OPARSP        PRINT A IN HEX + SPACE
021E  5A                   DECB
021F  26   F8               BNE     D1
0221  35   24               PULS    Y,B           RESTORE
0223  A6   AO       D2      LDA     ,Y+           GET BYTE AGAIN
0225  81   20               CMPA    £$20          ASCII PRINTABLE ?
0227  25   04               BCS     D3
0229  81   7E               CMPA    £$7E          TOO BIG ?
022B  23   02               BLS     D4
022D  86   FF       D3      LDA     £$FF          PRINT WHITE BLOCK
022F  BD   FA21     D4      JSR     CONOUT        CONSOLE OUTPUT A
0232  5A                   DECB
0233  26   EE       BNE     D1
0235  BD   F8E9             JSR     OPCRLF        START NEW LINE
0238  6A   E4               DEC     ,S            DECREMENT LINE COUNTER
023A  26   D4               BNE     GO
023C  35   82               PULS    A,PC          PULL COUNTER + RETURN.
                    *
                    * ADDRESSES IN MONITOR
                    *
           FF51     CMNDS   EQU     $FF51         COMMAND TABLE
           FB6C     NUMB    EQU     $FB6C         INPUT HEX TO D
           FA75     OPXREG  EQU     $FA75         OUTPUT HEX FROM X
           FA97     OPARSP  EQU     $FA97         OUTPUT HEX FROM A
           FA21     CONOUT  EQU     $FA21         PRINT A
           F8E9     OPCRLF  EQU     $F8E9         NEW LINE *
                           END
```

When the dump program has been entered from the listing in Fig. 4.1 it can be executed by typing:

*G207

Any address typed after the Go command will be read by the call to subroutine NUMB and used as the start address of the area to be dumped. For example, to display the contents of memory starting from $F800, execute with:

*G207 F800

To make the monitor recognize D as the DUMP command the auxiliary command table, from 0200 to 0206, is needed. The address of this table, 0200, should be put at 0371 to replace the address of the monitor's command table, FF51. The monitor will then search the auxiliary table first, and the D command will be redefined to cause a jump to 0207.

The dump command can be saved on cassette so that, when loaded, it will automatically install itself as part of the monitor's command set. To do this, save it as follows:

*M371 02 00;

*S200 23D DUMP - S371 372

The first save command saves the program in a file named DUMP. The minus sign inhibits the save command from writing an end-of-file marker on the cassette, and the second save command writes a block of two bytes to redefine the command-table address. If the program is now loaded with:

*L DUMP

both blocks will be loaded, and D will have its new meaning, DUMP.

4.3 Interrupt Vectoring

All interrupts are vectored, using indirect jumps, via addresses held in RAM. The RAM locations are assigned as follows:

0379	ISWI3	SWI3 vector
037B	ISWI2	SWI2 vector
037D	IFIRQ	FIRQ vector
037F	IIRQ	IRQ vector
0381	ISWI	SWI vector
0383	INMI	NMI vector

These addresses can be altered to point to user interrupt service routines, so that all the interrupts are potentially available for user applications. Interrupts IRQ and SWI are used by the monitor; the other interrupts are vectored to a routine which will display an error message I-Err if an unexpected interrupt occurs.

The indirect jumps add an overhead of 8 cycles to the servicing of each interrupt.

4.3.1 Example

The following example illustrates the use of FIRQ in a user program. Each time the interrupt occurs an interrupt service routine ISR is called; this writes a hex number to the display, and increments it. The routine SETUP should be executed first to replace the address of the FIRQ indirect vector with the address of the interrupt service routine.

```
                * USE OF FIRQ
                * *
                * INTERRUPT SERVICE ROUTINE
                * *
0100 34    10   ISR    PSHS    X           FIRQ DOESN'T STACK REGS
0102 BE    0109        LDX     COUNTER
0105 30    01          LEAX    1,X         ADD 1
0107 BD    FA75        JSR     OPXREG      PRINT HEX; SAVES REGS
010A BF    0109        STX     COUNTER     REPLACE IT
010D 35    10          PULS    X           RESTORE X
010F 3B                RTI
               *
               * SET-UP ROUTINE
               *
0110 8E    0100 SETUP  LDX     ISR
0113 BF    037D        STX     IFIRQ
0116 1C    BF          ANDCC   £$BF        ENABLE FIRQ
0118 39                RTS
               *
0119 0000  COUNT       FDB     0           COUNT FROM ZERO
```

4.4 Driver Routines

The monitor calls the input/output driver routines indirectly through addresses in RAM, and these addresses may be changed by the user to enable the monitor to drive other devices. The addresses are assigned as follows:

```
0365        COPADR        Console output
0367        CINADR        Console input (from buffer)
0369        CASOPA        Cassette output
036B        CASINA        Cassette input
036D        PRINTI        Printer output
```

By default, these locations contain the following addresses:

```
0375 F95B                 FDB     DISPLA
0377 FA5E                 FDB     GETCHR
0379 FD25                 FDB     MCASOP
037B FD53                 FDB     MCASIN
037D FA9D                 FDB     PRINT
```

4.5 Command Passing

A command line can be passed as a text string to the monitor by a user program. The address of the command line should be stored in X, and the line should be terminated by a null byte. Multiple input lines are allowed, each line terminated with a carriage return. A typical calling sequence would be:

```
                        LDX     £COMMD      COMMAND LINE ADDRESS
                        JSR     MEMUSE
                        TSTA
                        BEQ     OK
                ERROR .......
```

The routine MEMUSE, at F871, interprets the command line and exits with A zero if all is well. A has the value FF if a null was found before it was expected, and the value of any character causing an error.

4.5.1 Demonstration Program to Illustrate Command Passing

The following program DEMO shows how the command line "L DUMP" can be passed to the monitor as described, causing the file DUMP to be loaded from cassette:

```
                    * DEMONSTRATION OF COMMAND PASSING
                    *
0010 8E   001B      DEMO    LDX     £STRING     POINT TO COMMAND LINE(S)
0013 BD   F871              JSR     MEMUSE
0016 4D                     TSTA
0017 27   01                BEQ     OX
0019 3F             ERROR   SWI                 ERROR RETURN
001A 39             OK      RTS
                    *
001B 4C204445       STRING  FCC     /L DU/
001F 4D50                   FCC     /MP/
0021 OD00                   FCB     $0D,0       TERMINATOR
```

4.6 Use of a Serial Terminal with 6809 Card

The Acorn 6809 card is primarily designed for use with an Acorn VDU card and a standard parallel keyboard, to form a complete 6809 development system. The serial interface, normally used to provide data and program storage on cassette, can also be used to interface the card to a serial terminal such as a teletype. The terminal may be used as a secondary output device in addition to the VDU, as the main output device, or for input and output:

4.6.1 Terminal as secondary output device (e.g. for hard copy). Store the cassette-output routine address CASOUT (FD25) at the printer indirect address location PRINTI (036D), and set PFLAG (0361) non-zero. This is achieved by typing:

```
M36D FD 25;M361 1;
```

4.6.2 Terminal as only output device, instead of VDU card.

Put CASOUT address (FD25) at console output address location COPADR (0365) by typing:

```
M365 FD 25;
```

4.6.3 Use of serial terminal for input and output.

In addition to the steps described in section 4.6.2 the monitor's input routine should be replaced by a user-written routine to get characters from the terminal, through the serial interface, and store them in the circular keyboard buffer.

The monitor automatically links to a ROM located between $F000 and $F7FF, if present, and the replacement input routine can conveniently be contained there.

4.6.4 Selection of baud rate.

The default baud rate for the serial interface is 300, and if other baud rates are required the contents of DELCNT (0363) should be changed as follows:

110 baud	$0234
300 baud	$00CD
1200 baud	$0030

5.0 HARDWARE DESCRIPTION

5.1 Memory Organization

The 6809 card uses part of the first 4K of address space, block zero, and part of the last 4K of address space, block F. The complete memory map is shown in Fig. 5.1. The map also shows the addresses assigned to memory and devices not on the 6809 card, but recognised by the monitor program. The 6909 card includes 1K of RAM from 0000 to 03FF, and this is contiguous with the 1K of RAM on the VDU card which occupies from 0400 to 07FF. The monitor uses locations 0359 to 03FF for the storage of variables, the user registers, and for the line input buffer. The monitor uses memory below this address for the hardware stack. The stack depth will not normally go below 0300, so the memory from 0000 to 02FF is free for use by user programs.

5.2 Memory Decoding

Memory decoding is performed by a 256 x 8 bipolar PROM, IC11, which divides the 64K of the 6809's address space into 256 256-byte pages. Any of the devices on the Acorn 6809 card may be mapped into any of these pages by providing a suitably programmed ROM. As provided the following signals are produced by the address PROM:

Signal	Address space
RAM	0 - 3FF
VIA	900 - 9FF
ROMO	F800 - FFFF
ROM1	F000 - F7FF
BLOCKO	0 - FFF
ONCARD	0 - 3FF, 900 - 9FF, F800 - FFFF.

All the signals are active low. The ONCARD signal is low whenever any of the on-card devices are addressed, and this signal controls the data-bus buffers. The BLOCK0 signal is low for addresses in the bottom 4K of memory, and is used to enable the VDU card.

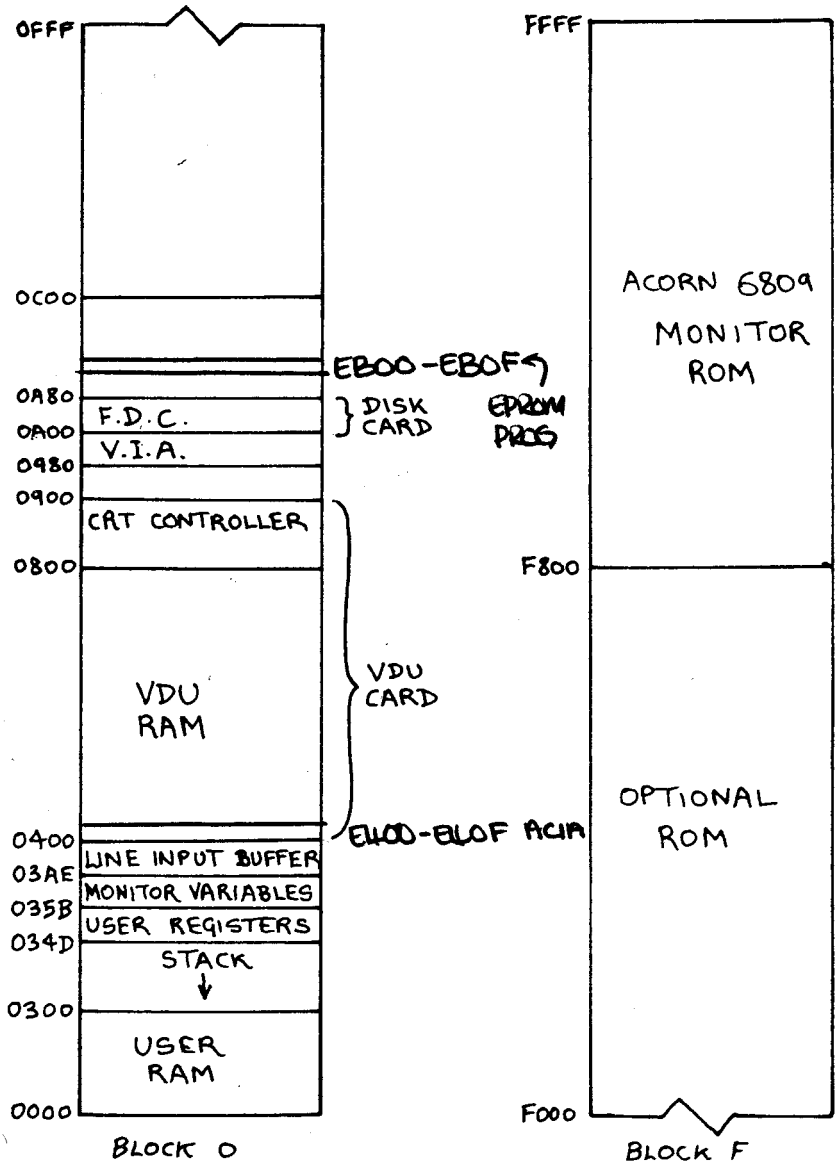


Fig. 5.1. 6809 Card Memory Map.

5.3 Extra ROM

The standard 2K monitor is contained in a 2716-type EPROM, occupying memory between F800 and FFFF. The Acorn 6809 card can be modified to accomodate a 2732-type EPROM in its place, addressed between F000 and FFFF, and the present monitor can be copied into the upper 2K of the 4K ROM to provide space for extended facilities on the same card. Details of how the extra ROM can be linked in with the present monitor are given in section 4.1.

The following modifications are necessary to use a 2732 in the place of IC4:

1. Break track from +5v to pin 21 of IC4.
2. Link pin 21 of IC4 to Bus All.
3. Link pins 9 and 11 of IC11. This will cause the ROM to be enabled for addresses F000 to F7FF in addition to F800 to.FFFF.
4. Link pins 13 and 14 of IC11. This will include the address space F000 to F7FF in the ONCARD signal.

5.4 Bus Buffering

The data bus is buffered by an octal tr^{i}-state bidirectional buffer, type 8208. The data-bus buffers are disabled when the E signal is low, and when the processor is addressing memory space on the 6809 card. They will also act correctly when a DMA device accesses memory, whether on or off the 6809 card.

The R/W and address lines are buffered by 74LS244 devices, and are tr^{i}-stated to allow other devices to do DMA. The R/W line is ANDed with the E signal to give separate NRDS and NWDS signals at the edge connector; these are provided so that devices from the 8080 family can easily be interfaced to the 6809 card, and the signals are used by some of the other cards in the Acorn range. The NWDS bus line is generated from the unbuffered E signal for improved timing.

The signals Q, E, BA, and BS are buffered but not tri-stated.

5.4.1 Reset - The reset line is not buffered. The 6809 will come out of reset when RESET is at a level above about 4v. Other devices are designed to reset at a lower voltage, typically 0.4v, so that on power-up they will be fully reset before the processor begins execution.

Power-up reset may be provided by connecting a capacitor of about 200 uF between RESET and Ov; a space on the board is provided for this.

5.4.2 VMA - A bus $\overline{VMA}$ line is provided which is low when the bus buffers are enabled. The $\overline{VMA}$ line is effectively open-collector so that any number of devices may be attached to the system using this line, and daisy-chaining the bus available (BA) and bus request lines.

5.4.3 DMA - When a DMA device requests bus control by taking HALT or BREQ low, the processor will eventually take BA (bus available) high. The cycle during which this occurs should not be used for a bus transfer; i.e. $\overline{VMA}$ should be high. The DMA device should ensure this.

When the DMA device relinquishes control of the bus the processor will set BA low. Again there is a null cycle while control is transferred. On the 6809 card IC12 is used to provide a delayed BA signal which will not go low until the negative transition of E following BA going low. The circuit is shown in Fig. 5.2. This ensures that $\overline{VMA}$ remains high during the null cycle, and that all irrelevant bus buffers will be turned off.

5.5 System Clock

The system clock is derived from a 4 MHz crystal, giving a 1 usec. instruction cycle time. The Q and E signals at the edge connector are therefore 1 MHz.

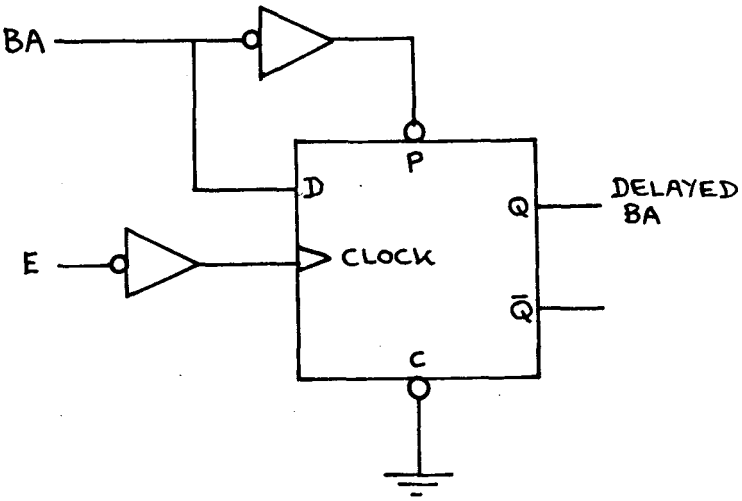


Fig. 5.2. Generation of delayed BA signal.

5.65.6 Audio Cassette Interface

Programs and data say be saved on cassette and loaded from cassette using the interface programs included in the 6809 monitor.
The 6809 card provides logic-level input and output lines, and the cassette interface routines transfer data onto these lines in standard serial format. The card does not include circuitry for the generation of tones to encode the serial data in a form suitable for storage on an audio tape, but the full circuit is given in Fig. 5.3. A kit of parts for this circuit, with a circuit board, is available from Acorn.

For flexibility the bit rate is determined by a software timing loop in the cassette input and output routines; the value of the delay counter can be altered to give different baud rates. The default rate, initialised on reset, is 300 baud but rates of up to 1200 baud are possible.

The serial input and output routines can also be used to provide interfacing with serial devices such as a terminal or a teletype. Output can be vectored to the serial output routine simply by changing a vector address stored in RAM. To receive input from a serial device is also possible, but requires the provision of specially-written routines to put the received data into the keyboard buffer.

5.6.1 Cassette Input

The software asynchronous receiver gets 1 start bit, 8 data bits, and 1 stop bit. Data is shifted in from PB7 of the VIA, lowest bit first. The data input is inverted.

Since keyboard interrupts could upset the cassette timing the keyboard interrupt is disabled during cassette load.

5.6.2 Cassette Output

The software asynchronous transmitter outputs the byte from the A register as a start bit, 8 data bits, and 2 stop bits. Data is

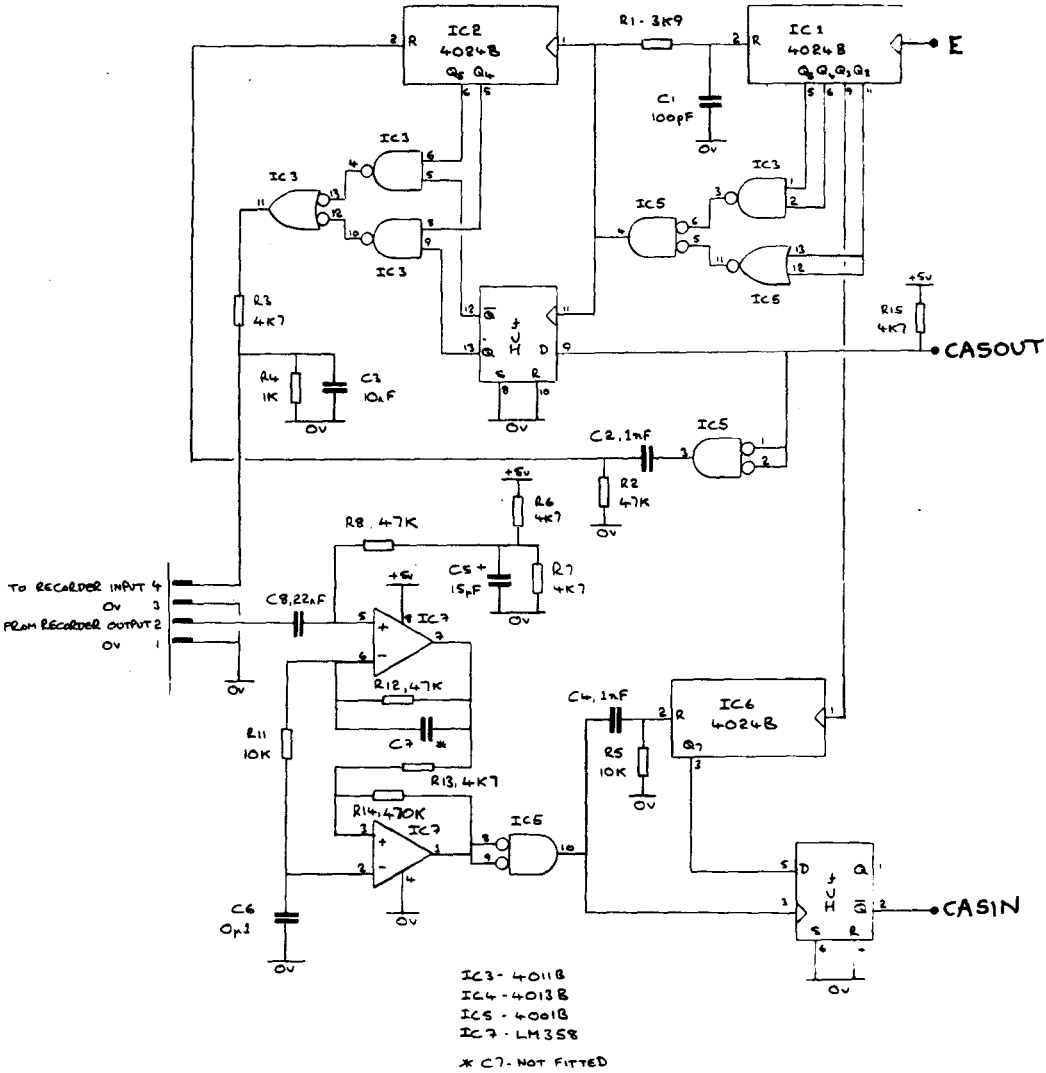


Fig. 5.3. Suggested circuit for a cassette interface. The three signals on the right side of the circuit connect to the 6809 card.

output from CB2 of the VIA, lowest bit first.

The IRQ and FIRQ masks are set during the output of a byte to the cassette output so that the timing will not be upset by servicing interrupts. Interrupts are permitted in between bytes.

5.6.3 Cassette Format

Three types of block are output by the cassette file save routine:

1. File Header Block

Format: $D8, $30, X, X, X, X, X, X, CK.

where X is any ASCII character, excluding space, comma, or carriage return, and CK is the checksum byte.

2. Data Block

Format: $D8, $31, SH, SL, EH, EL, DO, D1, D2, ... Dn, CK. where SH and SL are the high and low-bytes of the start address respectively, EH and EL are the high and low-bytes of the end address, DO to Dn are the data bytes in binary, and CK is the checksum byte. The number of data bytes is from 1 to 256.

3. Terminator Block

Format: $D8, $39.

The sum over all bytes after the header pair of bytes, and including the checksum, is $FF in each block.

5.7 Keyboard Interface

The monitor is designed to receive commands from a parallel keyboard connected to the inputs PBO - PB6, with a strobe line to CB1 on the VIA. The keyboard should have a negative-going strobe signal and non-inverted data outputs. The edge connections on the 6809 card include connections to the power rails, and a 5v keyboard may be powered from these. The break key on the keyboard may be wired to the reset line

which is also available at the keyboard connector. The break signal should provide a negative-going signal when pressed.

The keyboard is interrupt-driven, making its operation totally independent of external programs; data may be typed at the keyboard while the processor is executing a program, and it will be echoed on the VDU and buffered in the line buffer. Programs may be written to read characters from the line buffer at any time. If there are already characters in the line buffer the read-character routine will immediately return with the character; otherwise it will wait for characters to be entered at the keyboard.

A negative-going strobe signal on CB1 of the VIA will generate an interrupt, if the IRQ mask is clear, and will set bit 4 in the VIA Interrupt Flag Register. The IRQ service routine tests this bit to determine whether the interrupt was due to a keyboard interrupt or an interrupt from the VIA timer 1, used in trace mode.

5.8 Trace

A hardware trace function is provided on the 6809 card to enable programs to be executed one, or more, instructions at a time. The function is controlled by one of the two timers in the VIA, timer 1. Timer 1 is addressed as follows:

Register:	Address:	Function:
4	0984	Counter low
5	0985	Counter high
6	0986	Latch low
7	0987	Latch high.

The following sequence is executed to jump to a user program in trace mode:

1. Initialize counter to $000F (15)

2. Pull all registers from the hardware stack.

The timer 1 counter will take the IRQ line low after 16.5 E cycles; i.e. on the rising edge of E following 16 instruction cycles. By the next falling edge of E, after 17 cycles, the IRQ will be latched by the processor. A delay of at least one bus cycle will then occur before the interrupt is serviced. The instruction to pull all the registers from the stack takes 17 cycles, and so the interrupt will not be serviced until after the next instruction of the user's program has been executed.

5.9 Printer Interface

A parallel handshake interface to a printer is provided on the Acorn 6809 card, and printer driving routines are included in the monitor. The handshaking signals comprise a BUSY line from the printer, which is connected to PA7 of the VIA, and a strobe signal from the VIA pin CA2 to the printer.

When the BUSY signal goes low, the printer routine will put a byte on the printer lines PAO - PA6, and take the strobe low for 7 usecs.

Although the $\overline{\text{ACK}}$ line from the printer is connected to CAl of the VIA, the existing software does not make use of this signal.

The outputs to the printer, PAO - PA6 and CA2, are buffered by an octal buffer device.

6.0 SOFTWARE DESCRIPTION

This section describes the operation of the most useful subroutines contained in the monitor; these can all be incorporated into user programs without a full understanding of the monitor being needed.

6.1 Input

When a key is typed at the keyboard an IRQ interrupt occurs, if it has not been masked, and the key's value is stored in the next location of the 80-character circular buffer. The console input routine, CONIN, is normally indirected via a RAM vector to the subroutine GETCHR. This reads a character from the circular buffer, or waits for one to be entered if the buffer is empty. CONIN can also be directed to return characters read from an area of memory.

An alternative character-input routine is used by most of the monitor routines: this is CONCHR, which will only return characters read up to a carriage-return. Thereafter it will return carriage-returns until the flag ONLINE is cleared. This ensures that monitor commands, such as S, that expect several parameters will not cause reading past the end of the line when parameters have been ommitted.

6.1.1 Routines

Name:	Address:	
CONIN	F890	Console input routine. Gets character from input routine via RAM vector CINADR if LINEPT=0, or from memory at address LINEPT. If it finds a null in memory it returns to caller with error $FF.
CONCHR	F87F	Alternative console input routine; reads up to CR calling CONIN, and then returns CRs. Make ONLINE non-zero to clear.

Name: Address:

GETCHR FA5C Default character-input routine. Gets character from buffer. If none then clears interrupt mask and waits. All registers saved, including CC.

The following routines call CONCHR to input single characters:

GETHEX FB95 Get hex digit in A, with V=0, else V=1 if non-hex.

GETHXS FB8A As above, but ignore leading spaces.

NUMB FB6C Get hex number, with any number of digits, from input stream. Allow leading spaces, and stop on first non-hex character. Number returned in D, with V=0. If no number then D=0 and V=1.

NAMEIN FD75 Get name from input stream, up to 6 characters long. Name stored at NAME (039D). No name leaves memory unaltered; any name is padded with spaces to 6 characters.

6.2 Output

The console output routine, CONOUT, is normally indirected via a RAM vector to subroutine DISPLA. This first checks the character for carriage-return, linefeed, formfeed, or delete; if none of these, the character is written to the next screen location, and the cursor is moved on one position. Attempting to move the cursor below the bottom line will cause the screen to be scrolled by reprogramming the 6845 CRT controller on the VDU card for a different display start address. Before the screen has been scrolled the address corresponding to the leftmost character in the top line is $0400. After scrolling the memory-to-screen mapping becomes more complicated , and the routine CCLOCN should be used to calculate the cursor location.

The four special characters have the following actions:

carriage-return: cursor to start of line.

linefeed: cursor to next line.

formfeed: display RAM cleared; screen format reset with cursor off screen.

delete: backspace cursor; blank character under cursor.

When scrolling takes place the bottom line of the screen is cleared.

6.2.1 Routines

Name:	Address:	
CONOUT	FA21	Console output routine. Sends character in A via RAM vector COPADR, and to printer via vector PRINTI if PFLAG is non-zero.
DISPLA	F95B	Default character-output routine called by CONOUT. Puts character in A to VDU, handling CR, LF, FF, and delete. All registers saved.
PRINT	FA9D	Default printer routine, called by CONOUT is PFLAG is non-zero. This interfaces to Anadex or Centronics parallel interface printers.

The following routines all call CONOUT to output characters:

STRING	F8EC	Output string pointed to by X, terminated by a null. Leaves X pointing to null+1; other registers saved.
OPCRLF	F8E9	Outputs CR, LF to console. Destroys X.
HEXOUT	FA8D	Output A as a single hex digit.
OPARSP	FA97	Output A as a single hex digit followed by a space.
OPAREG	FA81	Output A as two hex digits. All registers except A are saved.
OPXREG	FA75	Output X register as 4 hex digits. All registers are saved.

6.3 Tape Routines

Name:	Address:	
MCASIN	FD53	Software asynchronous receiver. Gets value into A with 1 stop bit. Saves all other registers.
MCASOP	FD25	Software asynchronous transmitter. Outputs value in A as a start bit, 8 data bits, and two stop bits. Rate controlled by DELCNT. Saves all registers,
CBIN1	FD1E	Get one byte from tape, and update checksum.
CBIN2	FD13	Gets 2 bytes and forms a 16-bit value in D

6.4 Disk Routines

Name:	Address:	
BOOT	FE44	Bootstrap from mini-floppy disk.
TRNSFR	FEC9	Transfers data from disk to memory, starting at address in U. Returns completion code in A when transfer finished or error occurs.
DRVRDY	FEB6	Test if drive ready. On entry X points to read drive status command sequence; on exit drive is ready and X points to next command sequence.
CMDPAR	FE9F	Send one command followed by a variable number of parameters. X points to command; next byte is number of parameters, possibly none. X left pointing to last parameter. Destroys D.

6.5 Miscellaneous

Name:	Address:	
DISPCH	F9A6	Dispach routine. Looks up character in A in a table at X. Table format is: First byte: number of entries, 1 to 255. For each entry: Character to match 2-byte offset to routine for match Flag byte: determines action if no match found $01 - next word is offset to default routine $00 - return to calling program $FF - next word is address of another table
CCLOCN	F905	Calculate real address of cursor in memory space. Result returned in X.
RESET	F800	Reset entry point.
MEMUSE	F871	Pass command to monitor. X is start of line which is terminated with a null. Multiple input lines are allowed, separated by CR. Exits with A zero if no error.

7.0 INSTRUCTION SET AND ADDRESSING MODES

7.1 Programming Model

A programming model of the 6809 is shown in Fig. 7.1. There are four 16-bit pointer registers, the program counter, two 8-bit accumulators which can be used as one 16-bit register, and two special purpose 8-bit registers.

7.1.1 Accumulators (A, B, D)

The A and B registers are general purpose accumulators which are used for arithmetic and logical operations. Most instructions will operate in an identical way with either accumulator. Certain instructions are provided which will operate on the A and B registers considered as one 16-bit register, referred to as the D register. The A register is the most significant byte of the D register.

7.1.2 Direct Page Register (DP)

The direct page register defines which page of memory is to be accessed by direct addressing; see section 7.3.5. When peripherals are being accessed the direct page register can be set to the peripheral's page, thus speeding up access.

7.1.3 Index Registers (X, Y)

The index registers are used in the indexed mode of addressing; the 16-bit address in the specified register takes part in the calculation of the effective address. This address may be used to point to data directly or may be modified by an optional constant or register offset. During some indexed modes the contents of the index register are incremented, or decremented, as a result of the operation. All
four pointer registers, X, Y, U, and S, may be used as index registers.

7.1.4 Stack Pointers (U, S)

The hardware stack pointer, S, is used automatically by the processor

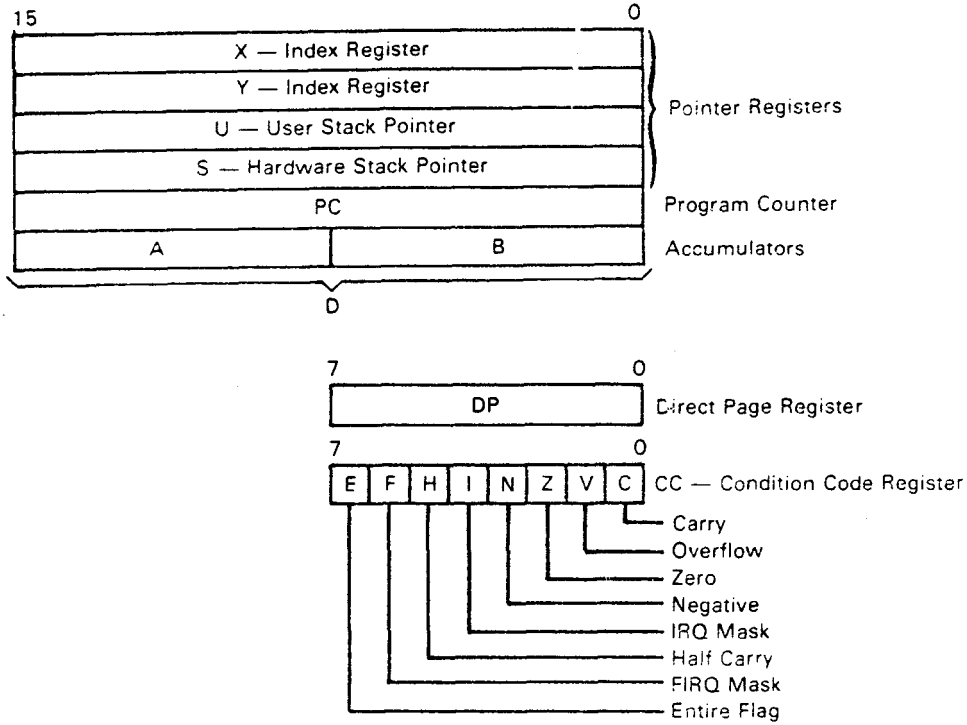


Fig. 7.1. Programming model of the 6809.

during subroutine calls and interrupts. The stack pointers point to the top of the stack. The user stack pointer, U, is controlled exclusively by the programmer thus allowing arguments to be passed to and from subroutines with ease. Both stack pointers have the same indexed mode addressing capabilities as the X and Y registers, but also support push and pull instructions.

7.1.5 Program Counter (PC)

The program counter is used by the processor to point to the next instruction to be executed. Relative addressing is provided, allowing the PC to be used like an index register in some situations.

7.1.6 Condition Code Register (CC)

The condition code register defines the state of the processor at any time:

Bit 0 - Carry Flag

For add operations the carry is set if and only if the addition causes a carry from the most significant bit. For subtract-like operations (SUB, SBC, CMP) the carry is set if and only if the operation does not cause a carry from the most significant bit.

Shifts and rotates affect the carry according to the data being shifted.

The MUL multiply instruction sets the carry if and only if bit 7 of the result is set.

Bit 1 - Overflow Flag

Set if and only if the operation causes a signed two's-complement overflow. Thus N + V will give the correct sign even if the sign is not correctly represented in the result.

Bit 2 - Zero Flag

Set if and only if the result of the operation was equal to zero.

Bit 3 - Negative Flag

Contains the value of the most significant bit of the result of the operation; thus a two's-complement result will set N if the result was negative.

Bit 4 - IRQ Mask Bit

The processor will not recognise interrupts from the IRQ line if this bit is set to a one. NMI, FIRQ, IRQ, RESET, and SWI all set I to one; SWI2 and SWI3 do not affect I.

Bit 5 - Half-Carry Bit

This bit indicates a carry from bit 3 as a result of an 8-bit addition (ADC or ADD). The bit is used by the decimal adjust (DAA) instruction to perform a BCD adjust operation. The state of this flag is undefined in all subtract-like operations.

Bit 6 - FIRQ Mask Bit

The processor will not recognise interrupts from the FIRQ line if this bit is set to a one. NMI, FIRQ, RESET, and SWI all set F to a one. IRQ, SWI2, and SWI3 do not affect F.

Bit 7 - Entire Flag

Set to a one if all the registers were stacked, as opposed to the PC and CC only, after an interrupt. The E flag of the stacked CC is used on return from an interrupt (RTI) to determine the extent of the unstacking.

7.2 Instruction Set

A complete list of the 6809 instruction set is given in Figs. 7.2 to 7.6. Some of the more unusual instructions are explained in the following sections.

7.2.1 Load Effective Address

The load effective address instruction, LEA, allows all the 6809's addressing modes to be used to form an address, and this address is then loaded into one of the four pointer registers. Some uses of this instruction are described:

```
LEAX 3,X
```

adds the constant, 3, to the X regsiter.

```
LEAU D,U
```

adds the signed number in the D register to the number in the U register and stores the result in U. Thus:

```
LEAX B,X
```

is similar to the ABX instruction, but whereas LEAX B,X treats B as a two's-complement signed number between -128 and +127, ABX uses B as a positive offset between 0 and 255.

```
LEAS -10,S
```

can be used to reserve 10 bytes of workspace on the hardware stack. This workspace can be addressed indexed using S; for example:

```
LDA    4,S    or    DEC    2,S.
```

The workspace is restored after use with:

```
LEAS 10,S
```

The destination and source registers can be different, as in:

```
LEAX 4,S
```

which loads the X register with a pointer to the 4th byte on the hardware stack.

```
LEAX DATA,PCR
```

loads the address of DATA into the X register; the code for the

Mnemonic(s)	Operation
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A-accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A x B — D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR, R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE: A, B, CC, or DP may be pushed to (pulled from) either stack with PSHS, PSHU, (PULS, PULU) instructions

Fig. 7.2. 8-bit accumulator and memory instructions.

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U or PC
TFR R, D	Transfer X, Y, S, U or PC to D

Fig. 7.3. 16-bit accumulator and memory instructions.

Mnemonic(s)	Operation
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U, or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push any register(s) onto hardware stack (except S)
PSHU	Push any register(s) onto user stack (except U)
PULS	Pull any register(s) from hardware stack (except S)
PULU	Pull any register(s) from hardware stack (except U)
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U or PC
ABX	Add B accumulator to X (unsigned)

Fig. 7.4. Index register/stack pointer instructions.

Mnemonic(s)	Operation
BCC, LBCC	Branch if carry clear
BCS, LBCS	Branch if carry set
BEQ, LBEQ	Branch if equal
BGE, LBGE	Branch if greater than or equal (signed)
BGT, LBGT	Branch if greater (signed)
BHI, LBHI	Branch if higher (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BLE, LBLE	Branch if less than or equal (signed)
BLO, LBLO	Branch if lower (unsigned)
BLS, LBLS	Branch if lower or same (unsigned)
BLT, LBLT	Branch if less than (signed)
BMI, LBMI	Branch if minus
BNE, LBNE	Branch if not equal
BPL, LBPL	Branch if plus
BRA, LBRA	Branch always
BRN, LBRN	Branch never
BSR, LBSR	Branch to subroutine
BVC, LBVC	Branch if overflow clear
BVS, LBVS	Branch if overflow set

Fig. 7.5. Branch instructions.

Mnemonic(s)	Operation
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

Fig. 7.6. Miscellaneous instructions.

instruction is position-independent since it contains the offset of the data from the instruction, not the absolute address of the data. Note that:

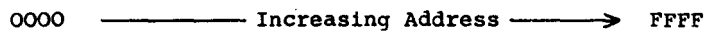
```
LEAX (,X)
```

has the same effect as the instruction:

```
LDX ,X
```

7.2.2 Push/Pull

The push and pull instructions enable any combination of registers to be saved and restored using either stack. The instruction is two bytes long regardless of the number of registers pushed or pulled; each bit in the second byte of the instruction corresponds to a register, and if the bit is set that register is pushed/pulled. The order of stacking or restoring is a hardware function, irrespective of the order specified in the assembler statement, and is shown below:



CC A B DP X Y U/S PC

Thus the CC is pushed last and pulled first (if specified).

Since the PC is pulled last, the sequence:

```
PULS A,B,X
RTS
```

may be shortened to the identical sequence:

```
PULS A,B,X,PC
```

Note that:

```
PSHS A      and     STA ,-S
```

are similar in effect, but PSHS does not affect the status flags.
Similarly for the two instructions:

```
PULS A      and     LDA ,S+
```

7.2.3 OR/AND Condition Code Register

To set or clear selected bits in the condition-code regsiter the 6809 ORs or ANDs an immediate operand into the register. Thus the 6800's six one-byte instructions SEC, CLC, SEV, CLV, SEI, and CLI hove been replaced by the two two-byte instructions ORCC and ANDCC.

7.2.4 Multiply

The multiply instruction, MUL, multiplies the unsigned 8-bit binary numbers in A and B and leaves the result in the A and B registers treated as one 16-bit number. The MUL instruction can be used as the basis for multiple-precision multiplications.

One common use for a multiply instruction is in the calculation of array subscripts; for example, to get the element M(S1,S2) from an array with dimensions M(100,50), the following code can be used:

```
LDY EM         Get base address of array
LDA S1         First subscript
LDB £100 First dimension
MUL            Multiply D = A * B
ADDD S2        Add second subscript
LDA D,Y        Load value from array element.
```

7.2.5 Sign Extend

The sign extend (SEX) instruction causes all bits in the A register to take on the value of the most significant bit in the B register. It is used to convert signed 8-bit numbers to a signed 16-bit number

7.2.6 Exchange/Transfer Registers

Any register may be transferred to any other of like size with the TFR instruction, or exchanged with any other of like size with the EXG instruction. These instructions are each two bytes long; bits 4-7 of the postbyte specify the source register, and bits 0-3 specify the destination register, as follows:

Register:	D	X	Y	U	S	PC	A	B	CC	DP
Hex Digit:	0	1	2	3	4	5	8	9	A	B

For example, to transfer the contents of A to B (TFR A,B) the postbyte is $89.

Note that the instructions:

```
TFR Y,X       and    LEAX ,Y
```

are similar in effect, but the TFR instruction does not affect any of the status flags.

7.2.7 Synchronize with Interrupt

The 6809's SYNC instruction is used to synchronize software with an external signal. The CPU will stop processing instructions when it encounters a SYNC instruction, and will wait for an interrupt. If the interrupt is non-maskable (NMI) or maskable and enabled, the processor will clear the SYNC state and handle the interrupt just as it would normally. If the interrupt is maskable and disabled, the SYNC state is simply cleared, and execution continues without vectoring to the interrupt service routine. For example, the following routine reads data from an input port on each occurrence of a masked interrupt:

```
FAST SYNC          Wait for interrupt
     LDA ,X        Read from port
     STA ,Y+       Put in array
     DECB          All done?
     BNE FAST      If not, continue.
```

7.2.8 Software Interrupts

The 6809 provides 3 software interrupts, SWI, SWI2, and SWI3, all of which save all the CPU registers on the S stack, and vector through an address in page $FF of memory to a service routine. In addition, SWI disables the FIRQ and IRQ interrupts. SWI is used in the Acorn 6809 monitor as a breakpoint and for trace mode. The other two software interrupts are useful for operating system calls and memory management.

7.3 Addressing Modes

The 6809 has the most powerful set of addressing modes available on any 8-bit microcomputer; it has 59 basic instructions, but recognizes 1464 different variations of instructions and addressing modes. The new addressing modes support modern programming techniques, and some of these have been described in Section 7. The following addressing modes are available on the 6809:

7.3.1 Inherent and Accumulator

In this addressing mode the op-code of the instruction contains all the address information necessary. Examples of Inherent Addressing are: ABX, DAA, SWI, ASRA, and CLRB.

7.3.2 Immediate Addressing

In Immediate Addressing the effective address of the data is the location immediately following the op-code. Both 8 and 16-bit immediate values are used, depending on the size of argument specified by the op-code. Examples of instructions with Immediate Addressing are:

```
LDA E$20
LDX £$F000
LDY £LEAF
```

Note: £ signifies Immediate Addressing
$ signifies hexadecimal value

7.3.3 Extended Addressing

In Extended Addressing the contents of the two bytes immediately following the op-code fully specify the 16-bit effective address used by the instruction. Note that the address generated by an extended instruction defines an absolute address and is not position independent. Examples of Extended Addressing include:

```
LDA ACORN
STX TREE
LDD $2000
```

7.3.4 Extended Indirect

As a special case of indexed addressing (see Section 8.1.7) one level of indirection may be added to Extended Addressing. In Extended Indirect the two bytes following the postbyte of an indexed instruction contain the address of the address of the data. Examples are:

```
LDA (ACORN)
LDX ($FFFE)
STU (TRUNK)
```

7.3.5 Direct Addressing

Direct addressing is similar to extended addressing except that only one byte of address follows the op-code. This byte specifies the lower 8 bits of the address to be used; the upper 8 bits are supplied by the direct page register. Since only one byte of address is required in direct addressing, this mode requires less memory and executes faster than extended addressing. Of course, only 256 locations (one page) can be accessed without redefining the contents of the DP register. To ensure compatability with the 6800 the DP register is set to $00 on Reset. Indirection is not allowed in direct addressing. Some examples are:

```
LDA     $30
SETDP   $10  (Assembler directive)
LDB     $1030
LDD    >CAT
```

Note: >is an assembler directive forcing direct addressing.

7.3.6 Register Addressing

Some op-codes are followed by a byte that defines a register or set of registers to be used by the instruction.

```
TFR    X,Y       Transfers X into Y
EXG    A,B       Exchanges A and B
PSHS   A,B,X,Y   Push onto S stack Y, X, B, then A
PULU   X,Y,D     Pull from U stack D, X, then Y.
```

7.3.7 Indexed Addressing

In all indexed addressing modex one of the pointer registers X, Y, U, S, and PC is used in a calculation of the effective address to be used by the instruction. Five basic types of indexing are available, and are discussed in the following sections. The postbyte of an indexed instruction specifies the basic type and variation of the addressing mode as well as the pointer register to be used. 7.3.7.1 Zero-Offset Indexed - In this mode the selected pointer register contains the effective address of the data to be used by the instruction. This is the fastest indexing mode. Examples are:

```
LDD  0,X
LDA  ,S
```

7.3.7.2 Constant Offset Indexed - In this mode a two's-complement offset and the contents of one of the pointer registers are added to form the effective address of the operand. The pointer register's contents are not changed by the addition. Three sizes of offset are available:

```
 5-bit     (-16 to +15)
 8-bit     (-128 to +127)
16-bit     (-32768 to +32767)
```

The signed 5-bit offset is included in the postbyte and is therefore most efficient in use of bytes and cycles. The 8-bit offset is contained in a single byte following the postbyte. The 16-bit offset is in the two bytes following the postbyte. If an assembler is being used this will select the optimal size automatically.

Examples of constant-offset indexing are:

```
LDA   23,X
LDX   -2,S
LDY   300,X
LDU   CAT,Y
```

7.3.7.3 Accumulator-Offset Indexed - This mode is similar to constant offset indexed except that the two's-complement value in one of the accumulators (A, B, or D) and the contents of one of the pointer registers are added to form the effective address of the operand.
The contents of both the accumulator and the pointer register are unchanged by the addition. The postbyte specifies which accumulator to use as an offset and no additional bytes are required. The advantage of an accumulator offset is that the value of the offset can be calculated by a program at run-time. Some examples are:

```
LDA   B,Y

LDX   D,Y

LEAX  B,X
```

7.3.7.4 Auto Increment/Decrement Indexed - In the auto increment addressing mode the pointer register contains the address of the operand. Then, after the pointer register is used, it is incremented by one or two. In auto decrement the pointer register is decremented before its use as the address of the data. These addressing modes are useful for stepping through tables, moving data, or for the creation of software stacks; the pre-decrement, post-increment nature of these modes makes them behave identically to the U and S stacks. The size of the increment/decrement can be either one or two to allow for tables of either 8 or 16-bit data to be accessed. Some examples of the
auto increment/decrement addressing modes are:

```
LDA    ,X+

STD    ,Y++

LDB    ,-Y

LDX    ,--S
```

7.3.7.5 Indexed Indirect - All of the indexing modes, with the exception of auto increment/decrement by one or a 5-bit offset, may have an additional level of indirection specified. In indirect addressing the effective address is contained at the location specified by the contents of the index register plus any offset. In the example below the A accumulator is loaded indirectly using an effective address calculated from the index register and an offset:

```
            Before execution:     A = XX (anything)
                                  X = $F000
    $0100 LDA (10,X)              effective address is $F010
    $F010 $F1
    $F011 $50                     $F150 is new effective address
    $F150 $AA                     Data
           After execution:       A = $AA
                                  X = $F000 (not changed)
```

Some examples of indexed indirect addressing are:

```
           LDA (,X)
           LDD (10,S)
           LDA (B,Y)
           LDD (,X++)
```

7.3.8 Relative Addressing

The byte or bytes following the op-code for a branch instruction are treated as a signed offset which is added to the program counter. If the branch condition is true the calculated address is loaded into the program counter, and program execution will continue at the location indicated by the program counter. Short (one byte offset) and long (two byte offset) relative addressing modes are available. All of the memory space can be reached using long relative addressing as the effective address wraps around between $FFFF and $0000. Some examples of relative addressing are:

```
           BEQ NEAR       (short)
           LBGT FAR       (long)
```

7.3.9 Program Counter Relative Addressing

The program counter can be used as the pointer register with 8 or 16-bit signed offsets. As in relative addressing the offset is added to the current program counter to create the effective address. The effective address is then used as the address of the operand or data. Program counter relative addressing is used for writing position independent programs; tables related to a particular routine will maintain the same relationship to the routine even if the program is moved. Examples are:

```
LDA    TABLE,PCR
LEAX   CONST,PCR
```

Since program counter relative addressing is a type of indexing, an additional level of indirection is available:

```
LDA    (CAT,PCR)
LDU    (DOG,PCR).
```

Note that all the indexed addressing modes are available with the JMP and JSR instructions, so that:

```
JMP    CAT,PCR
```

can be used to give the same effect as:

```
LBRA   CAT
```

8.0 PROGRAMMING TECHNIQUES

8.1 Position-Independent Code

One particularly powerful feature of the 6809 is its support of position-independent code. Programs written to be position-independent can be loaded anywhere in memory without needing to be re-assembled with a different origin.

The 6809 makes this possible in five ways:

1. Position-independent transfer of control; long and short relative branches are provided.

2. Position-independent temporary storage; workspace may be allocated on the stack, rather than using fixed RAM locations.

3. Position-independent access to constants within the same block of code, using program-counter relative addressing. E.g. LDA CONST,PCR.

4. Position-independent access to tables within the same block vi code. The start address of the table is loaded into X using the 'load effective address' instruction LEAX TABLE,PCR; the table can then be accessed using indexed addressing.

5. Position-independent access to constants and variables located in ROM and RAM outside the block of position-independent code, and whose addresses are not known at the time that the code is assembled. This is achieved by providing a table, external to the position-independent block of code, which gives the addresses of all the external variables, and the constants, needed by the program. Before the position-independent routine is called, a register is pointed to this table: e.g. LDX £TABLE. The routine can then load constants from the table using indexed addressing, as in LDA 2,X, and access variables in RAM by indirecting through the addresses in the table: e.g. LDA (5,X) , or STY (8,X).

8.1.1 Example

The following section takes a simple program, to convert a binary number into decimal, and shows how to modify it so that it is position independent, making use of the features just described.

The first version of the program, Fig. 7.1, is not relocatable because the instructions:

```
LDX ££K10TAB
```

and `CMPX £K10TAB+8`

contain absolute addresses. If re-assembled with a different origin these bytes in the program would change, and so the program is not position independent.

The first version of the program suffers from two other drawbacks. First, it is not re-entrant. In other words, it needs some dedicated RAM locations for the variables COUNT and TEMP. The routine could not be used by both an interrupt service routine and a main program because one call might overwrite the variables being used by the other call. Secondly, the program changes the values of some of the registers. This is bad practice; the routine could not be incorporated into a larger program without some caution.

The second version of the binary-to-decimal program, in Fig. 7.2, removes all three drawbacks; it is position independent, re-entrant, and saves the values of all the registers. The drawbacks are solved as follows:

Firstly, program-counter relative addressing is used to pick up the address of the table. The instruction:

```
LEAX £K10TAB
```

does not change depending on its position in memory. The end of the table is detected by testing the value of the power of ten. Secondly, the two temporary locations are replaced by two stack locations. The space is allocated by the instruction:

```
LEAS -2,S
```

```
                    *
                    * BINARY-TO-DECIMAL VERSION 1
                    *
                    * NON RELOCATABLE
                    * NON RE-ENTRANT
                    *
0000                COUNT   RMB1
0001                TEMP    RMB1
                    *
0002      2710      K10TAB  FDB   10000
0004      03E8              FDB   1000
0006      0064              FDB   100
0008      000A              FDB   10
                    *
000A 8E   0002      BINDEC  LDX   £K10TAB       POINT TO TABLE
000D 7F   0000      LOOP1   CLR   COUNT
0010 20   03                BRA   NOINC
0012 7C   0000      LOOP2   INC   COUNT
0015 A3   84        NOINC   SUBD  ,X            SUBTRACT POWER OF 10
0017 24   F9                BCC   LOOP2
0019 E3   81                PRN5  ADDD ,X++     MAKE POSITIVE
001B B7   0001              STA   TEMP
001E B6   0000              LDA   COUNT
0021 BD   FA8B              JSR   HEXOUT        IN MONITOR
0024 B6   0001              LDA   TEMP          RESTORE D
0027 8C   000A              CMPX  £K10TAB+8     ALL DONE?
002A 26   El                BNE   LOOP1
002C IF   98                TFR   B,A           GET REMAINDER
002E BD   FA8B              JSR   HEXOUT        LAST DIGIT OF RESULT
0031 39                     RTS
```

Fig. 8.1. Original binary-to-decimal routine.

```
                    *
                    *  BINARY-TO-DECIMAL VERSION 2
                    *
                    *  RELOCATABLE AND RE-ENTRANT
                    *  ALL REGISTERS PRESERVED
                    *
0000        2710    K10TAB   FDB     10000
0002        03E8             FDB     1000
0004        0064             FDB     100
0006        000A             FDB     10
0008        FF               FCB     $FF          END OF TABLE MARKER
                    *
0009 34     36      BINDEC   PSHS    A,B,X,Y      SAVE REGISTERS
000B 30     8CF2             LEAX    K10TAB,PCR   POSITION INDEPENDENT
000E 32     7E               LEAS    -2,S         GET WORKSPACE ON STACK
0010 6F     E4      CV       CLR     ,S
0012 20     02               BRA     NOINC
0014 6C     E4      CV2      INC     ,S
0016 A3     84      NOINC    SUED    ,X           SUBTRACT POWER OF 10
0018 24     FA               BCC     CV2
001A E3     81      CV3      ADDD    ,X++         MAKE POSITIVE AGAIN
001C A7     61               STA     1,S          SAVE A
001E A6     E4               LDA     ,S           GET COUNT
0020 BD     FA8B             JSR     HEXOUT       PRINT A IN HEX DIGIT
0023 A6     61               LDA     1,S          RESTORE A
0025 6D     84               TST     ,X           DONE?
0027 2A     E7               BPL     CV
0029 1F     98               TFR     B,A          GET REMAINDER
002B BD     FA8B             JSR     HEXOUT       PRINT IT
002E 32     62               LEAS    2,S          RESTORE WORKSPACE
0030 35     B6               PULS    A,B,X,Y,PC   RESTORE REGS. & RETURN
```

Fig. 8.2. Improved binary-to-decimal routine.

and restored by the instruction:

```
LEAS 2,X
```

Finally, the contents of the registers used by the routine are saved on the stack on entry to the routine with the instruction:

```
PSHS A,B,X,Y
```

and restored on exit from the routine.

8.2 Recursive Programming

The provision of a user stack, and the wide variety of addressing modes, make the 6809 very suitable for recursive programming. Many programming problems can be solved better recursively than by conventional iterative methods, and such solutions are often shorter and simpler to understand. Typical applications include the writing of high-level language compilers and syntax analyzers, and algebraic manipulation.

A recursive routine is a routine whose definition includes a reference to itself. As an example of the ease with which the 6809 handles recursive programming a routine to calculate binomial coefficients will be considered. The binomial coefficient ${}^{n}C_{r}$ gives the number of different combinations of n things taken r at a time. Thus the number of different combinations of three things, A, B, and C, taken two at a time is ${}^{3}C_{2}$ or 3; namely AB, AC, and BC.

One possible recursive definition of this function is as follows:

$$C(n,r) = 1 \quad \text{if } n=0$$

$$= 1 \quad \text{if } n=r$$

$$= C(n-1,r) + C(n-1,r-1) \quad \text{otherwise.}$$

For this definition the function has beed written in the form C(n,r) rather than the traditional ${}^{n}C_{r}$.

The provision of a user stack on the 6809 makes it possible to push and pull the arguments to subroutines without interference from subroutine return addresses. In the routine of Fig. 7.3 the values of n and r are passed to the routine in the A and B registers respectively, and the routine returns the result on the user stack. If n has a value other than 0 or r the routine will be entered recursively, and the user stack will expand to hold intermediate results.

```
                *
                * RECURSIVE SUBROUTINE TO CALCULATE
                * BINOMIAL COEFFICIENTS.
                *
                * RELOCATABLE & RE-ENTRANT
                *
0000 36   06    NCR     PSHU     A,B
0002 5D                 TSTB
0003 27   04            BEQ      ONE        B=O?
0005 El   C4            CMPB     ,U
0007 26   05            BNE      NONE       A=B?
0009 CC   0001  ONE     LDD      El
000C 20   OD            BRA      RESULT     RETURN 1
000E 4A         NONE    DECA
000F 8D   EF            BSR      NCR        C(A-1,B)
0011 EC   42            LDD      2,U
0013 5A                 DECB
0014 4A                 DECA
0015 8D   E9            BSR      NCR        C(A-1,B-1)
0017 37   06            PULU     A,B
0019 E3   Cl            ADDD     ,U++       ADD RESULTS
001B ED   C4    RESULT  STD      ,U         ON USER STACK
001D 39                 RTS
                *
                * TEST C(7,3)
                *RESULT SHOULD BE $23
                *
001E CE   0300  TEST    LDU      £$0300     USER STACK
0021 CC   0703          LDD      E$0703     SET UP A AND B
0024 BD   0000          JSR      NCR
0027 37   10            PULU X   PRINT RESULT
0029 7E   FA75          JMP      OPXREG     FROM STACK.
                *
```

Fig. 8.3. Recursive routine to calculate ${}^{n}C_{r}$.

8.3 Software Compatibility with 6800.

The 6809 is source-code compatible with the 6800; in other words, any assembler program for the 6800 can be re-assembled for the 6809. However the resulting program is unlikely to be optimal, and in most cases where size or speed are important it is probably better to rewrite the program to take advantage of the 6809's more advanced features. Many of the 6800's instructions have direct equivalents on the 6809. The following section lists exceptions to this:

6800 Instruction	6809 Equivalent
ABA	PSHS B; ADDA ,S+
CBA	PSHS B; CMPA ,S+
CLC	ANDCC #$FE
CLI	ANDCC #$EF
CLV	ANDCC #$FD
CPX	CMPX P
DES	LEAS -1,S
DEX	LEAX -1,X
INS	LEAS 1,S
INX	LEAX 1,X
LDAA	LDA
LDAB	LDB
ORAA	ORA
ORAB	ORB
PSHA	PSHS A
PSHB	PSHS B
PULA	PULS A
PULB	PULS B
SBA	PSHS B; SUBA ,S+
SEC	ORCC #$01
SEI	ORCC #$10
SEV	ORCC #$02
STAA.	STA
STAB	STB
TAB	TFR A,B; TST A
TAP	TFR A,CC
TBA	TFR B,A; TST A
TPA	TFR CC,A
TSX	TFR S,X
TXS	TFR X,S
WAI	CWAI #$FF

8.3.1 Software Incompatibilities Between 6800 and 6809.

1. The new stacking order on the 6809 exchanges the order of ACCA and ACCB; this allows ACCA to stack as the MS byte of the pair.

2. The new stacking order on the 6809 invalidates previous 6800 code which displayed X or PC from the stack.

3. Additional stacking length on the 6809 stacks five more bytes for each NMI, IRQ, or SWI when compared to 6800.

4. The 6809 stack pointer points directly at the last item placed on the stack, instead of the location before the last item as in 6800. In general this is not a problem since the most-usual method of pointing at the stack in the 6800 is to execute a TSX. The TSX increments the value during the transfer, making X point directly at the last item on the stack.

   The stack pointer may thus be initialized one location higher on the 6809 than in the 6800; similarly, comparison values may need to be one location higher.

   Any 6800 program which does all stack manipulation through X (i.e., LDX #CAT, TXS instead of LDS #CAT) will have an exactly-correct stack translation when assembled for 6809.

5. Instruction timings in 6809 will, in general, be different from other 6800-family processors.

6. The 6809 uses the two high-order condition code register bits. Consequently, these will not, in general, appear as l's as on the 6800.

1. The 6809 TST instruction does not affect the C-flag, while 6800 TST does clear the C-flag.

2. The 6809 right shifts (ASR, LSR, ROR) do not affect V; the 6800 shifts set $V = b_7 \oplus b_6$.

3. The 6809 H-flag is not defined as having any particular state after subtract-like operations (CMP, NEG, SBC, SUB); the 6800 clears the H-flag under these conditions.

4. The 6800 CPX instruction compared MS byte then LS byte; consequently only the Z-flag was set correctly for branching. The 6809 instructions (CPX/CMPX) set all flags correctly.

5. 11. The 6809 instruction LEA may or may not affect the Z-flag depending upon which register is being loaded; LEAX and LEAY do affect the Z-flag, while LEAS and LEAU do not. Thus, the User Stack does not exactly emulate the index registers in this respect.

8.4 Software Compatibility with 6502.

Acorn have decided to support both the 6502 and the 6809 because these processors each have advantages for different application areas. The 6502 will generally produce shorter, faster programs in simple applications, such as industrial control, where only 8-bit arithmetic is needed, and where data is to be moved between pre-defined areas of memory. For more complicated programming tasks, such as the writing of high-level language compilers, text processors, and interpreters, the 6809's more sophisticated addressing modes and 16-bit arithmetic operations will make the 6809 better suited to the task.

Most programs for the 6502 can be translated fairly directly into instructions for the 6809, but because programs for the 6502 tend to make use of assumptions about the positions of data and variables, programs translated directly from 6502 to 6809 will generally be longer and slower.

The following differences between the two processors should be noted:

1. The order of the address bytes on the 6502 is the opposite to that on the 6809. Thus: JSR $1234 is

```
20 34 12       on the 6502, but:

BD 12 34       on the 6809.
```

2. The SBC and CMP operations on the 6502 set the carry flag if there was no borrow, but clear the carry under the same circumstances on the 6809. For example:

```
LDA $40

SBC $20      sets the carry on the 6502 but clears it on the 6809.
```

3. The X and Y registers on the 6809 are each 16 bits wide, but only 8 bits wide on the 6502.

4. The 6502's instructions INX, DEX, INY, DEY correspond roughly to the 6809's instructions LEAX 1,X; LEAX -1,X; LEAY 1,Y; and LEAY -1,Y. However, note that whereas the 6502's instructions affect both the N and Z flags, the 6809 instructions affect only the Z flag.

5. The 6502's indexed indirect addressing mode can be directly replaced by the 6809's indexed indirect mode:

```
    LDA (TABLE,X)      on the 6502 becomes:
    LDA (TABLE,X)      on the 6809.
```

However the 6502's post-indexed indirect mode:

```
    LDA (TABLE),Y
```

has no direct equivalent in the 6809. Instead the address can be held in a 16-bit register, such as U, and the B register can then be used for indexing:

```
    LDU TABLE
    LDA B,U
```

6. The 6502's BIT instruction not only sets the Z flag depending on the result of ANDing the accumulator with the specified memory location, but also copies bits 6 and 7 of the location into the N and V flags respectively. The 6809's BIT instruction does not do this, but the TST instruction can be used instead.

9.0 ASSEMBLY INSTRUCTIONS

9.1 6809 Card

Before attempting to assemble the 6809 card check that all the components are present and that none have been damaged.
It is worthwhile taking a few minutes to make sure that all the components can be identified. Sometimes components will be substituted in case of supply difficulties. For instance, 0.047 uF capacitors may replace 0.1 uF capacitors shown on the parts list. The components substituted will in no way detrimental to the system's operation. Also some manufacturers have similar but different type numbers •

For capacitors note that the value may be expressed in one of two ways:-

100 nF = 0.1 uF
10 nF = 0.01 uF
1 nF = 1000 pF
0.1 nF = 100 pF etc.

Capacitors supplied with the Acorn cards are usually identified by a 3 digit number, the first two digits being the first two digits of the value and the third being the number of following zeroes eg.

101 = 10 and one zero ie. 100 pF
103 = 10 and three zeroe's ie. 10000 pf = 10 nF
473 = 47 and three zero's i.e. 47000 pF = 47 nF = 0.047 uF.

If in doubt about the capacitor values, count the number of each of type supplied in the kit and then identify them using the parts list quantities.
The electrolytic capacitors are polarised and the positive end marked + must be located as indicated on the circuit card.

6809 C.P.U. Card Parts List

PCB	Printed Circuit Board 200.012	
IC1	6809 Microprocessor	& 40 pin socket
IC2	6522 Versatile Interface Adapter	"
IC3	74LS244 Buffer	& 20 pin socket
IC4	2716 2K monitor ROM	& 24 pin socket
IC5	2114 RAM	& 18 pin socket
IC6	2114 RAM	"
IC7	74LS244	& 20 pin socket
IC8	74LS244	"
IC9	74LS244	"
IC10	INS8208 (or DP8304)	"
IC11	74S470 Bipolar ROM	"
IC12	74LS74	& 14 pin socket
IC13	74LS00	"
IC14	74LS00	"
IC15	74LS86	"
XTAL	4 MHz Crystal	
R1 - 3	3 off 1K resistor	
R4 - 9	6 off 560R resistor	
R10 - 15	6 off 3K3 resistor	
C1	22uF electrolytic capacitor	
C2 - 6	5 off 47 (or 100) nF capacitor	
C7 - 8	2 off 24pF capacitors	
C9	Optional - not supplied.	

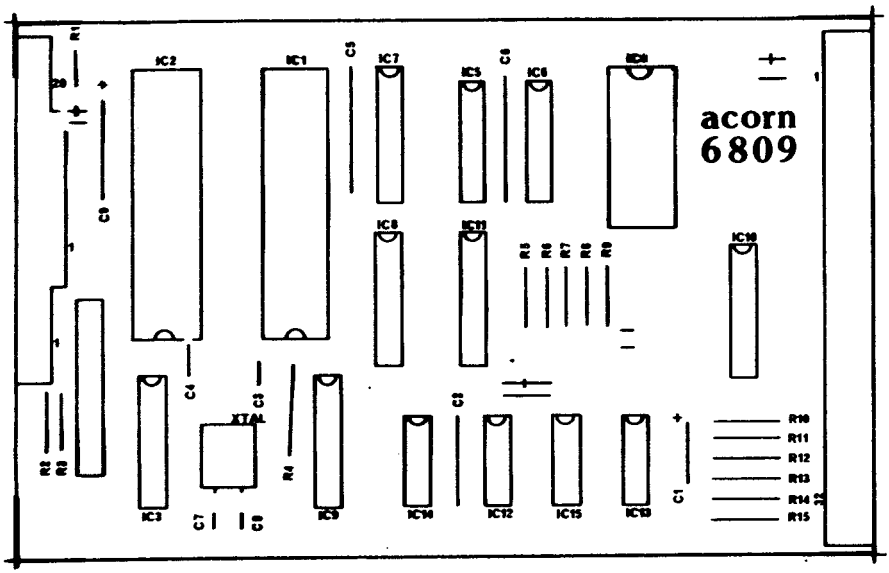
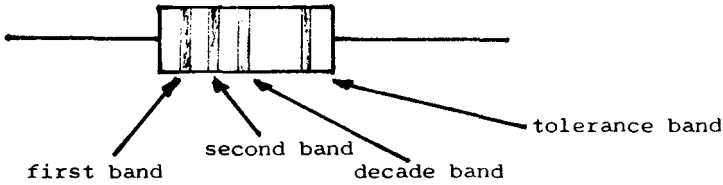


Fig. 9.1. 6809 Card Component Layout.

The resistor colour code is shown here.



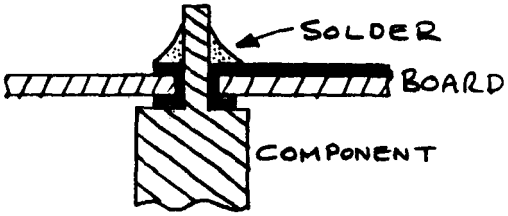
The first and second bands give the resistor value and the decade band shows the number of zeros following:-

0 Black
1 Brown
2 Red
3 Orange
4 Yellow
5 Green
6 Blue
7 Violet
8 Grey
9 White

e.g. Yellow, Violet, Orange is Yellow, Violet . 4,7 and Orange 3 zeros i.e. 000. So the value is 47000 ohms, i.e. 47 kilo-ohms or 47K.

The tolerance band is red for ± 2%, gold for ± 5% or silver for ± 10%.

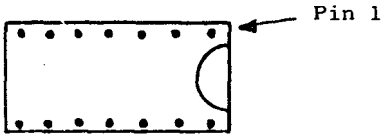
Assembling the card will require a considerable amount of soldering and a small electric soldering iron is essential with a diameter at the end of the bit not exceeding 0.1 inches. The iron should be rated between 10 and 30 watts and fine 22 guage flux cored solder should be used. If you have never soldered before we advise you not try to assemble the card without assistance as Acorn Computer Ltd. can not accept responsibility for kits which have beem improperly assembled. When soldering make sure the component is well pushed on to the board as shown, use a minimum of solder and once the solder has run remove the iron.



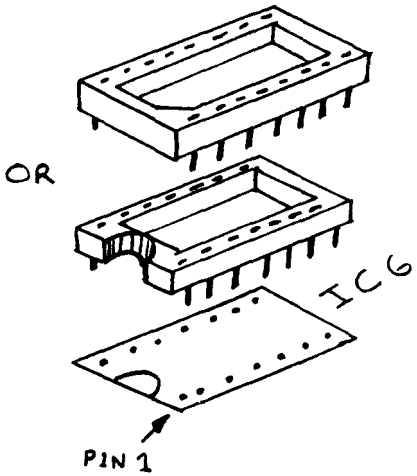
Some of the integrated circuits used in the system employ M.O.S. technology and they can be damaged by static electricity. As a general rule if there is no noticable static charge in the area and no nylon clothes or carpets are present all will be well. An earthed soldering iron should be used when soldering on a board containing M. O.S., I.C.'s and the I.C.'s should be kept on the conducting foam on which they are supplied until required.

The Acorn Printed Circuit cards are double sided, through hole plated glass fibre and are manufactured to the highest standards. A layer of green solder resist ensures that accidental solder splashes do not stick to the tracks and a clearly marked white silk screen indicates component positions. Examine the cards for faults or damage before proceeding. It is not necessary to solder through holes which connect one side of a board to the other and do not have a component lead in them and attempting to do so can break the through hole plating and thus the connection. All soldering should only be done on the opposite side of the board to the components (i.e. side 1).

The cards are each supplied with a full set of integrated circuit sockets. The sockets must be fitted the right way round, on the circuit board viewing it from the top pin 1 of an I.C. is identified as shown:-



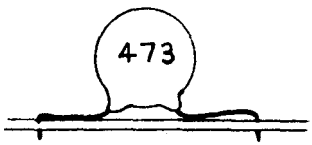
The sockets will have either a 45° chamfer for pin 1 or a semi circular cut out as shown:-



Note that on the 6809 card IC1 and IC2 are the opposite way round to the other sockets nearby. Fit the sockets one at a time and ensure that they are pressed fully down with no leads bent under the socket before first soldering two diagonally opposite pins at the corners. Check that the socket is the right way round and successfully fitted nefore soldering the rest of the pins.

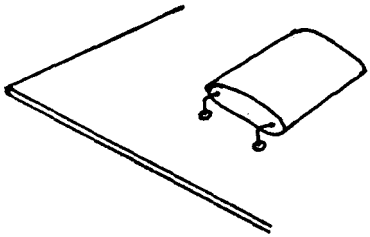
There is no need to snip off the excess of the socket pins.

After the I.C. sockets the resistors and capacitors are fitted to the circuit cards. Identify the component from the component list and fit it to the board. Some capacitors will need to be fitted as shown.



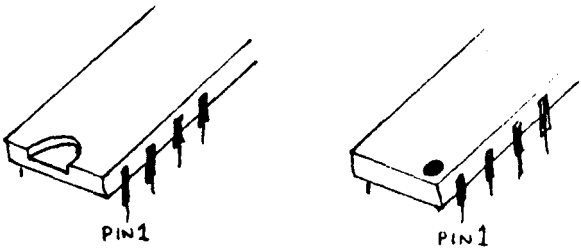
Do not crack the capacitor body when bending the leads.

The crystal on the C.P.U. card is fitted as shown:-



Again bend the leads away from the component body and lay the Crystal down on the board before soldering. Snip off any excess leads under the board.

The connector is fixed to a card using two 2.5 mm screws and nuts before soldering the pins. When all of the components are soldered the integrated circuits may be fitted in their sockets, pin 1 is identified by either a semicircle or a dot as shown:-



Identify the I.C. type from the components list and plug it into the appropriate socket. If the leads are splayed out press them all in together until the I.C. fits easily to the socket.

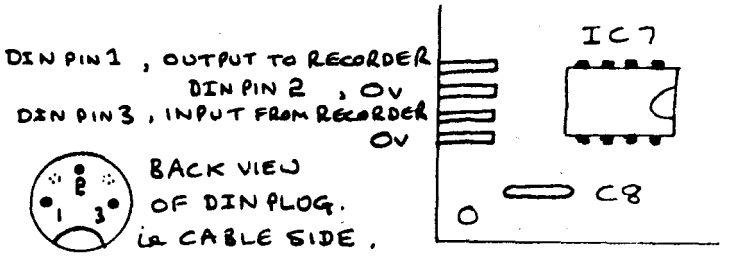
Note that IC1 and IC2 are the opposite way round to the other ICs on the 6809 card.

9.2 Visual Display Unit Card

The 6809 card is designed to work with the Acorn VDU Card, which will drive a monitor or television. The standard output is a 1 volt 75 ohm composite video signal, and a 75 ohm coaxial cable connected to this will drive a monitor directly. This signal may also be used to drive a UHF modulator to give an interface to a standard television.

9.3 Cassette Interface

The Acorn cassette interface card is a Computer Users Tape Standard interface which connects to the recorder as shown:-



This recorder output consists of one of two tones, 2.4 KHz represents a logic 1 and 1.2 KHz a logic 0. Each bit i.e. o or 1 lasts for 3.3 mS giving an operating speed of 300 bits/ second.

Both recording and playback are crystal controlled giving a low error rate and except on very cheap recorders whose speed may vary, no trouble should be experienced in transferring tapes from one machine to another.

9.4 Keyboard and Cassette Interface Connections

A parallel ASCII keyboard is required to be fitted on the front of C.P.U. card. A 5 volt supply for the keyboard is available and the board requires a 'low for key depressed' strobe signal. The connections may be soldered to the front of the board as follows:-

(top)	20	+ 5 volts
	19	Reset
	18	Key strobe
	17	Data bit 6
	16	Data bit 5
	15	Data bit 4
	14	Data bit 3
	13	Data bit 2
	12	Data bit 1
	11	Data bit 0
	10	E
	9	CASIN
	8	CASOUT
	7	-
	6	-
	5	-
	4	-
	3	-
	2	-
(bottom)	1	0 volts

The reset is provided by a simple push button connected to 0 volts. This is often available as an extra key on ASCII keyboards.

If desired a 20 way pcb header can be fitted to the C.P.U. card in which case the keyboard connections are as follows:-

20 Data bit 5	
	19 Data bit 6
18 Data bit 4	
	17 Key strobe
16 Data bit 3	
	15 Reset
14 Data bit 2	
	13 + 5 volts
12 Data bit 1	
	11 0 volts
10 Data bit 0	
	9 -
8 E	
	7 -
6 CASIN	
	5 -
4.CASOUT	
	3 -
2 -	
	1 -

9.5 Backplane

The 'A' side of the edge connector on the 6809 card carries all the essential bus signals, and these should be connected to other cards in the system by means of a suitable backplane. A piece of 0.1" matrix strip veroboard may be used, the cards connecting to the backplane by means of 32-way indirect plugs and sockets. Alternatively the Acorn backplane card may be used; this will accomodate 64-way indirect connectors, thus providing access to the 'B' side connections on the cards.

The connections between the cassette interface and the 6809 CPU card may be made by connecting the following pins on the backplane connectors:

CPU card pin A29 to cassette interface pin B11, E clock

" B19 " " B12, CASIN

" B20 " " B13, CASOUT.

9.6 Power Supply

To power the 6809 card and the VDU card a 5 volt stabilized power supply will be needed; this should be capable of supplying at least 1.5 amps.

```
0001                    *               Acorn monitor for 6809 processor
0002                    *
0003                    *
0004                    *               ISSUE 1  -  V49  -  FEB 1980
0005                    *
0006                    *
0007                    *               This program handles a memory mapped vdu, encoded keyboard,
0008                    *               cassette interface, parallel printer, and mini-floppy bootstrap.
0009                    *
0010          F800      MPROM   EQU     $F800           normal monitor position at top of memory
0011          0003      MONDP   EQU     $0300/256       direct page for monitor
0012                    *
0013          0019      ROWS    EQU     25              number of rows on display
0014          0028      COLS    EQU     40              number of characters per row
0015          0400      PSIZE   EQU     1024            size of display memory in total
0016          0400      PAGE    EQU     $0400           location of memory that vdu uses
0017          0800      CRTC    EQU     $0800           location of crt controller on vdu card
0018          000C      PAGHI   EQU     12              page address register, high byte
0019          000E      CURHI   EQU     14              cursor address register, high byte
0020                    *
0021          0040      DRIVE   EQU     $40             drive to bootstrap from, other drive is $80
0022          0A00      FLOPY   EQU     $A00            location of floppy disc controller
0023          0A00      FDCC    EQU     FLOPY+0         command register
0024          0A00      FDCS    EQU     FDCC            status register
0025          0A01      FDCP    EQU     FLOPY+1         parameter register
0026          0A01      FDCR    EQU     FDCP            result register
0027          0A02      FDRST   EQU     FLOPY+2         reset register
0028          0A04      FDCD    EQU     FLOPY+4         data register
0029                    *
0030          0980      KVIA    EQU     $0980           location of versatile interface adaptor
0031          0980      KORB    EQU     KVIA+$0         output register b
0032          0980      KIRB    EQU     KORB            input register b
0033          0981      KORA    EQU     KVIA+$1         output register a
0034          0981      KIRA    EQU     KORA            input register a
0035          0982      KDDRB   EQU     KVIA+$2         data direction register b
0036          0983      KDDRA   EQU     KVIA+$3         data direction register a
0037          0984      KT1CL   EQU     KVIA+$4         timer 1 counter low
0038          0985      KT1CH   EQU     KVIA+$5         timer 1 counter high
0039          0986      KT1LL   EQU     KVIA+$6         timer 1 latch low
0040          0987      KT1LH   EQU     KVIA+$7         timer 1 latch high
0041          0988      KT2CL   EQU     KVIA+$8         timer 2 counter low
0042          0989      KT2CH   EQU     KVIA+$9         timer 2 counter high
0043          098A      KSR     EQU     KVIA+$A         shift register
0044          098B      KACR    EQU     KVIA+$B         auxilliary control register
0045          098C      KPCR    EQU     KVIA+$C         peripheraal control register
0046          098D      KIFR    EQU     KVIA+$D         interrupt flag register
0047          098E      KIER    EQU     KVIA+$E         interrupt enable register
0048          098F      KORA2   EQU     KVIA+$F         input/output register a without handshake
0049                    *
0050          0F00      INTDEL  EQU     15*256          delay for single instruction trace is 15 cycles
0051          0040      T1IFLG  EQU     %01000000       interrupt flag position for timer 1
0052          0010      CB1FLG  EQU     %00010000       interrupt flag position for keyboard input
0053                    *
0054          00EF      IKPCR   EQU     %11101111       initial value for peripheral control register,
0055                    *                               bit 0,    positive edge printer interrupt on ca1,
0056                    *                                         but not used in this monitor.
0057                    *                               bits 1-3, output to activate printer, normally high
0058                    *                               bit 4,    negative edge keyboard interrupt
0059                    *                               bits 5-7, cassette output initially set high
0060                    *
0061          00D0      IKIER   EQU     %11010000       interrupt enable register control
0062                    *                               bit 4,  keyboard interrupt enable
0063                    *                               bit 6,  timer 1 interrupt enable
0064                    *                               bit 7,  set interrupts enabled
0065                    *                               other enables not altered
0066                    *
0067          0002      PSTRB   EQU     %00000010       bit postion of printer strobe in pcr
0068          0020      COPBIT  EQU     %00100000       bit position that controls cassette output in pcr
0069                    *
0070          003F      SWI     EQU     $3F             a software interrupt used for breakpoints
0071                    *
0072          0052      BUFLEN  EQU     81+1            buffer up to 80 characters from keyboard
0073                    *
0074          002A      PROMPT  EQU     '*              monitor prompt character
0075          007F      RUBCH   EQU     $7F             keyboard character that does rubout operation
0076          007F      BSPACE  EQU     $7F             character that backspaces vdu, also used to do rubout
0077          000A      LF      EQU     $0A             linefeed character
0078          000D      CR      EQU     $0D             carriage return character
0079          0020      SPACE   EQU     $20             blank space character
0080          002C      COMMA   EQU     ',              a comma character
0081          002D      MINUS   EQU     '-              a minus character
0082          000C      FFEED   EQU     $0C             character used as clear screen command
0083          003B      SEMIC   EQU     ';              a semicolon
0084                    *
0085          0010      IRQ     EQU     $10
0086          0004      ZERO    EQU     $04
0087                    *
0088                    *               definitions of variables on page 3
0089                    *
0090 035B                       ORG     $035B
0091                    *
0092          035B      ISTACK  EQU     *               stack pointer starts here
0093          035B      RTAB1   EQU     *               ram table 1 starts here
0094                    *
0095                    *
0096                    *               this table copied from rom on start up
0097                    *
```

```
0098 035B       0002        STACK  RMB    2              positon of stack pointer when empty
0099 035D       0002        NTRACE RMB    2              number of instructions to trace before stopping
0100 035F       0001        BSECHO RMB    1              character sent to backspace display
0101 0360       0001        ECHOF  RMB    1              keyboard buffer/echo control
0102                        *                            bits 0-5,  don't cares
0103                        *                            bit 6      echo console input to console output if set
0104                        *                            bit 7      buffer input lines, allow rubout if set
0105                        *
0106 0361       0001        PFLAG  RMB    1              printer control flag, echo console output to printer if
0107 0362       0001        PNEW   RMB    1              this character not sent to printer            non-zero
0108 0363       0002        DELCNT RMB    2              delay count for cassette, controls baud rate
0109 0365       0002        COPADR RMB    2              address for console output
0110 0367       0002        CINADR RMB    2              address for console input
0111 0369       0002        CASOPA RMB    2              address for cassette output
0112 036B       0002        CASINA RMB    2              address for cassette input
0113 036D       0002        PRINTI RMB    2              address of printer output routine
0114 036F       0002        FUNCTI RMB    2              address of vdu function table
0115 0371       0002        CMNDI  RMB    2              address of monitor command table
0116 0373       0002        IRQRTS RMB    2              address to go to on timer 1 interrupt
0117 0375       0002        LINEPT RMB    2              address of memory input line, 0 if none
0118 0377       0002        IRESV  RMB    2              address of reserved vector routine
0119 0379       0002        ISWI3  RMB    2              address of swi3 routine
0120 037B       0002        ISWI2  RMB    2              address of swi2 routine
0121 037D       0002        IFIRQ  RMB    2              address of firq routine
0122 037F       0002        IIRQ   RMB    2              address of irq routine
0123 0381       0002        ISWI   RMB    2              address of swi routine
0124 0383       0002        INMI   RMB    2              address of nmi routine
0125 0385       0002        OFFSET RMB    2              cassette load offset
0126                        *
0127                        *              general variables for monitor use
0128                        *
0129 0387       0002        HEADST RMB    2              static head pointer into line buffer
0130 0389       0002        HEADDY RMB    2              dynamic head pointer into line buffer
0131 038B       0002        TAIL   RMB    2              tail pointer into line buffer
0132 038D       0002        MSTACK RMB    2              stack saved whilst memory interpreting
0133 038F       0001        CROW   RMB    1              current row of cursor on display
0134 0390       0001        CCOL   RMB    1              current column of cursor on display
0135 0391       0002        CPAGE  RMB    2              current start of display page in memory
0136 0393       0002        MSAV   RMB    2              saved address for memory command
0137 0395       0002        GSAV   RMB    2              saved address for go command
0138 0397       0006        NAME   RMB    6              saved name for cassette input/output
0139 039D       0002        CSSTRT RMB    2              saved cassette output start address
0140 039F       0002        CSEND  RMB    2              saved cassette output end address
0141 03A1       0001        ONLINE RMB    1              flag set to zero when find cr in input line
0142 03A2       0001        LASTC  RMB    1              saved last character from input line
0143 03A3       0002        CBREAK RMB    2              current address of a breakpoint, $FFFF if none
0144 03A5       0002        NBREAK RMB    2              number of breakpoints to ignore before stopping user
0145 03A7       0001        CINST  RMB    1              user instruction at breakpoint address
0146 03A8       0002        CTRACE RMB    2              number of instructions left to trace before stopping us
0147 03AA       0002        USRSTK RMB    2              saved user stack pointer when user halted
0148 03AC       0002        TEMP   RMB    2              temporary storage
0149 03AE       0052        BUFFER RMB    BUFLEN         line input buffer
0150                        *
0151 F800                          ORG    MPROM
0152            E000               PUT    $E000
0153                        *
0154                        *
0155                        *              hardware reset starts at this address
0156                        *
0157 F800 86    03       2 RESET  LDA    #MONDP         setup..
0158 F802 1F    8B       6        TFR    A,DP           ..direct page
0159             0003               SETDP  MONDP          tell assembler
0160 F804 8E    FF25     3        LDX    #PTAB1         rom table start
0161 F807 CE    0358     3        LDU    #RTAB1         ram table start
0162 F80A A6    80       6 RST1   LDA    ,X+            copy rom..
0163 F80C A7    C0       6        STA    ,U+            ..to ram
0164 F80E 8C    FF51     4        CMPX   #PTAB2         until end..
0165 F811 26    F7       3        BNE    RST1           ..of table
0166 F813 10DE  5B       6        LDS    STACK          setup stack pointer
0167 F816 BE    F7FE     6        LDX    MPROM-2        check for..
0168 F819 8C    A55A     4        CMPX   #$A55A         ..extra rom,
0169 F81C 26    04       3        BNE    STRT1          not there,
0170 F81E AD    9F F7FC 12        JSR    [MPROM-4]      else call it
0171 F822 CE    FBEA     3 STRT1  LDU    #BACK          put monitor return..
0172 F825 34    40       7        PSHS   U              ..onto stack
0173 F827 CC    000C     3        LDD    #12            put dummy..
0174 F82A 34    02       6 STRT2  PSHS   A
0175 F82C 5A             2        DECB                  ..registers..
0176 F82D 26    FB       3        BNE    STRT2          ..onto stack
0177 F82F 10DF  AA       6        STS    USRSTK         save stack pointer
0178 F832 8E    03AE     3        LDX    #BUFFER        get start of buffer..
0179 F835 9F    89       5        STX    HEADDY         ..and setup..
0180 F837 9F    87       5        STX    HEADST
0181 F839 9F    8B       5        STX    TAIL           ..pointers
0182 F83B 17    00E7     9        LBSR   CRTCI          initialise crt controller
0183 F83E 17    00D1     9        LBSR   VIAI           and via chips
0184 F841 0F    A2       6        CLR    LASTC          set no saved character
0185 F843 17    0400     3        LBSR   BRKOUT         remove if exists
0186 F846 8E    FFFF     3        LDX    #$FFFF         then set..
0187 F849 9F    A3       5        STX    CBREAK         ..non-existing
0188 F84B 1C    EF       3        CLI                   allow interrupts
0189             0000               SETDP  0
0190 F84D 86    2A       2 MON    LDA    #PROMPT        send prompt..
0191 F84F BE    0375     6        LDX    LINEPT         ..unless..
0192 F852 26    03       3        BNE    MON1           ..memory input
0193 F854 17    01CA     9        LBSR   CONIN
```

```
0194 F857 B7   03A1      5 MON1   STA    ONLINE          set no cr yet
0195 F85A 86   03        2 PARSE  LDA    #MONDP
0196 F85C 1F   8B        6        TFR    A,DP
0197           0003               SETDP  MONDP
0198 F85E 9E   75        5        LDX    LINEPT          see if mem input,
0199 F860 27   04        3        BEQ    PARSEC          if so, then
0200 F862 A6   84        4        LDA    ,X              see if null yet
0201 F864 27   41        3        BEQ    MEND            end if is a null
0202 F866 8D   17        7 PARSEC BSR    CONCHR          get input
0203 F868 27   E3        3        BEQ    MON             prompt on cr
0204 F86A 9E   71        5        LDX    CMNDI           else command table..
0205 F86C 17   0137      9        LBSR   DISPCH          ..search
0206 F86F 20   E9        3        BRA    PARSE
0207                       *
0208                       *               enter here for monitor to use memory input line
0209                       *               (x) is start of line, ends with a null.
0210                       *               multiple input lines are allowed, each line ends
0211                       *               with a carriage return
0212                       *
0213                       *               exits with (a) zero if all ok, else (a) is $FF
0214                       *               if found null too early, else (a) is character
0215                       *               causing error.
0216           0000                SETDP  0
0217 F871 34   00        6 MEMUSE PSHS   DP
0218 F873 10FF 038D      7        STS    MSTACK          save stack for return
0219 F877 BF   0375      6        STX    LINEPT          save pointer in memory
0220 F87A 7F   03A2      7        CLR    LASTC           set none saved
0221 F87D 20   CE        3        BRA    MON             and call monitor
0222                       *
0223 F87F 86   0D        2 CONCHR LDA    #CR             assume cr now
0224 F881 7D   03A1      7        TST    ONLINE          and if found cr then
0225 F884 27   06        3        BEQ    CON1            is correct
0226 F886 8D   08        7        BSR    CONIN           get input
0227 F888 81   0D        2        CMPA   #CR             if not cr then..
0228 F88A 26   03        3        BNE    CON2            ..done
0229 F88C 7F   03A1      7 CON1   CLR    ONLINE          set found cr
0230 F88F 39             5 CON2   RTS
0231                       *
0232           0003                SETDP  MONDP
0233                       *
0234                       *               console input routine, gets character from keyboard
0235                       *               buffer or memory, if finds null in memory then returns to
0236                       *               to caller with error $FF
0237                       *
0238 F890 34   10        8 CONIN  PSHS   DP,X
0239 F892 86   03        2        LDA    #MONDP          setup..
0240 F894 1F   8B        6        TFR    A,DP            ..direct page
0241 F896 96   A2        4        LDA    LASTC           saved one?
0242 F898 27   04        3        BEQ    CON5            none saved
0243 F89A 0F   A2        6        CLR    LASTC           not saved anymore
0244 F89C 35   90       10        PULS   DP,X,PC
0245                       *
0246 F89E 9E   75        5 CON5   LDX    LINEPT          see if mem input
0247 F8A0 27   0E        3        BEQ    CON3            no, use buffer
0248 F8A2 A6   80        6        LDA    ,X+             get mem value
0249 F8A4 26   0E        3        BNE    CON4            ok if not null
0250 F8A6 4A             2        DECA                   set to $FF
0251 F8A7 10DE 8D        6 MEND   LDS    MSTACK
0252 F8AA 0F   75        6        CLR    LINEPT          clear mem..
0253 F8AC 0F   76        6        CLR    LINEPT+1        ..input
0254 F8AE 35   88        8        PULS   DP,PC           and return to caller
0255                       *
0256 F8B0 AD   9F 0367   12 CON3   JSR    [CINADR]        get console input
0257 F8B4 9F   75        5 CON4   STX    LINEPT          store new mem pointer
0258 F8B6 35   90       10        PULS   DP,X,PC
0259                       *
0260                       *
0261                       *               this routine prints the registers from the stack as pointed
0262                       *               to by usrstk, then prints usrstk itself.
0263                       *               also prints 5 bytes starting at (pc)
0264                       *
0265 F8B8 8E   FF96      3 EXREG  LDX    #TITLES         headings..
0266 F8BB 8D   2F        7        BSR    STRING          ..printed first
0267 F8BD DE   AA        5        LDU    USRSTK
0268 F8BF C6   04        2        LDB    #4              ie, cc,a,b,dp
0269 F8C1 A6   C0        6 EX1    LDA    ,U+             get data..
0270 F8C3 17   01D1      9        LBSR   OPARSP          ..output as 2 hex digits
0271 F8C6 5A             2        DECB
0272 F8C7 26   F8        3        BNE    EX1             until all 4 output
0273 F8C9 C6   04        2        LDB    #4              ie, x,y,u,pc
0274 F8CB AE   C1        8 EX2    LDX    ,U++            get 2 bytes..
0275 F8CD 17   01A5      9        LBSR   OPXREG          ..as 4 hex digits
0276 F8D0 5A             2        DECB
0277 F8D1 26   F8        3        BNE    EX2             until all output
0278 F8D3 9E   AA        5        LDX    USRST<          then put out stack
0279 F8D5 17   019D      9        LBSR   OPXREG          ..address
0280 F8D8 EE   0A        6        LDU    10,X            get user pc value
0281 F8DA 8E   FFBD      3        LDX    #PCMESS         send..
0282 F8DD 8D   0D        7        BSR    STRING          ..title
0283 F8DF C6   05        2        LDB    #5              then 5 bytes
0284 F8E1 A6   C0        6 EX3    LDA    ,U+
0285 F8E3 17   01B1      9        LBSR   OPARSP
0286 F8E6 5A             2        DECB
0287 F8E7 26   F8        3        BNE    EX3
0288                       *
0289                       *               output the string cr,lf to console
0290                       *
```

```
0291 F8E9 8E   FF93     3 OPCRLF LDX   #SCRLF          string address of cr,lf
0292                     *
0293                     *             output the string pointed to by (x) until a null,
0294                     *             leaves x pointing to null+1, other registers intact
0295                     *
0296 F8EC 34   02       6 STRING PSHS  A
0297 F8EE A6   80       6 STRNG1 LDA   ,X+             get data..
0298 F8F0 27   05       3        BEQ   EOTXT           ..finish if null..
0299 F8F2 17   012C     9        LBSR  CONOUT          ..else output..
0300 F8F5 20   F7       3        BRA   STRNG1          ..and repeat
0301 F8F7 35   82       8 EOTXT  PULS  A,PC            get back a, and return
0302                     *
0303                     *             calculate apparent address of cursor without
0304                     *             allowing for memory wrap around, result in d
0305                     *
0306 F8F9 86   28       2 CCOFST LDA   #COLS
0307 F8FB D6   8F       4        LDB   CROW
0308 F8FD 3D            11       MUL
0309 F8FE DB   90       4        ADDB  CCOL
0310 F900 89   00       2        ADCA  #0
0311 F902 D3   91       6        ADDD  CPAGE
0312 F904 39            5        RTS
0313                     *
0314                     *             calculate real address of cursor in memory space,
0315                     *             result returned in x
0316                     *
0317 F905 34   06       7 CCLOCN PSHS  D
0318 F907 8D   F0       7        BSR   CCOFST          apparent address
0319 F909 84   03       2        ANDA  #%11            wrap around in 1k page
0320 F90B C3   0400     4        ADDD  #PAGE           add start of page
0321 F90E 1F   01       6        TFR   D,X
0322 F910 35   86       9        PULS  D,PC            unsave and return
0323                     *
0324                     *             initialise versatile interface adapter.
0325                     *
0326 F912 CC   EFD0     3 VIAI   LDD   #IKPCR*256+IKIER
0327 F915 8E   0980     3        LDX   #KVIA           point to via
0328 F918 A7   0C       5        STA   KPCR-KVIA,X     peripheral control register
0329 F91A E7   0E       5        STB   KIER-KVIA,X     interrupt enable register
0330 F91C 86   7F       2        LDA   #$7F            printer output (7 bits) plus..
0331 F91E A7   03       5        STA   KDDRA-KVIA,X    ..cassette input on 'a' side
0332 F920 A6   0D       5        LDA   KIFR-KVIA,X     cancel any..
0333 F922 A7   0D       5        STA   KIFR-KVIA,X     ..interrupts present
0334 F924 39            5        RTS
0335                     *
0336                     *             initialise crt controller on vdu card, r or s version
0337                     *
0338 F925 8E   FF0D     3 CRTCI  LDX   #CRTCSV         table for v version
0339 F928 CC   0DAA     3        LDD   #PAGHI+1*256+$AA       test the page register..
0340 F92B FD   0800     6        STD   CRTC            ..(low byte) for read/write..
0341 F92E F1   0801     5        CMPB  CRTC+1          ..ability
0342 F931 27   02       3        BEQ   CRTCI1          if so, then v version..
0343 F933 30   0C       5        LEAX  CRTCRV-CRTCSV,X ..else get r table instead
0344 F935 86   0B       2 CRTCI1 LDA   #11             setup 12 registers
0345 F937 E6   86       5 CRTCI2 LDB   A,X             get data..
0346 F939 FD   0800     6        STD   CRTC            ..and store it in correct register
0347 F93C 4A            2        DECA
0348 F93D 2A   F8       3        BPL   CRTCI2          repeat until all 12 done
0349                     *
0350                     *             reset display to a blank page with cursor
0351                     *             at top left of screen
0352                     *
0353 F93F 0F   8F       6 CLRALL CLR   CROW            row 0
0354 F941 0F   90       6        CLR   CCOL            column 0
0355 F943 0F   91       6        CLR   CPAGE           current page set..
0356 F945 0F   92       6        CLR   CPAGE+1         ..to start of display memory
0357 F947 17   009C     9        LBSR  SETTOP          set page in crtc chip
0358 F94A 17   00A1     9        LBSR  SETCUR          and set cursor in crtc chip
0359                     *
0360                     *             clear the display memory to all blanks
0361                     *
0362 F94D CC   2020     3 CLRSCN LDD   #SPACE*256+SPACE
0363 F950 8E   0400     3        LDX   #PAGE           start of display memory
0364 F953 ED   81       8 CLRS1  STD   ,X++            store two blanks
0365 F955 8C   0800     4        CMPX  #PAGE+1024      and repeat..
0366 F958 26   F9       3        BNE   CLRS1           ..until done all page
0367 F95A 39            5        RTS
0368                     *
0369                     *             this routine used to put a character to the vdu,
0370                     *             handling cr, l.f, backspace, and form feed.
0371                     *             all registers are saved
0372                     *
0373 F95B 34   7F       15 DISPLA PSHS  DP,CC,D,X,Y,U  save registers
0374 F95D C6   03       2        LDB   #MONDP
0375 F95F 1F   9B       6        TFR   B,DP
0376 F961 9E   6F       5        LDX   FUNCTI          get function table
0377 F963 1A   10       3        ORCC  #IRQ            stop interrupts since not re-entrant
0378 F965 8D   3F       7        BSR   DISPCH          jump on function table
0379 F967 35   FF       17       PULS  DP,CC,D,X,Y,U,PC       then restore and return
0380                     *
0381                     *             this routine puts a character on display and moves cursor,
0382                     *
0383 F969 8D   9A       7 SIMCHR BSR   CCLOCN          find location in memory..
0384 F96B A7   84       4        STA   ,X              ..and store character
0385 F96D 0C   90       6        INC   CCOL            move cursor across..
0386 F96F 96   90       4        LDA   CCOL            ..then done if..
```

```
0387 F971 81   28       2          CMPA   #COLS            ..if all columns..
0388 F973 26   0E       3          BNE    SIM1             ..not yet filled
0389 F975 0F   90       6          CLR    CCOL             do cr..
0390                               *
0391                               *                       move cursor down 1 line, scroll display if required
0392                               *
0393 F977 0C   0F       6 DOLF     INC    CROW             down a row
0394 F979 96   0F       4          LDA    CROW
0395 F97B 81   19       2          CMPA   #ROWS            last row yet..
0396 F97D 26   04       3          BNE    SIM1             ..if not then done..
0397 F97F 0A   0F       6          DEC    CROW             ..else back up..
0398 F981 8D   48       7          BSR    SCROLL           ..and scroll instead
0399 F983 20   69       3 SIM1     BRA    SETCUR           then set new cursor
0400                               *
0401                               *                       move cursor back erasing last character
0402                               *
0403 F985 8D   08       7 DORUB    BSR    BSONE            back up cursor..
0404 F987 86   20       2          LDA    #SPACE
0405 F989 8D   D0       7          BSR    DISPLA           ..then blank last character..
0406 F98B 8D   02       7          BSR    BSONE            ..and back up again
0407 F98D 20   5F       3          BRA    SETCUR           then set cursor up
0408                               *
0409                               *                       back cursor up allowing line and row underflow
0410                               *
0411 F98F 0A   90       6 BSONE    DEC    CCOL             left move cursor
0412 F991 2A   0C       3          BPL    BS1              no underflow..
0413 F993 86   27       2          LDA    #COLS-1          ..else set to..
0414 F995 97   90       4          STA    CCOL             ..right margin..
0415 F997 0A   0F       6          DEC    CROW             ..and up one row
0416 F999 2A   04       3          BPL    BS1              no row underflow..
0417 F99B 0F   0F       6          CLR    CROW
0418 F99D 0F   90       6          CLR    CCOL
0419 F99F 39            5 BS1      RTS
0420                               *
0421                               *                       dispatch routine looks up a     racter in a table,
0422                               *                       character in a, table addr      in X.
0423                               *
0424                               *                       table format is- first byte, number of entries 1 to 255
0425                               *
0426                               *                         repeat for each entry- character to match with,
0427                               *                                                2 byte offset from start of this
0428                               *                                                table of routine to jump to if
0429                               *                                                characters match.
0430                               *
0431                               *
0432                               *                           flag byte- control if no match found :-
0433                               *
0434                               *                            positive- next word is offset of default routine
0435                               *                                zero- return to calling program
0436                               *                            negative- next word is address of another table to search
0437                               *
0438 F9A0 EC   84       5 DIS2     LDD    ,X               get address of new table..
0439 F9A2 ED   62       6          STD    2,S              ..replace old on stack
0440 F9A4 35   56      11          PULS   D,X,U            and begin again
0441 F9A6 34   56      11 DISPCH  PSHS   D,X,U            save registers
0442 F9A8 E6   81       7          LDB    ,X++             get length and move to offset
0443 F9AA A1   1F       5 DIS1     CMPA   -1,X             compare characters
0444 F9AC 27   0B       3          BEQ    DIS3             found it, else..
0445 F9AE 30   03       5          LEAX   3,X              ..move down..
0446 F9B0 5A            2          DECB                    ..repeat until..
0447 F9B1 26   F7       3          BNE    DIS1             ..done
0448 F9B3 6D   1F       7          TST    -1,X             test flag byte
0449 F9B5 27   0A       3          BEQ    DIS4             zero means return
0450 F9B7 2B   E7       3          BMI    DIS2             search new table, else..
0451 F9B9 EC   84       5 DIS3     LDD    ,X               get offset and..
0452 F9BB E3   62       7          ADDD   2,S              add start of table, then..
0453 F9BD ED   64       6          STD    4,S              ..store in stack
0454 F9BF 35   96      11          PULS   D,X,PC           restore and go to routine
0455 F9C1 35   D6      13 DIS4     PULS   D,X,U,PC         return to caller
0456                               *
0457                               *                       scroll the display up one line, leave cursor at same
0458                               *                       position on screen, leave registers intact
0459                               *
0460 F9C3 DC   91       5 SCROLL   LDD    CPAGE            get start of page..
0461 F9C5 C3   03E8     4          ADDD   #ROWS*COLS       ..then move off end
0462 F9C8 8E   0400     3          LDX    #PAGE            actual memory address
0463 F9CB CE   2020     3          LDU    #SPACE*256+SPACE        double space
0464 F9CE 108E 0014     4          LDY    #COLS/2          number to blank
0465 F9D2 84   03       2 SCR1     ANDA   #%11             wrap around on 1k
0466 F9D4 EF   8B       9          STU    D,X              and put 2 blanks
0467 F9D6 C3   0002     4          ADDD   #2               move up and..
0468 F9D9 31   3F       5          LEAY   -1,Y             ..repeat until..
0469 F9DB 26   F5       3          BNE    SCR1             ..line done
0470 F9DD 83   03E8     4          SUBD   #ROWS*COLS       move to second line now
0471 F9E0 84   03       2          ANDA   #%11             wrap around and..
0472 F9E2 DD   91       5          STD    CPAGE            ..set as new page
0473 F9E4 8D   08       7          BSR    SETCUR           put cursor back in position
0474                               *
0475                               *                       put page address into crt controller
0476                               *
0477 F9E6 9E   91       5 SETTOP   LDX    CPAGE            get page
0478 F9E8 86   0C       2          LDA    #PAGHI           point to page start register
0479 F9EA 20   09       3          BRA    SETPAR           then enter parameters
0480                               *
0481                               *                       do a carriage return by setting column to 0
0482                               *
0483 F9EC 0F   90       6 DOCR     CLR    CCOL
```

```
0484                        *
0485                        *           set cursor register in crt controller
0486                        *
0487 F9EE 17   FF08     9 SETCUR LBSR   CCOFST          get cursor address, no wrap around
0488 F9F1 1F   01       6        TFR    D,X             into x
0489 F9F3 86   0E       2        LDA    #CURHI          point to cursor register
0490                        *
0491                        *           put 2 byte value into crt controller, value in x,
0492                        *                 high byte register number in a
0493                        *
0494 F9F5 34   10       7 SETPAR PSHS   X
0495 F9F7 35   04       6        PULS   B               get high byte
0496 F9F9 FD   0800     6        STD    CRTC            set register and data
0497 F9FC 4C            2        INCA                   move to low byte
0498 F9FD 35   04       6        PULS   B               get low byte
0499 F9FF FD   0800     6        STD    CRTC            set register and data
0500 FA02 39            5        RTS
0501                        *
0502                        *           this routine puts keyboard input into line buffer
0503                        *           if no room then ignores character, else echoes to display
0504                        *           if echo is switched on. handles rubout unless line buffer
0505                        *           is switched off.
0506                        *
0507 FA03 D6   60       4 HAVCHR LDB    ECHOF
0508 FA05 2A   04       3        BPL    HAV3            buffer off, so no rubout
0509 FA07 81   7F       2        CMPA   #RUBCH
0510 FA09 27   23       3        BEQ    BSP1            go do rubout
0511 FA0B 8D   37       7 HAV3   BSR    PUTCHR          put into buffer
0512 FA0D 27   43       3        BEQ    PUT1            no room, do not echo
0513 FA0F 5D            2        TSTB                   if buffer off, then no..
0514 FA10 2A   08       3        BPL    HAV4            ..line feed on cr input
0515 FA12 81   0D       2        CMPA   #CR
0516 FA14 26   06       3        BNE    KBECHO          not cr, so no..
0517 FA16 8D   04       7        BSR    KBECHO          echo of a line..
0518 FA18 86   0A       2        LDA    #LF             ..feed
0519 FA1A 9F   87       5 HAV4   STX    HEADST          move static pointer up
0520                        *
0521                        *           put character in a to console output if echo on
0522                        *
0523 FA1C D6   60       4 KBECHO LDB    ECHOF           see if echo on
0524 FA1E 58            2        LSLB                   need bit 6
0525 FA1F 2A   31       3        BPL    PUT1            not on if zero
0526                        *
0527                        *           console output routine, sends to printer also
0528                        *           if switched on
0529                        *
0530           0000             SETDP  0
0531 FA21 7D   0361     7 CONOUT TST    PFLAG           if printer off then
0532 FA24 27   04       3        BEQ    CANOP1          value is zero
0533 FA26 AD   9F 036D 12        JSR    [PRINTI]        else call printer
0534 FA2A 6E   9F 0365  8 CANOP1 JMP    [COPADR]        then console output
0535                        *
0536           0003             SETDP  MONDP
0537 FA2E 9E   89       5 BSP1   LDX    HEADDY          end of line, if at..
0538 FA30 9C   87       6        CMPX   HEADST          ..start of line..
0539 FA32 27   1E       3        BEQ    PUT1            ..then nothing to rubout
0540 FA34 8C   03AE     4        CMPX   #BUFFER         do cyclic..
0541 FA37 26   03       3        BNE    BSP2
0542 FA39 30   88 52    5        LEAX   BUFLEN,X        ..decrement of pointer
0543 FA3C 30   1F       5 BSP2   LEAX   -1,X
0544 FA3E 9F   89       5        STX    HEADDY          set new end of line
0545 FA40 96   5F       4        LDA    BSECHO          and echo a backspace..
0546 FA42 20   D8       3        BRA    KBECHO
0547                        *
0548                        *
0549                        *           put character into buffer if room, return z=0,
0550                        *           else return z=1
0551                        *
0552 FA44 9E   89       5 PUTCHR LDX    HEADDY          get pointer
0553 FA46 8D   0B       7        BSR    BUMPU           if no room, then..
0554 FA48 9C   8B       6        CMPX   TAIL            ..pointers equal..
0555 FA4A 27   06       3        BEQ    PUT1            ..so done
0556 FA4C A7   84       4        STA    ,X              store it and..
0557 FA4E 9F   89       5        STX    HEADDY          ..set new pointer
0558 FA50 1C   FB       3        ANDCC  #255-ZERO
0559 FA52 39            5 PUT1   RTS
0560                        *
0561                        *           cyclic increment of buffer pointers
0562                        *
0563 FA53 30   01       5 BUMPU  LEAX   1,X
0564 FA55 8C   0400     4        CMPX   #BUFFER+BUFLEN
0565 FA58 26   03       3        BNE    ANRTS
0566 FA5A 8E   03AE     3        LDX    #BUFFER
0567 FA5D 39            5 ANRTS  RTS
0568                        *
0569                        *           get character from buffer, if none then clears interrupt
0570                        *           mask and waits. all registers saved, including cc
0571                        *
0572           0000             SETDP  0
0573 FA5E 34   11       8 GETCHR PSHS   CC,X             save and..
0574 FA60 20   02       3        BRA    GETCH1          ..skip wait
0575 FA62 3C   EF      20 GETCH2 CWAI   #255-IRQ        wait for an interrupt
0576 FA64 BE   038B     6 GETCH1 LDX    TAIL            get tail pointer
0577 FA67 BC   0387     7        CMPX   HEADST          if equals static head
0578 FA6A 27   F6       3        BEQ    GETCH2          ..then no characters
0579 FA6C 8D   E5       7        BSR    BUMPU           ..else move up
0580 FA6E A6   84       4        LDA    ,X              ..and get it
```

```
0581 FA70 BF    0388     6          STX    TAIL              set new tail
0582 FA73 35    91      10          PULS   CC,X,PC
0583            0003                SETDP  MONDP
0584                               *
0585                               *          output x register as 4 hex digits to console,
0586                               *          all registers saved.
0587                               *
0588 FA75 34    06       7 OPXREG  PSHS   D
0589 FA77 1F    10       6          TFR    X,D
0590 FA79 8D    06       7          BSR    OPAREG
0591 FA7B 1F    98       6          TFR    B,A
0592 FA7D 8D    18       7          BSR    OPARSP
0593 FA7F 35    86       9          PULS   D,PC
0594                               *
0595                               *          output 'a' register as 2 hex digits to console,
0596                               *          all registers saved except a.
0597                               *
0598 FA81 34    02       6 OPAREG  PSHS   A
0599 FA83 44             2          LSRA
0600 FA84 44             2          LSRA
0601 FA85 44             2          LSRA
0602 FA86 44             2          LSRA
0603 FA87 8D    04       7          BSR    HEXOUT            left nibble
0604 FA89 A6    E0       6          LDA    ,S+               get byte back and..
0605 FA8B 84    0F       2          ANDA   #15               ..do right nibble
0606                               *
0607                               *          output a as a single hex digit
0608                               *
0609 FA8D 8B    30       2 HEXOUT  ADDA   #'0
0610 FA8F 81    39       2          CMPA   #'9
0611 FA91 23    02       3          BLS    HEX2
0612 FA93 8B    07       2          ADDA   #'A-'9-1
0613 FA95 20    8A       3 HEX2    BRA    CONOUT            output and return
0614                               *
0615                               *          output a hex digit followed by a space
0616                               *
0617 FA97 8D    E8       7 OPARSP  BSR    OPAREG
0618 FA99 86    20       2          LDA    #SPACE
0619 FA9B 20    F8       3          BRA    HEX2
0620                               *
0621                               *          printer routine, this interfaces to anadex or centronics
0622                               *          parallel interface printers.
0623            0000                SETDP  0
0624                               *
0625 FA9D 34    46       9 PRINT   PSHS   D,U
0626 FA9F CE    0980     3          LDU    #KVIA             point to via
0627 FAA2 81    0362     5          CMPA   PNEW              if spec'd symbol then..
0628 FAA5 27    0E       3          BEQ    PEXIT             do not send
0629 FAA7 E6    4F       5 PWAIT1  LDB    KORA2-KVIA,U      check if busy
0630 FAA9 2B    FC       3          BMI    PWAIT1            if so then wait
0631 FAAB A7    41       5          STA    KORA-KVIA,U       store data
0632 FAAD C6    ED       2          LDB    #IKPCR-PSTRB      then low strobe..
0633 FAAF E7    4C       5          STB    KPCR-KVIA,U
0634 FAB1 C6    EF       2          LDB    #IKPCR            ..and high..
0635 FAB3 E7    4C       5          STB    KPCR-KVIA,U       ..strobe
0636 FAB5 35    C6      11 PEXIT   PULS   D,U,PC
0637            0003                SETDP  MONDP
0638                               *
0639                               *          these tables are the decision tables for the
0640                               *          memory examine and change function.
0641                               *
0642 FAB7       08                 MTABA  FCB    MTABAE-MTABA/3-1        number of entries
0643 FAB8       56                        FCB    'V
0644 FAB9       002C                      FDB    VADDR-MTABA       modify break address and memory
0645 FABB       47                        FCB    'G
0646 FABC       0034                      FDB    GADDR-MTABA       modify go address and memory
0647 FABE       50                        FCB    'P
0648 FABF       003C                      FDB    PADDR-MTABA       modify proceed address and memory
0649 FAC1       52                        FCB    'R
0650 FAC2       0042                      FDB    RADDR-MTABA       modify register locations
0651 FAC4       20                        FCB    SPACE
0652 FAC5       0051                      FDB    SPACEA-MTABA
0653 FAC7       2C                        FCB    COMMA
0654 FAC8       007A                      FDB    COMMAA-MTABA
0655 FACA       3B                        FCB    SEMIC
0656 FACB       008A                      FDB    SEMICA-MTABA
0657 FACD       2D                        FCB    MINUS
0658 FACE       008E                      FDB    MINUSA-MTABA
0659 FAD0       01                        FCB    1
0660 FAD1       0071                      FDB    NOTA-MTABA
0661            FAD3               MTABAE EQU    *
0662                               *
0663 FAD3       04                 MTABB  FCB    MTABBE-MTABB/3-1
0664 FAD4       20                        FCB    SPACE
0665 FAD5       005E                      FDB    SPACEB-MTABB
0666 FAD7       2C                        FCB    COMMA
0667 FAD8       008D                      FDB    COMMAB-MTABB
0668 FADA       3B                        FCB    SEMIC
0669 FADB       0069                      FDB    SEMICB-MTABB
0670 FADD       2D                        FCB    MINUS
0671 FADE       0072                      FDB    MINUSB-MTABB
0672 FAE0       01                        FCB    1
0673 FAE1       007B                      FDB    NOTB-MTABB
0674            FAE3               MTABBE EQU    *
0675                               *
0676                               *          memory examine and change routine
0677                               *
```

```
0678 FAE3 17   0160     9 VADDR  LBSR   BRKOUT          take out any break
0679 FAE6 CE   03A3     3        LDU    #CBREAK         point to break address store
0680 FAE9 20   03       3        BRA    ADDR1
0681 FAEB CE   0395     3 GADDR  LDU    #GSAV           point to go address store
0682 FAEE 10AE C4       6 ADDR1  LDY    ,U              get initial value
0683 FAF1 20   3E       3        BRA    DATA
0684 FAF3 DE   AA       5 PADDR  LDU    USRSTK
0685 FAF5 33   4A       5        LEAU   10,U            point to user pc
0686 FAF7 20   F5       3        BRA    ADDR1
0687 FAF9 109E AA       6 RADDR  LDY    USRSTK          get address off 'cc' register
0688 FAFC CE   03AC     3 RADDR1 LDU    #TEMP           dummy location for new value
0689 FAFF 20   30       3        BRA    DATA
0690                      *
0691 FB01 CE   0393     3 MEM    LDU    #MSAV           point to memory address store
0692 FB04 10AE C4       6        LDY    ,U              get intial value
0693 FB07 5F            2        CLRB                   set status zero
0694 FB08 17   FD74     9 SPACEA LBSR   CONCHR          get input
0695 FB0B 27   0B       3        BEQ    CRA             no address given
0696 FB0D 8E   FAB7     3        LDX    #MTABA          search address..
0697 FB10 16   FE93     5        LBRA   DISPCH          ..table
0698                      *
0699 FB13 5D            2 CRD    TSTB                   if status
0700 FB14 2F   02       3        BLE    CRA             -1 or 0 then no change
0701 FB16 31   21       5        LEAY   1,Y             else up one
0702                      *
0703 FB18 1F   21       6 CRA    TFR    Y,X             print out..
0704 FB1A 17   FF58     9        LBSR   OPXREG          ..address
0705 FB1D A6   84       4        LDA    ,X              then..
0706 FB1F 17   FF75     9        LBSR   OPA?SP          ..data
0707 FB22 C6   01       2        LDB    #1              set status +1
0708 FB24 D7   A1       4        STB    ONLINE          allow next line
0709 FB26 20   09       3        BRA    DATA            and continue
0710                      *
0711 FB28 97   A2       4 NOTA   STA    LASTC           save for re-use
0712 FB2A 8D   40       7        BSR    NUMB            get numberwith..
0713 FB2C 29   38       3        BVS    MERR            ..error if none
0714 FB2E 1F   02       6        TFR    D,Y             is new address
0715 FB30 5F            2 DAT1   CLRB                   set status 0
0716           FB31       COMMAA EQU    *
0717           FB31       SPACEB EQU    *
0718 FB31 17   FD4B     9 DATA   LBSR   CONCHR          get input
0719 FB34 27   DD       3        BEQ    CRD             no data, found cr
0720 FB36 8E   FAD3     3        LDX    #MTABB          else search..
0721 FB39 16   FE6A     5        LBRA   DISPCH          ..data table
0722                      *
0723 FB3C 5D            2 SEMICB TSTB                   test status,
0724 FB3D 2C   02       3        BGE    SEMICA          if -1 then..
0725 FB3F 31   3F       5        LEAY   -1,Y            dec before
0726 FB41 10AF C4       6 SEMICA STY    ,U              save new address
0727 FB44 39            5        RTS                    and exit
0728                      *
0729           FB45       MINUSA EQU    *
0730 FB45 31   3F       5 MINUSB LEAY   -1,Y            back down 1,
0731 FB47 5D            2        TSTB                   but if status..
0732 FB48 2C   02       3        BGE    MIND2           ..is -1 then..
0733 FB4A 31   3F       5        LEAY   -1,Y            ..back down 2
0734 FB4C 20   E2       3 MIND2  BRA    DAT1            then continue
0735                      *
0736 FB4E 97   A2       4 NOTB   STA    LASTC           save for re-use
0737 FB50 8D   1A       7        BSR    NUMB            in number
0738 FB52 29   12       3        BVS    MERR            should have number
0739 FB54 E7   A4       4        STB    ,Y              store data then..
0740 FB56 E1   A4       4        CMPB   ,Y              ..check it
0741 FB58 27   06       3        BEQ    COMMAB          is ok,
0742 FB5A 8E   FF8F     3        LDX    #MQRY           else tell..
0743 FB5D 17   FD8C     9        LBSR   STRING          ..user
0744 FB60 31   21       5 COMMAB LEAY   1,Y             go up 1
0745 FB62 C6   FF       2        LDB    #$FF            with status -1
0746 FB64 20   CB       3        BRA    DATA            then continue
0747                      *
0748 FB66 17   FD16     9 MERR   LBSR   CONCHR          get wrong symbol
0749 FB69 16   011D     5        LBRA   BADCMD          and tell user
0750                      *
0751                      *              get hex number from input stream, allow leading spaces,
0752                      *              and stop on first non-hex character. return number in d,
0753                      *              with v=0, if no number then d=0 and v=1.
0754                      *
0755           0000       SETDP  0
0756                      *
0757 FB6C 4F            2 NUMB   CLRA
0758 FB6D 5F            2        CLRB
0759 FB6E 34   06       7        PSHS   D               put initial zero value
0760 FB70 8D   18       7        BSR    GETHXS          get first non-blank as hex value
0761 FB72 29   14       3        BVS    NUMB1           wasn't hex
0762 FB74 C6   04       2 NUMB3  LDB    #4
0763 FB76 48            2        ASLA                   move to..
0764 FB77 48            2        ASLA
0765 FB78 48            2        ASLA
0766 FB79 48            2        ASLA
0767 FB7A 48            2 NUMB2  ASLA                   ..high nibble
0768 FB7B 69   61       7        ROL    1,S             rotate into..
0769 FB7D 69   60       7        ROL    0,S             ..value
0770 FB7F 5A            2        DECB
0771 FB80 26   F8       3        BNE    NUMB2           do 4 times
0772 FB82 8D   11       7        BSR    GETHEX          get a hex digit from console
0773 FB84 28   EE       3        BVC    NUMB3           was hex so use
0774 FB86 1C   FD       3 NUMB4  CLV                    else finish with v clear
```

```
0775 FB88 35   86       9 NUMB1  PULS   D,PC
0776                      *
0777                      *              gethxs - get a hex digit ignoring leading spaces
0778                      *              gethex - get a hex digit
0779                      *              both return value in a, with v=0, else set v=1 if non-hex
0780                      *
0781 FB8A 17   FCF2     9 GETHXS LBSR   CONCHR          get input
0782 FB8D 27   22       3        BEQ    GETH5           on cr, no number
0783 FB8F 81   20       2        CMPA   #SPACE          if space..
0784 FB91 27   F7       3        BEQ    GETHXS          ..then ignore
0785 FB93 20   05       3        BRA    GETH2           change to hex
0786                      *
0787 FB95 17   FCE7     9 GETHEX LBSR   CONCHR          get input
0788 FB98 27   17       3        BEQ    GETH5           on cr, no number
0789 FB9A 81   30       2 GETH2  CMPA   #'0
0790 FB9C 25   0C       3        BLO    GETH1           illegal hex
0791 FB9E 81   39       2        CMPA   #'9
0792 FBA0 23   14       3        BLS    GETH3           number hex
0793 FBA2 81   41       2        CMPA   #'A
0794 FBA4 25   04       3        BLO    GETH1           illegal hex
0795 FBA6 81   46       2        CMPA   #'F
0796 FBA8 23   0A       3        BLS    GETH4           alpha hex
0797 FBAA 81   2C       2 GETH1  CMPA   #COMMA
0798 FBAC 27   03       3        BEQ    GETH5           absorb comma
0799 FBAE B7   03A2     5        STA    LASTC           else re-use
0800 FBB1 1A   02       3 GETH5  SEV                    bad hex
0801 FBB3 39            5        RTS
0802                      *
0803 FBB4 80   07       2 GETH4  SUBA   #7              alpha offset
0804 FBB6 80   30       2 GETH3  SUBA   #'0             number offset
0805 FBB8 39            5        RTS                    with v clear
0806                      *
0807           0003             SETDP  MONDP
0808                      *
0809                      *              resume user program using stack as stands
0810                      *
0811 FBB9 32   62       5 RESUME LEAS   2,S             strip return address
0812 FBBB 8D   AF       7        BSR    NUMB            get number or zero
0813 FBBD DD   A5       5        STD    NBREAK          and set break ignore count
0814 FBBF 8D   6C       7 RES2   BSR    BRKIN           insert breakpoints
0815 FBC1 35   FF      17        PULS   CC,A,B,DP,X,Y,U,PC      and pull all user registers off stack
0816                      *
0817                      *              software interrupt handler, come here on breakpoint
0818                      *              either stops and displays registers or traces past
0819                      *              breakpoint and resumes.
0820                      *
0821 FBC3 86   03       2 SWIHAN LDA    #MONDP          setup..
0822 FBC5 1F   8B       6        TFR    A,DP            ..direct page
0823           0003             SETDP  MONDP           tell assembler
0824 FBC7 AE   6A       6        LDX    10,S            back up..
0825 FBC9 30   1F       5        LEAX   -1,X            ..user..
0826 FBCB AF   6A       6        STX    10,S            ..program counter
0827 FBCD 8D   77       7        BSR    BRKOUT          remove breakpoint
0828 FBCF 9E   A5       5        LDX    NBREAK          get count
0829 FBD1 27   08       3        BEQ    RES1            stop if zero, else..
0830 FBD3 30   1F       5        LEAX   -1,X            ..decrement,..
0831 FBD5 9F   A5       5        STX    NBREAK          ..and restore
0832 FBD7 8D   76       7        BSR    TUSER1          trace past break..
0833 FBD9 20   E4       3        BRA    RES2            ..then resume again
0834 FBDB 10DF AA       6 RES1   STS    USRSTK
0835 FBDE 17   FCD7     9        LBSR   EXREG           display registers
0836 FBE1 20   19       3        BRA    BACK1           and stay in monitor
0837                      *
0838                      *              change number in x if one given in input stream,
0839                      *              destroys d
0840                      *
0841 FBE3 8D   87       7 NUMBX  BSR    NUMB
0842 FBE5 29   02       3        BVS    NUMBX1          no number
0843 FBE7 1F   01       6        TFR    D,X
0844 FBE9 39            5 NUMBX1 RTS
0845                      *
0846                      *              user program returns here if rts done
0847                      *
0848 FBEA 32   7E       5 BACK   LEAS   -2,S             make room for new return address
0849 FBEC 34   FF      17        PSHS   CC,A,B,DP,X,Y,U,PC      and save all user registers
0850 FBEE 8E   FBEA     3        LDX    #BACK           set return address..
0851 FBF1 AF   6C       6        STX    12,S            ..again
0852 FBF3 C6   03       2        LDB    #MONDP
0853 FBF5 1F   9B       6        TFR    B,DP
0854 FBF7 8D   4D       7        BSR    BRKOUT          remove breakpoints
0855 FBF9 10DF AA       6        STS    USRSTK          save user stack pointer
0856 FBFC 1C   EF       3 BACK1  CLI
0857 FBFE 16   FC59     5        LBRA   PARSE           and resume monitor functions
0858                      *
0859                      *              go to user program, optional address specified
0860                      *              the stack pointer is reset, but the register contents
0861                      *              are maintained as listed by the e command.
0862                      *
0863 FC01 0F   A5       6 GOUSER CLR    NBREAK          put zero in..
0864 FC03 0F   A6       6        CLR    NBREAK+1        ..break count
0865 FC05 DE   5B       5        LDU    STACK           get pointer
0866 FC07 8E   FBEA     3        LDX    #BACK           return address..
0867 FC0A AF   C3       8        STX    ,--U            ..pushed first
0868 FC0C C6   0D       2        LDB    #11+2           12 registers plus return
0869 FC0E A6   E5       5 GO1    LDA    B,S             get value and..
0870 FC10 A7   C2       6        STA    ,-U             ..push it
```

```
0871 FC12 5A            2          DECB
0872 FC13 2A  F9        3          BPL    GO1              for all registers
0873 FC15 32  42        5          LEAS   2,U              new stack, ignore return address
0874 FC17 9E  95        5          LDX    GSAV             saved address
0875 FC19 8D  C8        7          BSR    NUMBX            change if given..
0876 FC1B 9F  95        5          STX    GSAV             ..and restore
0877 FC1D AF  6A        6          STX    10,S             put as user pc
0878 FC1F 8D  0C        7          BSR    BRKIN            insert breakpoint
0879 FC21 35  FF       17          PULS   CC,A,B,DP,X,Y,U,PC       and begin user program
0880                               *
0881                               *                       change user breakpoint position
0882                               *
0883 FC23 8D  21        7 BRKSET   BSR    BRKOUT           ensure old is out
0884 FC25 8E  FFFF      3          LDX    #$FFFF           value for no break
0885 FC28 8D  B9        7          BSR    NUMBX            change if given
0886 FC2A 9F  A3        5          STX    CBREAK           and save
0887 FC2C 39            5          RTS
0888                               *
0889                               *                       put breakpoint in if one exists
0890                               *
0891 FC2D 8D  09        7 BRKIN    BSR    BRKTST           see if exists
0892 FC2F 27  06        3          BEQ    BRK1             already a swi, so done
0893 FC31 97  A7        4          STA    CINST            else save it
0894 FC33 86  3F        2          LDA    #SWI
0895 FC35 A7  84        4          STA    ,X               and insert a swi instead
0896 FC37 39            5 BRK1     RTS
0897                               *
0898                               *                       check break required, if not does rts twice
0899                               *
0900 FC38 9E  A3        5 BRKTST   LDX    CBREAK           get address
0901 FC3A 8C  FFFF      4          CMPX   #$FFFF           $FFFF means none
0902 FC3D 27  05        3          BEQ    BRK10            not one
0903 FC3F A6  84        4          LDA    ,X               get instruction
0904 FC41 81  3F        2          CMPA   #SWI             see if swi
0905 FC43 39            5          RTS                     then exit
0906 FC44 35  90        9 BRK10    PULS   X,PC             exit twice
0907                               *
0908                               *                       remove a breakpoint if one present in code
0909                               *
0910 FC46 8D  F0        7 BRKOUT   BSR    BRKTST           see if exists
0911 FC48 26  04        3          BNE    BRK2             not swi,leave alone
0912 FC4A 96  A7        4          LDA    CINST            get saved instruction
0913 FC4C A7  84        4          STA    ,X               and restore in code
0914 FC4E 39            5 BRK2     RTS
0915                               *
0916                               *                       trace one instruction of user code
0917                               *
0918 FC4F 35  10        7 TUSER1   PULS   X                get return address..
0919 FC51 9F  73        5          STX    IRQRTS           and save it
0920 FC53 86  EF        2          LDA    #255-IRQ         clear irq mask..
0921 FC55 A4  E4        4          ANDA   ,S               ..in user..
0922 FC57 A7  E4        4          STA    ,S               ..condition codes
0923 FC59 CC  0F00      3          LDD    #INTDEL          delay before interrupt
0924 FC5C FD  0984      6          STD    KT1CL            set timer going..
0925 FC5F 35  FF       17          PULS   CC,A,B,DP,X,Y,U,PC      ..start user program running
0926                               *
0927                               *                       set number of instructions to trace on each command
0928                               *
0929 FC61 17  FF98      9 TRACEN   LBSR   NUMB             get 0 if no number
0930 FC64 DD  5D        5          STD    NTRACE           save result
0931 FC66 39            5 TRACE1   RTS
0932                               *
0933                               *                       trace required number of instructions then display
0934                               *                       register contents and halt user
0935                               *
0936 FC67 9E  5D        5 TRACE    LDX    NTRACE           get number to trace
0937 FC69 27  FB        3          BEQ    TRACE1           ignore command if zero
0938 FC6B 32  62        5          LEAS   2,S              strip return address
0939 FC6D 9F  A8        5 TRACE2   STX    CTRACE           save number left
0940 FC6F 8D  DE        7          BSR    TUSER1           trace one instruction
0941 FC71 9E  A8        5          LDX    CTRACE           get number left..
0942 FC73 30  1F        5          LEAX   -1,X             ..and decrement
0943 FC75 26  F6        3          BNE    TRACE2           repeat if required
0944 FC77 16  FF61      5          LBRA   RES1             else show registers and halt user
0945                               *
0946                               *                       turn printer echo of console output on or off
0947                               *
0948 FC7A 17  FC02      9 PCNTL    LBSR   CONCHR           get input
0949 FC7D 27  06        3          BEQ    POFF             if cr then off
0950 FC7F 81  2B        2          CMPA   #'+              if plus..
0951 FC81 27  03        3          BEQ    PON              ..then switch on
0952 FC83 97  A2        4          STA    LASTC            re-use if not +
0953 FC85 4F            2 POFF     CLRA                    switch off value
0954 FC86 97  61        4 PON      STA    PFLAG            set flag
0955 FC88 39            5          RTS
0956                               *
0957                               *                       if get bad command, query it and ignore rest of line
0958                               *
0959 FC89 9E  75        5 BADCMD   LDX    LINEPT           if memory input..
0960 FC8B 1026 FC18     6          LBNE   MEND             ..then exit
0961 FC8F 8E  FFC3      3          LDX    #CQRY            output query..
0962 FC92 17  FC57      9          LBSR   STRING           ..message
0963 FC95 17  FD89      9          LBSR   CONOUT           and character
0964 FC98 17  FC4E      9          LBSR   OPCRLF           followed by cr,lf
0965 FC9B 17  FBE1      9 BAD1     LBSR   CONCHR           get input and..
0966 FC9E 26  FB        3          BNE    BAD1             if not cr then ignore it
0967 FCA0 39            5          RTS                     then carry on
```

```
0968                            *
0969                            *               cassette file load routine, this searches for named file
0970                            *               followed by data.
0971                            *
0972 FCA1 17   00D1       9 LOAD    LBSR    NAMEIN          get file name
0973 FCA4 17   FEC5       9         LBSR    NUMB            get offset
0974 FCA7 DD   85         5         STD     OFFSET          and save
0975 FCA9 8E   0397       3 LOAD2   LDX     #NAME
0976 FCAC 86   30         2 LOAD4   LDA     #'0             get name file..
0977 FCAE 8D   4E         7         BSR     GETHDR          ..header
0978 FCB0 26   FA         3         BNE     LOAD4           ignore others
0979 FCB2 C6   06         2         LDB     #6              name length
0980 FCB4 D7   AC         4         STB     TEMP            save it
0981 FCB6 8D   66         7 LOAD3   BSR     CBIN1           get input character
0982 FCB8 E6   80         6         LDB     ,X+             and name character
0983 FCBA C1   3F         2         CMPB    #'?             if wildcard..
0984 FCBC 27   04         3         BEQ     LOAD1           ..then matches
0985 FCBE A1   1F         5         CMPA    -1,X            else compare
0986 FCC0 26   E7         3         BNE     LOAD2           wrong name
0987 FCC2 0A   AC         6 LOAD1   DEC     TEMP            count name length
0988 FCC4 26   F0         3         BNE     LOAD3           repeat for all 6
0989 FCC6 8D   56         7         BSR     CBIN1           ignore checksum byte
0990                            *
0991                            *               enter here for match without file name check
0992                            *
0993 FCC8 86   31         2 LOAD7   LDA     #'1             get data..
0994 FCCA 8D   32         7         BSR     GETHDR          ..header
0995 FCCC 26   24         3         BNE     LOAD5           wrong header
0996 FCCE 8D   43         7         BSR     CBIN2           get start address
0997 FCD0 1F   01         6         TFR     D,X             and save while..
0998 FCD2 8D   3F         7         BSR     CBIN2           ..getting end address
0999 FCD4 34   06         7         PSHS    D               put end on stack
1000 FCD6 8D   46         7 LOAD6   BSR     CBIN1           get data item..
1001 FCD8 A7   80         6         STA     ,X+             ..and store it
1002 FCDA AC   E4         6         CMPX    ,S              if not done..
1003 FCDC 23   F8         3         BLS     LOAD6           ..then repeat
1004 FCDE 8D   3E         7         BSR     CBIN1           get checksum byte
1005 FCE0 35   10         7         PULS    X               get old end address
1006 FCE2 17   FD90       9         LBSR    OPXREG          show address so far
1007 FCE5 1F   30         6         TFR     U,D             check summed in u
1008 FCE7 53              2         COMB                    lower byte only
1009 FCE8 27   DE         3         BEQ     LOAD7           if ok, then repeat
1010 FCEA 8E   FFCD       3         LDX     #LQRY           else message..
1011 FCED 17   FBFC       9         LBSR    STRING          ..output
1012 FCF0 20   04         3         BRA     KEYON
1013                            *
1014 FCF2 81   39         2 LOAD5   CMPA    #'9             if not 'x9' then..
1015 FCF4 26   02         3         BNE     LOAD7           ..ignore
1016 FCF6 86   90         2 KEYON   LDA     #CB1FLG+$80     turn on..
1017 FCF8 B7   098E       5         STA     KIER            ..keyboard
1018 FCFB 16   FBEB       5         LBRA    OPCR_F          then exit
1019                            *
1020                            *               get a header from the tape, if the expected one
1021                            *               then set a zero status, else return non-zero status
1022                            *               initialises the checksum in u to 0
1023                            *
1024 FCFE 34   02         6 GETHDR  PSHS    A               save character
1025 FD00 86   10         2         LDA     #CB1FLG         turn off..
1026 FD02 B7   098E       5         STA     KIER            ..keyboard
1027 FD05 8D   17         7 GETHD1  BSR     CBIN1           get from tape..
1028 FD07 81   D8         2         CMPA    #'X+$80         ..and if not 'x' then..
1029 FD09 26   FA         3         BNE     GETHD1          ..try again
1030 FD0B 8D   11         7         BSR     CBIN1           get next character..
1031 FD0D CE   0000       3         LDU     #0              setup checksum
1032 FD10 A1   E0         6         CMPA    ,S+             and compare with required
1033 FD12 39              5         RTS                     then return
1034                            *
1035                            *               get 2 bytes and form a 16 bit value in d
1036                            *               add offset since is address
1037                            *
1038 FD13 8D   09         7 CBIN2   BSR     CBIN1           get 1 byte..
1039 FD15 1F   89         6         TFR     A,B             ..and save while..
1040 FD17 8D   05         7         BSR     CBIN1           ..get second
1041 FD19 1E   89         7         EXG     A,B             wrong order, swap over
1042 FD1B D3   85         6         ADDD    OFFSET          move by offset
1043 FD1D 39              5         RTS
1044                            *
1045                            *               get 1 byte from tape, modifying checksum to suit
1046                            *
1047 FD1E AD   9F 036B   12 CBIN1   JSR     [CASINA]        get byte then..
1048 FD22 33   C6         5         LEAU    A,U             ..add to checksum
1049 FD24 39              5         RTS
1050                            *
1051                            *               software asynchronous transmitter, outputs value in a
1052                            *               as start bit, 8 data bits, 2 stop bits, at rate controlled
1053                            *               by count in delcnt
1054                            *               saves all registers
1055                            *
1056 FD25 34   17        10 MCASOP  PSHS    CC,D,X
1057 FD27 1A   50         3         SEIF                    keep timing
1058 FD29 C6   0B         2         LDB     #11             total length
1059 FD2B 34   04         6         PSHS    B               save on stack
1060 FD2D C6   CF         2         LDB     #IKPCR-COPBIT   low start bit
1061 FD2F 43              2         COMA                    want data inverted
1062 FD30 12              2 MCASO1  NOP                     get timing..
1063 FD31 20   00         3         BRA     **2             ..constant
```

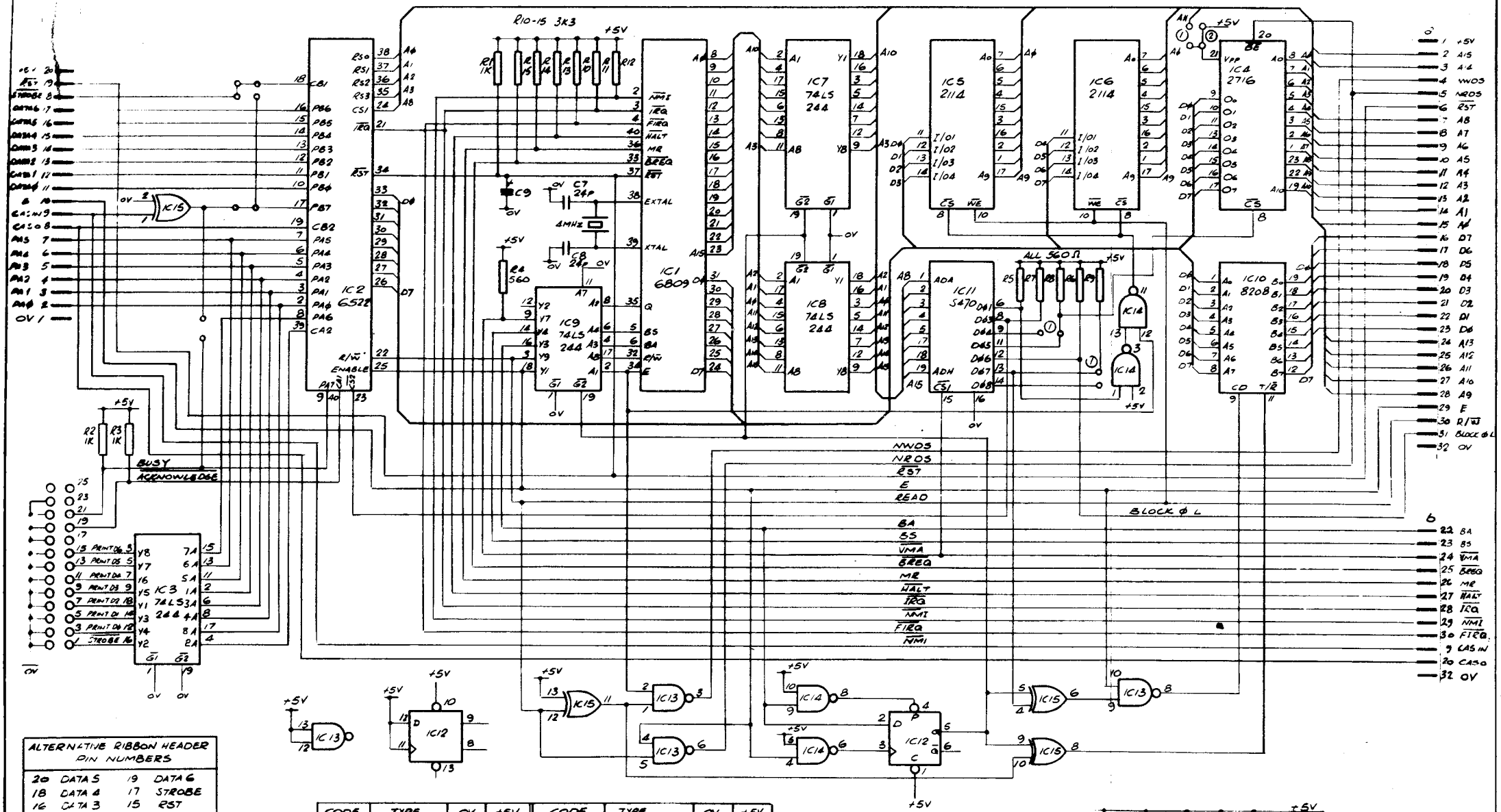
```
1064 FD33 6A    E4       6 MCAS02 DEC    ,S              bits counter
1065 FD35 2B    0E       3        BMI    MCAS03          all done
1066 FD37 F7    0080     5        STB    KPCR            put out bit
1067 FD3A 8D    0D       7        BSR    CWAIT           wait 1 bit time
1068 FD3C C4    DF       2        ANDB   #255-COPBIT     assume next is zero
1069 FD3E 44            2        LSRA                   get next bit
1070 FD3F 25    EF       3        BCS    MCAS01          do want zero..
1071 FD41 CA    20       2        ORB    #COPBIT         ..else set bit
1072 FD43 20    EE       3        BRA    MCAS02          and loop round
1073 FD45 32    61       5 MCAS03 LEAS   1,S             remove counter
1074 FD47 35    97      12        PULS   CC,D,X,PC       and return
1075                       *
1076                       *              cwait waits for 1 bit time, destroys x
1077                       *              hwait waits 1/2 bit time, also destroys x
1078                       *
1079             0000              SETDP  0
1080 FD49 8D    00       7 CWAIT  BSR    HWAIT           do first half
1081 FD4B BE    0363     6 HWAIT  LDX    DELCNT          get count required
1082 FD4E 30    1F       5 HW1    LEAX   -1,X            decrement..
1083 FD50 26    FC       3        BNE    HW1             ..while non-zero
1084 FD52 39             5        RTS
1085             0003              SETDP  MONDP
1086                       *
1087                       *              software asynchronous receiver, gets value into a
1088                       *              saves all other registers, only gets 1 stop bit
1089                       *
1090 FD53 34    14       8 MCASIN PSHS   B,X
1091 FD55 86    80       2        LDA    #$80            rotating counter
1092 FD57 F6    0080     5 MCASI1 LDB    KIRB            wait for..
1093 FD5A 2B    FB       3        BMI    MCASI1          ..start bit
1094 FD5C 8D    ED       7        BSR    HWAIT           wait 1/2 bit time
1095 FD5E F6    0080     5        LDB    KIRB            recheck..
1096 FD61 2B    F4       3        BMI    MCASI1          ..start bit
1097 FD63 8D    E4       7 MCASI2 BSR    CWAIT           wait whole bit time
1098 FD65 F6    0080     5        LDB    KIRB            and get input
1099 FD68 AC    E4       6        CMPX   ,S              waste time to..
1100 FD6A 11AC  E4       7        CMPS   ,S              ..match loop delays
1101 FD6D 58             2        LSLB                   move bit to carry..
1102 FD6E 46             2        RORA                   ..then into byte
1103 FD6F 24    F2       3        BCC    MCASI2          repeat for 8 bits
1104 FD71 8D    D6       7        BSR    CWAIT           get into stop bit
1105 FD73 35    94      10        PULS   B,X,PC          and done
1106                       *
1107                       *              routine gets name from input stream, up to 6
1108                       *              characters long. no name leaves memory unaltered.
1109                       *              any name is padded to 6 characters with spaces.
1110                       *
1111 FD75 8E    039D     3 NAMEIN LDX    #NAME+6
1112 FD78 17    FB04     9 NAM2   LBSR   CONCHR          get a character
1113 FD7B 27    23       3        BEQ    NAM1            ..no name
1114 FD7D 81    20       2        CMPA   #SPACE          if space..
1115 FD7F 27    F7       3        BEQ    NAM2            ..ignore
1116 FD81 81    2C       2        CMPA   #COMMA
1117 FD83 27    1B       3        BEQ    NAM1            null name
1118 FD85 C6    FA       2        LDB    #256-6          minus name length
1119 FD87 A7    85       5 NAM3   STA    B,X             store a letter
1120 FD89 5C             2        INCB                   and move up
1121 FD8A 27    15       3        BEQ    NAM6            done 6 chars, exit
1122 FD8C 17    FAF0     9        LBSR   CONCHR          get next letter
1123 FD8F 27    08       3        BEQ    NAM5            on cr, pad name
1124 FD91 81    20       2        CMPA   #SPACE          on space..
1125 FD93 27    06       3        BEQ    NAM4            ..pad name
1126 FD95 81    2C       2        CMPA   #COMMA          if not comma..
1127 FD97 26    EE       3        BNE    NAM3            ..then use
1128 FD99 86    20       2 NAM5   LDA    #SPACE          padding
1129 FD9B A7    85       5 NAM4   STA    B,X             pad until..
1130 FD9D 5C             2        INCB                   ..end of..
1131 FD9E 26    FB       3        BNE    NAM4            ..name buffer
1132 FDA0 39             5 NAM1   RTS
1133 FDA1 17    FADB     9 NAM6   LBSR   CONCHR          get next input
1134 FDA4 27    FA       3        BEQ    NAM1            leave if cr
1135 FDA6 81    2C       2        CMPA   #COMMA          else
1136 FDA8 27    F6       3        BEQ    NAM1            absorb comma
1137 FDAA 97    A2       4        STA    LASTC           else re-use
1138 FDAC 39             5        RTS
1139                       *
1140                       *              save files on cassette, dumps name block, data blocks
1141                       *              as required in 256 byte blocks maximum, then end file block
1142                       *
1143                       *              can also inhibit end of file block
1144                       *
1145 FDAD 9E    9D       5 SAVE   LDX    CSSTRT          modify start address..
1146 FDAF 17    FE31     9        LBSR   NUMBX           ..if..
1147 FDB2 9F    9D       5        STX    CSSTRT          ..required
1148 FDB4 9E    9F       5        LDX    CSEND           modify end address..
1149 FDB6 17    FE2A     9        LBSR   NUMBX           ..if..
1150 FDB9 9F    9F       5        STX    CSEND           ..required
1151 FDBB 8D    B8       7        BSR    NAMEIN          and get name
1152 FDBD C6    30       2        LDB    #'0             output name..
1153 FDBF 8D    3C       7        BSR    XHEAD           ..header
1154 FDC1 C6    06       2        LDB    #6              name length
1155 FDC3 30    1A       5        LEAX   -6,X            point to name
1156 FDC5 8D    46       7        BSR    DATOUT          output name..
1157 FDC7 8D    4E       7        BSR    CHKOUT          ..then checksum
1158 FDC9 9E    9D       5        LDX    CSSTRT          get start address
1159 FDCB 34    10       7 SAV6   PSHS   X               save start
1160 FDCD DC    9F       5        LDD    CSEND           and get end address
```

```
1161 FDCF A3   E1        9        SUBD   ,S++              form length needed
1162 FDD1 25   1D        3        BLO    SAV2              done all output
1163 FDD3 4D             2        TSTA                     if <=256 then..
1164 FDD4 27   03        3        BEQ    SAV5              ..leave alone..
1165 FDD6 CC   00FF      3        LDD    #255              ..else set to 256
1166 FDD9 33   8B        8 SAV5   LEAU   D,X               form end of block
1167 FDDB 34   56       11        PSHS   D,X,U             put start/end on stack
1168 FDDD C6   31        2        LDB    #'1               put a data..
1169 FDDF 8D   1C        7        BSR    XHEAD             ..header
1170 FDE1 30   62        5        LEAX   2,S               point to start/end
1171 FDE3 C6   04        2        LDB    #4                two words
1172 FDE5 8D   26        7        BSR    DATOUT            and put start/end out
1173 FDE7 35   56       11        PULS   D,X,U             get all back
1174 FDE9 5C             2        INCB                     modify length
1175 FDEA 8D   21        7        BSR    DATOUT            and send data bytes
1176 FDEC 8D   29        7        BSR    CHKOUT            then checksum
1177 FDEE 20   DB        3        BRA    SAV6              and repeat
1178                       *
1179 FDF0 17   FA8C      9 SAV2   LBSR   CONCHR            get input
1180 FDF3 27   06        3        BEQ    SAV3              on cr send eof block
1181 FDF5 81   2D        2        CMPA   #'-               if eof inhibit..
1182 FDF7 27   1D        3        BEQ    RTS1              ..then skip x9
1183 FDF9 97   A2        4        STA    LASTC             re-use input
1184 FDFB C6   39        2 SAV3   LDB    #'9               send eof..
1185                       *
1186                       *                 routine to send header to block, header type in b.
1187                       *                 also initialises checksum in y,
1188                       *
1189 FDFD 108E 0000      4 XHEAD  LDY    #0
1190 FE01 31   3F        5 XH1    LEAY   -1,Y              loop..
1191 FE03 26   FC        3        BNE    XH1               ..delay
1192 FE05 86   D8        2        LDA    #'X+$80           send an x to..
1193 FE07 8D   13        7        BSR    CASOPI            ..cassette
1194 FE09 1F   98        6        TFR    B,A               get type..
1195 FE0B 20   0F        3        BRA    CASOPI            ..and send
1196                       *
1197                       *                 data output routine, sends b data bytes starting from x
1198                       *                 (b zero means 256 bytes), x moves up by b bytes
1199                       *
1200 FE0D A6   80        6 DATOUT LDA    ,X+               get data
1201 FE0F 31   A6        5        LEAY   A,Y               modify checksum
1202 FE11 8D   09        7        BSR    CASOPI            send data
1203 FE13 5A             2        DECB                     repeat until..
1204 FE14 26   F7        3        BNE    DATOUT            ..zero count
1205 FE16 39             5 RTS1   RTS
1206                       *
1207                       *                 send checksum to tape from y, lower byte only
1208                       *
1209 FE17 1F   20        6 CHKOUT TFR    Y,D               checksum to d
1210 FE19 1F   98        6        TFR    B,A               then get low byte
1211 FE1B 43             2        COMA                     want result $FF
1212 FE1C 6E   9F 0369   8 CASOPI JMP    [CASOPA]          send check byte
1213                       *
1214                       *                 drive commands to perform a boot operation
1215                       *
1216                       *                 drive parameter specification
1217                       *
1218 FE20      35040006  DISCIT FCB    $35,4,$0D,$14,$05,$AA     for Shugart drive
               18E5
1219                       *
1220                       *                 drive bad tracks
1221                       *
1222 FE26      350410FF          FCB    $35,4,DRIVE/$80+8+$10,$FF,$FF,$FF
               FFFF
1223                       *
1224                       *                 mode register setup
1225                       *
1226 FE2C      7A0217C1          FCB    $3A-DRIVE,2,$17,$C1
1227                       *
1228                       *                 load head onto disc, starts motor
1229                       *
1230 FE30      7A022368          FCB    $3A+DRIVE,2,$23,$28+DRIVE
1231                       *
1232                       *                 query drive ready
1233                       *
1234 FE34      6C00              FCB    $2C+DRIVE,0
1235                       *
1236                       *                 seek to track 0
1237                       *
1238 FE36      690100            FCB    $29+DRIVE,1,$00
1239                       *
1240                       *                 query drive ready
1241                       *
1242 FE39      6C00              FCB    $2C+DRIVE,0
1243                       *
1244                       *                 read sector 2
1245                       *
1246 FE3B      53030002          FCB    $13+DRIVE,3,$00,$02,$21
               21
1247                       *
1248                       *                 read starting at sector 3
1249                       *
1250 FE40      53020003          FCB    $13+DRIVE,2,$00,$03
1251                       *
1252                       *                 this routine bootstraps from a mini-floppy disc
1253                       *                 reads sector 2 to find where to put program
```

```
1254                            *
1255 FE44 8E   FF04      3 BOOT   LDX   #ANRTI          set dummy..
1256 FE47 9F   83        5        STX   INMI            ..interrupt routine
1257 FE49 86   0A        2        LDA   #FLOPY/256      set direct page..
1258 FE4B 1F   8B        6        TFR   A,DP            ..to floppy controller
1259           000A               SETDP FLOPY/256       and tell assembler
1260 FE4D 8E   FE20      3        LDX   #DISKIT         point to tables
1261 FE50 8D   4D        7        BSR   CMDPAR          drive parameters
1262 FE52 8D   4B        7        BSR   CMDPAR          bad tracks
1263 FE54 8D   49        7        BSR   CMDPAR          mode register
1264 FE56 8D   47        7        BSR   CMDPAR          drive on
1265 FE58 8D   5C        7        BSR   DRVRDY          check ready
1266 FE5A 8D   43        7        BSR   CMDPAR          seek track 0
1267 FE5C 8D   58        7        BSR   DRVRDY          check ready
1268 FE5E 8D   3F        7        BSR   CMDPAR          read sector 2
1269 FE60 CE   0000      3        LDU   #0              put at 0
1270 FE63 1F   32        6        TFR   U,Y             and point to it
1271 FE65 8D   62        7        BSR   TRNSFR          move disc to memory
1272 FE67 26   2A        3        BNE   DERR            non-zero means error
1273 FE69 CC   FF42      3        LDD   #$FF42         error $FF in case
1274 FE6C 10A3 A1       10        CMPD  ,Y++            if not $FF42..
1275 FE6F 26   22        3        BNE   DERR            ..then error, no boot present
1276 FE71 8D   2C        7        BSR   CMDPAR          start read at sector 3
1277 FE73 EE   A1        8        LDU   ,Y++            get address to put at
1278 FE75 A6   A0        6        LDA   ,Y+             and number of sectors
1279 FE77 10AE A4        6        LDY   ,Y              start of program
1280 FE7A 8B   20        2        ADDA  #$20            add sector length value
1281 FE7C D6   00        4 BOOT1  LDB   FDCS            get fdc status and..
1282 FE7E C5   20        2        BITB  #$20            ..if parameter register full..
1283 FE80 26   FA        3        BNE   BOOT1           ..then wait
1284 FE82 97   01        4        STA   FDCP            send number of sectors
1285 FE84 8D   43        7        BSR   TRNSFR          move data to memory
1286 FE86 26   0B        3        BNE   DERR            error if non-zero
1287 FE88 17   FCE1      9        LBSR  NUMB            try get number
1288 FE8B 28   3B        3        BVC   RTS2            got one, stay in monitor
1289 FE8D 10FE 035B      7        LDS   STACK           reset stack and..
1290 FE91 6E   A4        3        JMP   ,Y              ..go to program
1291 FE93 8E   FFCE      3 DERR   LDX   #DQRY           query user..
1292 FE96 17   FA53      9        LBSR  STRING          ..on display
1293 FE99 17   FBE5      9        LBSR  OPAREG          with error number
1294 FE9C 16   FA4A      5        LBRA  OPCRLF          newline and exit
1295                            *
1296                            *             this routine sends 1 command followed by a variable
1297                            *             number of parameters, possibly none
1298                            *             x points to command, next byte is number of parameters
1299                            *             x left pointing after last parameter, destroys d
1300                            *
1301 FE9F EC   81        8 CMDPAR LDD   ,X++            get command and number
1302 FEA1 0D   00        6 CP1    TST   FDCS            test status and..
1303 FEA3 2B   FC        3        BMI   CP1             ..wait if busy
1304 FEA5 97   00        4        STA   FDCC            send command
1305 FEA7 5A             2 CP2    DECB                  if no more parameters..
1306 FEA8 2B   1E        3        BMI   RTS2            ..then exit
1307 FEAA 96   00        4 CP4    LDA   FDCS            if parameter..
1308 FEAC 85   20        2        BITA  #$20            ..register full..
1309 FEAE 26   FA        3        BNE   CP4             ..then wait
1310 FEB0 A6   80        6        LDA   ,X+             get parameter and..
1311 FEB2 97   01        4        STA   FDCP            ..send it
1312 FEB4 20   F1        3        BRA   CP2             then repeat
1313                            *
1314                            *             test if drive ready, on entry x points to read drive
1315                            *             status command sequence, on exit drive is ready and x points
1316                            *             to next command sequence
1317                            *
1318 FEB6 1F   13        6 DRVRDY TFR   X,U             save pointer
1319 FEB8 1F   31        6 DR2    TFR   U,X             restore pointer
1320 FEBA 8D   E3        7        BSR   CMDPAR          ask for drive status
1321 FEBC 96   00        4 DR1    LDA   FDCS            wait until..
1322 FEBE 85   10        2        BITA  #$10            ..result,.
1323 FEC0 27   FA        3        BEQ   DR1             ..ready
1324 FEC2 96   01        4        LDA   FDCR            get result
1325 FEC4 85   04        2        BITA  #DRIVE/128*60+4 ready bit mask
1326 FEC6 27   F0        3        BEQ   DR2             not ready, wait
1327 FEC8 39             5 RTS2   RTS
1328                            *
1329                            *             this routine transfers data from disc to memory
1330                            *             sta-ting at address in u.  returns completion code in
1331                            *             'a' when transfer finished or error occurs
1332                            *
1333                            *
1334 FEC9 34   01        6 TRNSFR PSHS  CC              save CC while..
1335 FECB 1A   50        3        SEIF                  ..set masks, disc is on NMI
1336 FECD C6   04        2        LDB   #$04            data available mask
1337 FECF 20   04        3        BRA   TRN2
1338 FED1 96   04        4 TRN1   LDA   FDCD            get data..
1339 FED3 A7   C0        6        STA   ,U+             ..and store it
1340 FED5 3C   FF       20 TRN2   CWAI  #$FF            wait for interrupt
1341 FED7 D5   00        4        BITB  FDCS            check data..
1342 FED9 26   F6        3        BNE   TRN1            ..available
1343 FEDB 35   01        6        PULS  CC              get interrupt masks back
1344 FEDD 96   01        4        LDA   FDCR            then get result
1345 FEDF 39             5        RTS                   and return Z bit accordingly
1346                            *
1347                            *             interrupt request handler, comes here on irq active
1348                            *             checks for timer 1 or keyboard interrupt, if neither
1349                            *             then complains to user.
```

```
                        *
FEE0 86   03       2 IRQHAN LDA    #MONDP          set up..
FEE2 1F   8B       6        TFR    A,DP            ..direct page
               0003         SETDP  MONDP           tell assembler
FEE4 8E   0980     3        LDX    #KVIA           point to via address
FEE7 A6   0D       5        LDA    KIFR-KVIA,X     get flag register
FEE9 2A   1A       3        BPL    UNUSED          not the via!
FEEB 84   40       2        ANDA   #T1IFLG         try timer 1
FEED 27   06       3        BEQ    IRQH1           not timer 1
FEEF A7   0D       5        STA    KIFR-KVIA,X     clear the interrupt
FEF1 6E   9F 0373  8        JMP    [IRQRTS]
                        *
FEF5 A6   0D       5 IRQH1  LDA    KIFR-KVIA,X     get flags again
FEF7 84   10       2        ANDA   #CB1FLG         try for keyboard
FEF9 27   0A       3        BEQ    UNUSED          query user if not
FEFB A7   0D       5        STA    KIFR-KVIA,X     clear the interrupt
FEFD A6   00       5        LDA    KIRB-KVIA,X     and get a character..
FEFF 84   7F       2        ANDA   #$7F            ..stripping spare bit
FF01 17   FAFF     9        LBSR   HAVCHR          put into buffer..
FF04 3B           15 ANRTI  RTI                    ..then leave irq level
                        *
                        *
                        *               come here if unused interrupts are active, so
                        *               complain to user and stop processor
                        *               since cannot clear an unknown interrupt
                        *
FF05 8E   FFCC     3 UNUSED LDX    #IERR           query user..
FF08 17   F9E1     9        LBSR   STRING          ..on display
FF0B 20   FE       3        BRA    *               and stop dead!
                        *
                        *
FF0D      3F283444 CRTCSV FCB    $3F,$28,$34,$44         set up table for s version 6845 crt controller
FF11      1E02191B       FCB    $1E,$02,$19,$1B
FF15      03127013       FCB    $03,$12,$70,$13
                        *
FF19      3F283404 CRTCRV FCB    $3F,$28,$34,$04         set up table for r version crt controller
FF1D      1E02191B       FCB    $1E,$02,$19,$1B
FF21      00096809       FCB    $00,$09,$68,$09
                        *
                        *               set up table 1, copied directly to ram at rtab1
                        *
FF25      035B     PTAB1 FDB    ISTACK          initial stack pointer
FF27      0000           FDB    0               trace initialy off
FF29      7F             FCB    BSPACE          character echoed on rubout
FF2A      FF             FCB    $FF             echo on, line buffer on
FF2B      00             FCB    0               printer off
FF2C      0A             FCB    LF              do not send line feeds, use cr only to printer
FF2D      00CD           FDB    205             initial baud rate for cassette is 300
                        *                        110 baud use 564 ($0234)
                        *                        300 baud use 205 ($00CD)
                        *                       1200 baud use  48 ($0030)
                        *
FF2F      F95B           FDB    DISPLA          console output
FF31      FA5E           FDB    GETCHR          console input
FF33      FD25           FDB    MCASOP          cassette output
FF35      FD53           FDB    MCASIN          cassette input
FF37      FA9D           FDB    PRINT           printer routine
FF39      FF7F           FDB    FUNCTS          display function table
FF3B      FF51           FDB    CMNDS           monitor command table
FF3D      FF05           FDB    UNUSED          initial timer 1 routine
FF3F      0000           FDB    0               no memory interpret
FF41      FF05           FDB    UNUSED          reserved vector
FF43      FF05           FDB    UNUSED          swi3
FF45      FF05           FDB    UNUSED          swi2
FF47      FF05           FDB    UNUSED          firq
FF49      FEE0           FDB    IRQHAN          irq
FF4B      FBC3           FDB    SWIHAN          swi
FF4D      FF05           FDB    UNUSED          nmi
FF4F      0000           FDB    0               initial load offset
                        *
          FF51     PTAB2 EQU    -               end of table
                        *
                        *
                        *               this table contains the standard set of commands
                        *               provided by the monitor.
                        *
FF51      0E       CMNDS FCB    CMNDE-CMNDS/3-1 number of entries
FF52      47             FCB    'G
FF53      FCB0           FDB    GOUSER-CMNDS    go to program
FF55      4D             FCB    'M
FF56      FBB0           FDB    MEM-CMNDS       memory examine
FF58      52             FCB    'R
FF59      F967           FDB    EXREG-CMNDS     examine registers
FF5B      50             FCB    'P
FF5C      FC68           FDB    RESUME-CMNDS    proceed after break
FF5E      54             FCB    'T
FF5F      FD10           FDB    TRACEN-CMNDS    set trace number
FF61      53             FCB    'S
FF62      FE5C           FDB    SAVE-CMNDS      save on cassette
FF64      4C             FCB    'L
FF65      FD50           FDB    LOAD-CMNDS      load from cassette
FF67      56             FCB    'U
FF68      FCD2           FDB    BRKSET-CMNDS    set break address
FF6A      44             FCB    'D
FF6B      FEF3           FDB    BOOT-CMNDS      disc bootstrap
FF6D      43             FCB    'C
```

```
1446 FF6E      FD29              FDB   PCNTL-CMNDS      copy to printer
1447 FF70      20                FCB   SPACE
1448 FF71      FB0C              FDB   ANRTS-CMNDS      ignore spaces
1449 FF73      2C                FCB   COMMA
1450 FF74      FB0C              FDB   ANRTS-CMNDS      ignore commas
1451 FF76      2E                FCB   '.
1452 FF77      FD16              FDB   TRACE-CMNDS      do trace operation
1453 FF79      46                FCB   'F
1454 FF7A      FD77              FDB   LOAD?-CMNDS      finish file load
1455 FF7C      01                FCB   1
1456 FF7D      FD38              FDB   BADCMD-CMNDS     default is query user
1457           FF7F       CMNDE  EQU   *
1458                             *
1459                             *           this table contains the standard functions provided
1460                             *           by the vdu control programs.
1461                             *
1462 FF7F      04         FUNCTS FCB   FUNCTE-FUNCTS/3-1       number of entries
1463 FF80      0D                FCB   CR
1464 FF81      FA6D              FDB   DOCR-FUNCTS      carriage return
1465 FF83      0A                FCB   LF
1466 FF84      F9F8              FDB   DOLF-FUNCTS      line feed
1467 FF86      7F                FCB   BSPACE
1468 FF87      FA06              FDB   DORUB-FUNCTS     rubout
1469 FF89      0C                FCB   FFEED
1470 FF8A      F9A6              FDB   CRTCI-FUNCTS     form feed
1471 FF8C      01                FCB   1
1472 FF8D      F9EA              FDB   SIMCHR-FUNCTS    default is display it
1473           FF8F       FUNCTE EQU   *
1474                             *
1475                             *           this is the list of strings used by the monitor
1476                             *
1477 FF8F      526F6D3F   MQRY   FCC   /Rom?/
1478 FF93      0D0A00     SCRLF  FCB   CR,LF,0
1479 FF96      43432020   TITLES FCC   /CC  A  B DP    X /
              41202042
              20445020
              20202058
              20
1480 FFA7      20202059          FCC   /  Y    U   PC    S/
              20202020
              55202020
              50432020
              202053
1481 FFBA      0D0A00            FCB   CR,LF,0
1482 FFBD      0D0A       PCMESS FCB   CR,LF
1483 FFBF      50435D            FCC   /PC]/
1484 FFC2      00                FCB   0
1485 FFC3      57686174   CQRY   FCC   /What is:/
              2069733A
1486 FFCB      00                FCB   0
1487 FFCC      49         IERR   FCC   /I/
1488 FFCD      2D         LQRY   FCC   /-/
1489 FFCE      45727220   DQRY   FCC   /Err /
1490 FFD2      00                FCB   0
1491                             *
1492                             *           this is the set of indirect jumps to redirect
1493                             *           the interrupt vector addresses.
1494                             *
1495 FFD3 6E   9F 0377  8 RESVI  JMP   [IRESV]
1496 FFD7 6E   9F 0379  8 SWI3I  JMP   [ISWI3]
1497 FFDB 6E   9F 037B  8 SWI2I  JMP   [ISWI2]
1498 FFDF 6E   9F 037D  8 FIRQI  JMP   [IFIRQ]
1499 FFE3 6E   9F 037F  8 IRQI   JMP   [IIRQ]
1500 FFE7 6E   9F 0381  8 SWII   JMP   [ISWI]
1501 FFEB 6E   9F 0383  8 NMII   JMP   [INMI]
1502                             *
1503                             *           the following hardware vectors reside in the top
1504                             *           16 bytes of memory when the monitor is in its
1505                             *           standard position.
1506                             *
1507 FFF0                        ORG   $FFF0
1508           E7F0              PUT   $E7F0
1509                             *
1510 FFF0      FFD3              FDB   RESVI
1511 FFF2      FFD7              FDB   SWI3I
1512 FFF4      FFDB              FDB   SWI2I
1513 FFF6      FFDF              FDB   FIRQI
1514 FFF8      FFE3              FDB   IRQI
1515 FFFA      FFE7              FDB   SWII
1516 FFFC      FFEB              FDB   NMII
1517 FFFE      F800              FDB   RESET
1518                             *
1519                             END
```



ALTERNATIVE RIBBON HEADER PIN NUMBERS			
20	DATA 5	19	DATA 6
18	DATA 4	17	STROBE
16	DATA 3	15	RST
14	DATA 2	13	+5V
12	DATA 1	11	0V
10	DATA Φ	9	PAΦ
8	E	7	PA1
6	CASIN	5	PA4
4	CASO	3	PA2
2	0V5	1	PA3

CODE	TYPE	0V	+5V	CODE	TYPE	0V	+5V
IC1	MC6809	1	7	IC11	SN75470	10	20
IC2	SY6522A	1	20	IC12	SN74LS74	7	14
IC3,9	74LS244	10	20	IC13,14	SN74LS00	7	14
IC4	2716/32	12	24	IC15	SN74LS86	7	14
IC5,6	2114	9	18				
IC7,8	74LS244	10	20				
IC10	INS 8208	10	20				

① LINK IN 3 PLACES FOR IC4 (2732)
② CUT 1 TRACK FOR IC4 (2732)
BOARD IS SUPPLIED FOR IC4 (2716)

ISSUE	2		© COPYRIGHT 1980 ACORN COMPUTERS LTD 4A MARKET HILL CAMBRIDGE. TEL 0223-312772	DRN GWOS	TITLE CIRCUIT DIAGRAM FOR 6809 PROCESSOR CARD	200,012/C
DATE	7-2-80			DATE 4-2-80		ACORN COMPUTERS LTD
	CBT					

Acorn Computers Ltd., 4a Market Hill, Cambridge.
Telephone 0223 312772