ACORN6809 User's Manual

<u>ACORN 6809</u>

TECHNICAL AND PROGRAMMING MANUAL

Chapter	1	-	Introduction	1
	2	-	Monitor Operation	4
	3	-	Summary of Monitor Commands	11
	4	-	Monitor Expansion	16
	5	-	Hardware Description	23
	6	-	Software Description	33
	7	-	Instruction Set and Addressing Modes	38
	8	-	Programming Techniques	55
	9	-	Assembly Instructions	67
Appendix	K		- Monitor Program Listing	77

© 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:

<u>*</u>M200

0200 XX 30,8D,00,04

0204 XX BD, F8, EC

<u>0207 XX</u> 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 \$08,\$8D,\$4F,\$4B,\$21,\$0D,\$0A FCB 0216 00 FCB \$00 TERMINATOR

To execute the program use the 'G' (Go) command. Type: \pm 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:

<u>*</u>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

```
*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
      ΒE
            0400 READ
                        LDX
                                E$400
                                            DISPLAY AREA START
0013
           FD53 READ2
                                            INPUT A FROM CASSETTE
     BD
                        JSR
                                MCASIN
0016
     Α7
          80
                        STA
                                ,X+
```

001D 20 Fl BRA READ

£\$800

READ2

DISPLAY AREA END

CMPX

BNE

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

2.3 Store and Load

0800

F6

0018 8C

26

0018

Execute the program by typing:

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

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 Х 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:

<u>*</u>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:

<u>*</u> 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:

<u>*</u> 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:

<u>*</u>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: \pm V200 T1

Then start execution at 200 with:

<u>*</u>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: <u>What is:X</u>

The rest of the line is ignored.

An unrecognized interrupt will give the message:

<u>I-Err</u>

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: $\underline{\text{Err XX}}$

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'

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.

11

MODIFY

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.

ΜP

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'

GΟ

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.

12

MG

Program Debugging

R

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.

13

REGISTERS

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 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.

14

LOAD

Printer Interface

C '+'

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

COPY

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		
			* LINKE *	ED TO MO	NITOR COMMAN	D SET
	01 44		AUXTAB	FCB FCB	1 'D	NO. OF COMMANDS IN TABLE COMMAND NAME
0202 0204	0007 FF	7		FDB FCB	DUMP-AUXTAB	OFFSET TO ROUTINE GO TO NEXT TABLE
0205	FF51	-	*	FDB	CMNDS	MONITOR COMMAND TABLE
0207 020A 020C	1F	FB6C 02 18	DUMP		D,Y	GET HEX NO. IN D PUT IN Y NO. OF LINES
020E 0210	3402	2	PSHS LDB	A	SAVE COUNTE BYTES PER L	R
0212 0214 0216	1F	24 21 FA75			Y,B Y,X OPXREG	X = ADDRESS PRINT X IN HEX + SPACE
0210 0219 0218	A6	AO FA97	D1	LDA JSR	,Y+ OPARSP	GET BYTE PRINT A IN HEX + SPACE
021F		F8		DECB BNE	D1	
0221 0223 0225		24 AO 20	D2	PULS LDA CMPA	Y,B ,Y+ £\$20	RESTORE GET BYTE AGAIN ASCII PRINTABLE ?
	25	04 7E		BCS	D3 £\$7E	TOO BIG ?
022B 022D	86	02 FF	D3	LDA	D4 £\$FF	PRINT WHITE BLOCK
0232	5A	FA21 EE	D4 BNE	JSR DECB D1	CONOUT	CONSOLE OUTPUT A
0235	BD 6A	F8E9 E4 D4	2.12		OPCRLF ,S GO	START NEW LINE DECREMENT LINE COUNTER
023A 023C		82	*	PULS	A,PC	PULL COUNTER + RETURN.
			* ADDRI *	ESSES IN	MONITOR	
			CMNDS NUMB OPXREG OPARSP	EQU EQU	\$FF51 \$FB6C \$FA75 \$FA97	COMMAND TABLE INPUT HEX TO D OUTPUT HEX FROM X OUTPUT HEX FROM A
			CONOUT OPCRLF	EQU	\$FA21 \$F8E9	PRINT A NEW LINE *
				END		

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

<u>*</u>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;

<u>*</u>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:

<u>*</u>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		
	* INTE * *	RRUPT SER	VICE ROUTINE	
	0109	PSHS LDX	X COUNTER	FIRQ DOESN'T STACK REGS
0107 BD H	01 FA75 0109	LEAX JSR STX	1,X OPXREG COUNTER	ADD 1 PRINT HEX; SAVES REGS REPLACE IT
	10 *	PULS RTI	X	RESTORE X
	* SET-U *	JP ROUTINE	5	
0113 BF (0100 SETUP 037D BF	LDX STX ANDCC RTS	ISR IFIRQ £\$BF	ENABLE FIRQ
0119 0000 0	* COUNT	FDB 19	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:

		ISTRATIO	N 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 BLOCKO signal is low for addresses in the bottom 4K of memory, and is used to enable the VDU card.

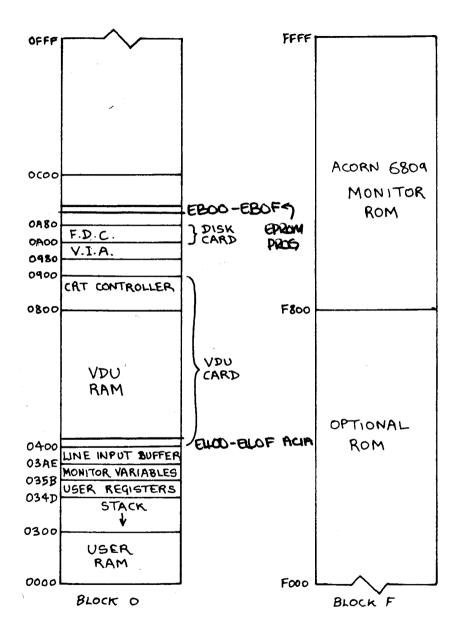


Fig. 5.1. 6809 Card Memory Map.

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 trⁱ-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 trⁱ⁻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 VMA line is provided which is low when the bus buffers are enabled. The 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. 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 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.

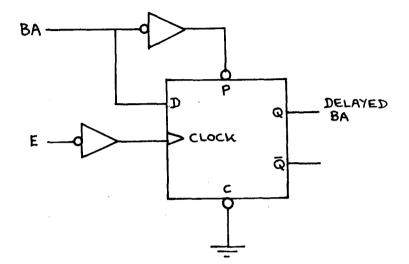


Fig. 5.2. Generation of delayed BA signal.

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

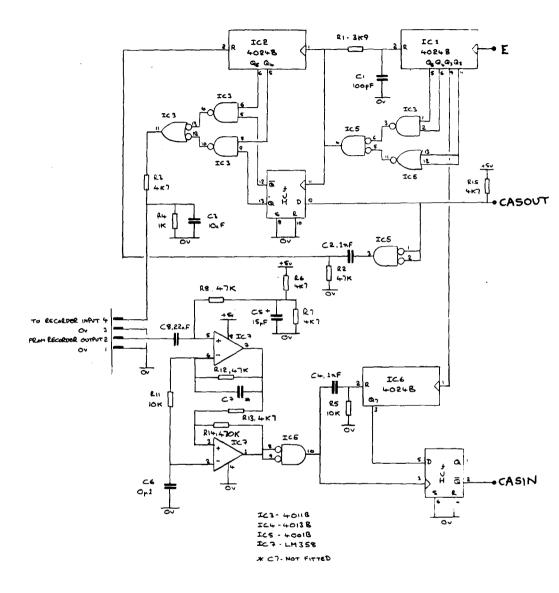


Fig. 5.3. Suggested circuit for a cassette interface. The three signals on the right side of the circuit connect to the 6809 card. output from CB2 of the VIA, lowest bit first.

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.

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 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 80character 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 nonzero. 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 FD5	3 Software asynchronous receiver. Gets value into A
	with 1 stop bit. Saves all other registers.
MCASOP FD2	5 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 FD1	E Get one byte from tape, and update checksum.
CBIN2 FD1	3 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:

Dispach routine. Looks up character in A in a table DISPCH F9A6 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 Calculate real address of cursor in memory space. CCLOCN F905 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

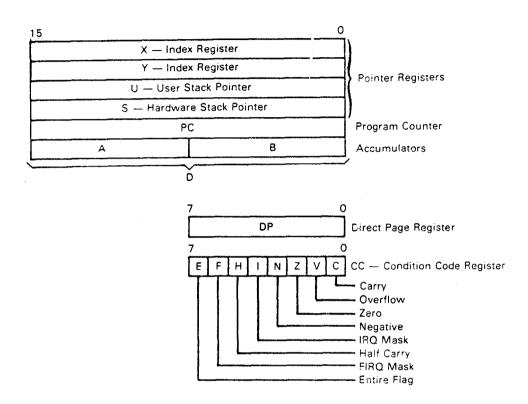


Fig. 7.1. Programming model of the 6809.

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'scomplement 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 42

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:

------ Increasing Address -----> FFFF 0000 -CC A В DP Х 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	Х	Y	U	S	PC	A	В	CC	DP
Hex Digit:	0	1	2	3	4	5	8	9	А	В

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 8bit 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	1	
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	Х,Ү	Transfer	зΧ	into Y	Ľ				
EXG	A,B	Exchanges	s A	and B					
PSHS	А, В, Х, Ү	Push onto	S	stack	Υ,	х,	в,	then	Α
PULU	X,Y,D	Pull from	n U	stack	D,	Х,	the	n 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 N	IEAR	(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 positionindependent 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:

 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-1 *	DECIMAL VERSION	1 1
	* NON RELOCA	TABLE	
	* NON RE-ENT	RANT	
	*		
0000	COUNT RMB1		
0001	TEMP RMB1 *		
0002 2710) K10TAB FDB	10000	
0004 03E8	B FDB	1000	
0006 0064		100	
0008 0002		10	
	*		
000A 8E 0002		£K10TAB	POINT TO TABLE
000D 7F 0000) LOOP1 CLR	COUNT	
0010 20 03	BRA	NOINC	
0012 7C 0000		COUNT	
0015 A3 84	NOINC SUBD	,X	SUBTRACT POWER OF 10
0017 24 F9	BCC		MAKE DOOTETTE
0019 E3 81 001B B7 0001	PRN5 STA	,	MARE POSITIVE
001E B6 0001		COUNT	
0011 BD FA8E			IN MONITOR
0021 BD 1401 0024 B6 0001			RESTORE D
0027 8C 0007			ALL DONE?
002A 26 E1	BNE	LOOP1	
002C IF 98	TFR		GET REMAINDER
002E BD FA8E	JSR	,	LAST DIGIT OF RESULT
0031 39	RTS		

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

		*				
		* BINAR	RY-TO-DE	CIMAL VERSIO	N 2	
*						
* RELOCATABLE AND RE-ENTRANT						
		* ALL H	REGISTER	S 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	А,В,Х,Ү	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	
	FA8B		JSR	HEXOUT	PRINT IT	
002E 32	62		LEAS	, -	RESTORE WORKSPACE	
0030 35	В6		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 if n=0 = 1 if n=r = C(n-1,r) + C(n-1,r-1) otherwise.

For this definition the function has beed written in the form $C\left(n,r\right)$ 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.

		* * RECUI	RSIVE SUB	ROUTINE TO	CALCULATE
			MIAL COEF		
		* RELO(*	CATABLE &	RE-ENTRANT	C
0000 36 0002 5D	06	NCR	PSHU TSTB	А,В	
0003 27 0005 El	04 C4		BEQ CMPB	ONE ,U	B=0?
0007 26 0009 CC	05 0001	ONE	BNE LDD	NONE El	A=B?
000C 20 000E 4A	OD	NONE	BRA DECA	RESULT	RETURN 1
000F 8D 0011 EC 0013 5A 0014 4A	EF 42		BSR LDD DECB DECA	NCR 2,U	C(A-1,B)
0015 8D 0017 37	E9 06		BSR PULU	NCR A,B	C(A-1,B-1)
0019 E3 001B ED 001D 39	C1 C4	RESULT	RTS	,U++ ,U	ADD RESULTS ON USER STACK
		* TEST *RESULI *	C(7,3) F SHOULD E	3E \$23	
001E CE 0021 CC 0024 BD 0027 37 0029 7E	0300 0703 0000 10 FA75	TEST	LDU LDD JSR PULU X JMP	£\$0300 E\$0703 NCR PRINT RESU OPXREG	

Fig. 8.3. Recursive routine to calculate ${}^{\rm n}{\rm C}_{\rm r}.$

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
АВА	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
MAI	CWAI #\$FF

8.3.1 Software Incompatibilities Between 6800 and 6809.

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

- Instruction timings in 6809 will, in general, be different from other 6800-family processors.
- The 6809 uses the two high-order condition code register bits. Consequently, these will not, in general, appear as 1's as on the 6800.

- 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 $V = b_7 \odot b_6$.

- 3. The 6809 H-flag is not defined as having any particular state after subtract-like operations (CMP, NEG, SEC, 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 eq.

> 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				
Cl	22uF electrolytic capacitor				
C2 - 6	5 off 47 (or 100) nF capacitor				
C7 - 8	2 off 24pF capacitors				
С9	Optional - not supplied.				

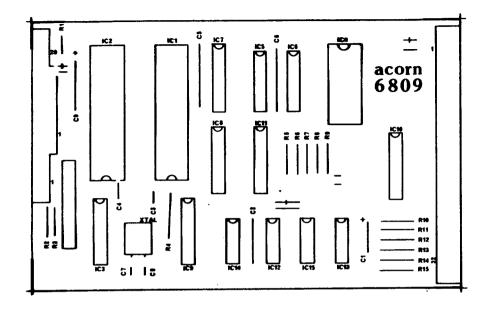
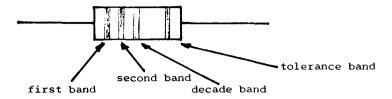


Fig. 9.1. 6809 Card Component Layout.

The resistor colour code is shown here.

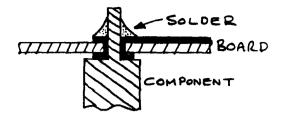


The first and second bands give the resistor value and the decade band shows the number of zeros following:-

0	Black	
1	Brown	
2	Red	
3	Orange	e.g. Yellow, Violet, Orange
4	Yellow	is Yellow, Violet . 4,7 and
5	Green	Orange 3 zeros i.e. 000.
6	Blue	So the value is 47000 ohms, $% \left({{{\left({{{\left({{{\left({{{\left({{{}}} \right)}} \right.}} \right)}_{0.2}}}}} \right)} \right)$
7	Violet	i.e. 47 kilo-ohms or 47K.
8	Grey	
9	White	

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

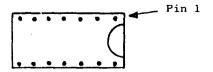
Assembling the card will require a considerable amount of soldering and a small electric soldering iron is essential with a diameter at the end of the bit not exceeding 0.1 inches. The iron should be rated between 10 and 30 watts and fine 22 guage flux cored solder should be used. If you have never soldered 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.



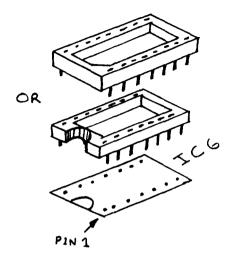
Some of the integrated circuits used in the system employ M.O.S. technology and they can be damaged by static electricity. As a general rule if there is no noticable static charge in the area and no nylon clothes or carpets are present all will be well. An earthed soldering iron should be used when soldering 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:-



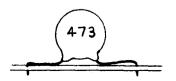
The sockets will have either a 45° chamfer for pin 1 or a semi circular cut out as shown:-



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.



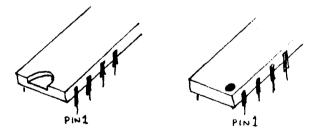
Do not crack the capacitor body when bending the leads.

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.

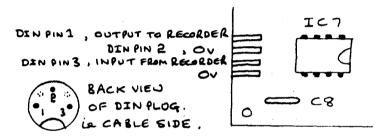
Note that IC1 and IC2 are the opposite way round to the other ICs on the 6809 card.

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	10 Data 1.'. C
18 Data bit 4	19 Data bit 6
16 Data bit 3	17 Key strobe
14 Data bit 2	15 Reset
12 Data bit 1	13 + 5 volts
10 Data bit 0	11 0 volts
	9 -
8 E	7 –
6 CASIN	5 -
4.CASOUT	3 -
2 -	1 -
	1 - 75

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,	Ecl	Lock
	"		В1	9	"	"		в12,	CASIN
	"		B2	0	"	"		в13,	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 0002		*		Acorn monito	r for 6889 processor
0003		*			
0004		*		ISSUE 1 -	V49 - FEB 1980
0005		*			
0006 0007		*		This program	handles a memory mapped vdu, encoded keyboard,
0008		*			erface, 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 0018	0800 000C	CRTC PAGHI	EQU EQU	\$0800 12	location of crt controller on vdu card 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	0000	FLOPY	EQU	\$A00 EL OBULO	location of floppy disc controller
0023 0024	0a00 0a00	FDCC	EQU	FLOPY+0 FDCC	command register status register
0025	0A01	FDCP	EQU	FLOPY+1	parameter register
0026	0401	FDCR	EQU	FDCP	result register
0027	0A02	FDRST	EQU	FLOPY+2	reset register
0028 0029	0684	FDCD	EQU	FLOPY+4	data register
0029	0980	KUIA	EQU	\$0980	location of versatile interface adaptor
0031	0980	KORB	EQU	KUIA+\$0	output register b
0 032	0980	KIRB	EQU	KORB	input ragister b
0033	0981	KORA KIRA	EQU	KUIA+\$1 KORA	output register a input register a
0034 0035	0981 0982		EQU	KUIA+\$2	data direction register b
0036	0983	KDDRA	EQU	KUIA+\$3	data direction register a
0037	0984	KTICL	EQU	KUIA+\$4	timer 1 counter low
0038	0985		EQU	KU1A+\$5	timer 1 counter high
0039 0040	0986 0987	KTILL KTILH	EQU EQU	KUIA+\$6 KUIA+\$7	timer 1 latch low timer 1 latch high
0041	0988		EQU	KUIA+\$8	timer 2 counter low
0042	0989	KT2CH	EQU	KVIA+\$9	timer 2 counter high
0043	098A	KSR	EQU	KUIA+\$A	shift register
0044 0045	0988 0980	KACR KPCR	EQU	KUIA+\$B KUIA+\$C	auxilliary control register peripheraal control register
0045	098D		EQU	KUIA+\$D	interrupt flag register
0047	098E		EQU	KUIA+SE	interrupt enable register
0048	098F	KORA2	EQU	KVIA+\$F	input/output register a without handshake
0049 0050	0F00	INTDEL	FOU	15*256	delay for single instruction trace is 15 cycles
0051	0040	TIIFLG		201000000	interrupt flag position for timer 1
0052	0010	CBIFLG		200010000	interrupt flag position for keyboard input
0053		*			the second se
0054 0055	ØØEF	IKPCR	EQU	%11101111	initial value for peripheral control register, bit 0, positive edge printer interrupt on cal,
0056		*			but not used in this monitor.
8057		*			bits 1-3, output to activate printer, normally high
0058		*			bit 4, negative edge keyboard interrupt
0059 0060		*			bits 5-7, cassette output initially set high
0061	0000	TKIER	EQU	211010000	interrupt enable register control
0062		*			bit 4, keyboard interrupt enable
0063		*			bit 6, timer 1 interrupt enable
. 9964 9965		*			bit 7, set interrupts enabled
8866		*			other enables not altered
8867	0002	PSTRB	EQU	200000010	bit postion of printer strobe in pcr
0068	0020	COPBIT	EQU	200100000	bit position that controls cassette output in pcr
0069 0070	003F	* SW1	EQU	\$3F	
8871	000	*	EWO	₩0F	a software interrupt used for breakpoints
0072	0052	BUFLEN	EQU	81+1	buffer up to 88 characters from keyboard
0073		*			
0074 0075	002A 007F	Prompt Rubch		* *7F	monitor prompt character keyboard character that does rubout operation
0075	007F	BSPACE		\$7F	character that backspaces vdu, also used to do rubout
8877	900A		EQU	\$0A	linefeed character
0078	000D		EQU	\$60	carriage return character
9079 9080	0020 0020		EQU	\$28	blank space character
0081	0020		EQU	12	a comma character a minus character
0082	999C		EQU	\$0C	character used as clear screen command
0083	003B	SEMIC	EQU	· ;	a semicolon
9984 9985	0010	*			
0036	0004		EQU	\$10 \$04	
0087		*			
0088		*		definitions	of variables on page 3
9689 ·		*	000	40350	
0090 0358 0091		*	ORG	\$035B	
0092	035B	ISTACK	EQU	*	stack pointer starts here
0093	035B	RTAB1		*	ram table 1 starts here
0094 0005		*			
0095 0096		*		this table of	opied from rom on stært up
0097		*			
					· ·

0098	035B	0002	STACK	RMB	2	positon of stack pointer when empty
0099	0350	0002	NTRACE		2	number of instructions to trace-before stopping
0100	035F	0001	BSECHO		1	character sent to backspace display
0101	0360	0001	ECHOF	RMB	1	keyboard buffer/echo control
0102			*			bits 8-5, 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		0001	PFLAG	RMB	1	printer control flag, echo console output to printer if
0107		0001	PNEH	RMB	1	this character not sent to printer
0108		0002	DELCN	RMB	2	address for console output
0109		0002	COPADE		2	address for console input
0110		0002	CINAD		2	
0111		9992	CASOPA		2	address for cassette output address for cassette input
0112		0082	CASIN		2 2	address of printer output routine
0113		0002	PRINTI FUNCTI		2	address of vdu function table
0114		9982 9982	CHNDI		2	address of monitor command table
0115 0116		0002	IRGRTS		2	address to go to on timer 1 interrupt
		0002	LINEPT		2	address of memory input line. Ø if none
0117 0118		0002	IRESU	RMB	2	address of reserved vector routine
0119		0002	ISHI3	RMB	2	address of swi3 routine
0120	037B	0002	15412	RMB	2	address of swi2 routine
0121	0370	0002	IFIRG	RMB	2	address of firg routine
0122		0002	IIRQ	RHB	2	address of irg routine
0123		0002	ISHI	RMB	2	address of swi routine
0124		0002	INMI	RMB	2	address of nmi routine
6125	0385	0002	OFFSET	RMB	2	cassette load offset
0126			*			
0127			*		general variabl	es for monitor use
0128			*		_	
0129		0002	HEADST		2	static head pointer into line buffer
0130		0002	HEADDY		2	dynamic head pointer into line buffer
0131		0002	TAIL	RMB	2	tail pointer into line buffer
0132		0002	MSTACK		2	stack saved whilst memory interpreting
0133		0001	CROM	RMB	1	current row of cursor on display
0134		0001	CCOL	RMB	1	current column of cursor on display
0135	0391	0002 0002	CPAGE	RMB	2	current start of display page in memory
0136			MSAU	RMB	2	saved address for memory command
0137		0002 0006	gsav Name	RMB RMB	2 6	saved address for go command saved name for cassette input/output
0138		0002	CSSTRT		2	saved cassette output start address
0139 0140	0390	8882	CSEND.	RMB	2	saved cassette output start address
0141		8001	ONLINE		1	flag set to zero when find or in input line
0:42		9001	LASTC	RMB	i	saved last character from input line
0143		0002	CBREAK		2	current address of a breakpoint, \$FFFF if none
0144		0002	NBREAK		2	number of breakpoints to ignore before stopping user
0145	9367	0001	CINST	RMB	1 .	user instruction at breakpoint address
0146		0002			2	number of instructions left to trace before stopping us
			USRSTK			
6147	03AA	0002	USRSTK	RMB	2	saved user stack pointer when user halted
0147 0148	03AA 03AC			RMB RMB	2	saved user-stack pointer when user halted temporary storage
0147 0148 0149 0150	03AA 03AC 03AE	0002 0002	USRSTK TEMP	RMB RMB	2 2 BUFLEN	saved user stack pointer when user halted
0147 0148 0149 0150 0151	03AA 03AC 03AE	0002 0082 0052	USRSTK TEMP	RMB RMB RMB ORG	2 2 BUFLEN MPROM	saved user-stack pointer when user halted temporary storage
0147 0148 0149 0150 0151 0152	03AA 03AC 03AE	0002 0002	USRSTK TEMP BUFFER *	RMB RMB RMB	2 2 BUFLEN	saved user-stack pointer when user halted temporary storage
0147 0148 0149 0150 0151 0152 0153	03AA 03AC 03AE	0002 0082 0052	USRSTK TEMP BUFFER *	RMB RMB RMB ORG	2 2 BUFLEN MPROM	saved user-stack pointer when user halted temporary storage
0147 0148 0149 0150 0151 0152 0153 0154	03AA 03AC 03AE	0002 0082 0052	USRSTK TEMP BUFFER *	RMB RMB RMB ORG	2 2 BUFLEN NPROM \$E000	saved user stack pointer when user halted temporary storage line input buffer
0147 0148 0149 0150 0151 0152 0153 0154 0155	03AA 03AC 03AE	0002 0082 0052	USRSTK TEMP BUFFER * *	RMB RMB RMB ORG	2 2 BUFLEN NPROM \$E000	saved user-stack pointer when user halted temporary storage
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156	03AA 03AC 03AE F800	9992 9992 9952 E999	USRSTK TEMP BUFFER * * *	RMB RMB RMB ORG PUT	2 2 BUFLEN MPROM \$E000 hardware reset	saved user stack pointer when user halted temporary storage line input buffer starts at this address
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157	03AA 03AC 03AE F800 F800 B6	0002 0002 0052 E000	USRSTK TEMP BUFFER * * 2 RESET	RMB RMB RMB ORG PUT	2 2 BUFLEN MPROM \$E000 hardware reset #MONDP	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup.
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157 0158	03AA 03AC 03AE F800	0002 0002 0052 E000 03 88	USRSTK TEMP BUFFER * * *	LDA	2 2 BUFLEN MPROM \$E000 hardware reset MONDP A, DP	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157 0158 0159	03AA 03AC 03AE F800 F800 86 F802 1F	0002 0002 0052 E000 03 88 90003	USRSTK TEMP BUFFER * * 2 RESET 6	LDA TFR SETDP	2 BUFLEN MPROM \$E000 hardware reset MONDP A, DP MONDP	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler
0147 0148 0149 0150 0151 0152 0153 0154 0155 0155 0156 0159 0160	93AA 93AC 93AE F899 F899 F899 F899 F899 F894 9E	0002 0002 0052 E000 03 88 9003 FF25	USRSTK TEMP BUFFER * * 2 RESET	LDA TFR SETDP LDX	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MPTAB1	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157 0158 0159 0160 0161	83AA 93AC 93AC 93AE F809 F809 F809 1F F804 8E F807 CE	0002 0002 0052 E000 03 88 0003 FF25 0358	USRSTK TEMP BUFFER * * 2 RESET 6 3	LDA TFR SETDP	2 BUFLEN MPROM \$E000 hardware reset MONDP A, DP MONDP #PTAB1 #RTAB1	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler
0147 0148 0149 0151 0152 0153 0154 0155 0156 0157 0158 0159 0160 0161 0:62	93AA 93AC 93AE F899 F899 F899 F899 F899 F894 9E	0002 0002 0052 E000 88 9003 FF25 9358 88	USRSTK TEMP BUFFER * 2 RESET 6 3	LDA TFR SETDP LDU	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MPTAB1	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start
0147 0148 0149 0151 0152 0153 0155 0155 0155 0155 0157 0158 0157 0158 0159 0160 0161 0:62 0163	93AA 93AC 93AC 93AE F809 86 F809 86 F802 1F F804 8E F807 CE F804 A6	0002 0002 0052 E000 88 9003 FF25 90 2358 80 C0 FF51	USRSTK TEMP BUFFER * * 2 RESET 6 3 3 6 RST1 6	LDA TFR SETOP LDA LDA LDX LDU LDA	2 BUFLEN MPROM \$E000 hardware reset MONDP A, DP MONDP #PTAB1 #RTAB1 , X+	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom.
0147 0148 0149 0150 0151 0152 0153 0155 0155 0155 0155 0157 0158 0157 0158 0157 0161 0:62 0161 0:62	03AA 03AC 03AE F800 F800 F800 F800 F802 F802 F804 8E F807 E804 A6 F80C A7 F806 BC F806 A6 F806 A7 F808 BC F808 26 F801 26	0002 00052 00552 00552 00552 00552 00552 00552 00552 00552 00555 00555 00555 00555 00555 00555 00555 00555 00552 00552 00552 00552	USRSTM TEMP BUFFER * * 2 RESET 6 3 RESET 6 6 RST1 6 4 3	RMB RMB RMB RMB ORG PUT UDA SETDP LDX LDA STA CMPX BME	2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 \$FTAB1 \$7.X+ , U+ \$PTAB2 RST1	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom. to ram
0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157 0158 0159 0160 0160 0160 0162 0163 0164 0165	03AA 03AC 03AE F800 86 F802 1F F804 8E F802 1F F804 8E F807 CE F802 A7 F802 8C F811 26 F813 100E	0002 0052 E000 63 88 9003 FF25 63 80 9358 63 80 6358 63 80 63 58 58	USRSTK TEMP BUFFER * * 2 RESET 6 3 3 6 RST1 6 6 4 3 6	RMB RMB RMB RMB ORG PUT LDA SETDP LDX LDU LDA STA CMPX BNE LDS	2 BUFLEN MPROM \$E000 hardware reset eHONOP MONOP #PTAB1 #RTAB1 ,X+ ,U+ @PTAB2 RST1 STACK	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start copy rom. to ram until end. of table setup stack pointer
0147 0149 0150 0151 0152 0153 0153 0155 0155 0155 0156 0157 0158 0160 0161 0:62 0164 0165 0165 0165	03AA 03AC 03AE F900 86 F902 1F F902 1F F902 1F F907 CE F907 CE F907 CE F906 9C F906 9C F911 26 F911 26 F913 100E	0002 00052 0055 0055 0055 0055 0055 005	USRSTK TEMP BUFFER * * 2 RESET 6 3 RESET 6 6 RST1 6 4 3 6 6	RMB RMB RMB RMB ORG PUT TFR SETOP LDX LDU LDA STA CMPX BNE LDX LDX	2 2 BUFLEN MPROM \$E000 hardware reset 0000P	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for.
0147 0148 0149 0150 0151 0152 0153 0155 0155 0155 0155 0155 0159 0160 0161 0:62 0163 0165 0165 0165 0166 0165	03AA 03AC 03AE F800 F800 F800 F802 F802 F802 F802 F804 8 F804 8 F804 F804 F804 F804 F804	0002 0002 0052 E000 88 0003 FF25 0358 80 0358 80 0358 80 C2 FF51 F7 F7 58 F7FE A55A	USRSTK TEMP BUFFER * 2 RESET 6 3 3 6 RST1 6 6 RST1 6 6 4 4	RMB RMB RMB ORG PUT LDA TFR P LDX LDU LDA LDA LDA LDA LDA LDA CMPX	2 BUFLEN MPROM \$E000 hardware reset eHONDP A, DP HONDP ePTAB1 eRTAB1 , X+ , U+ ePTAB2 RST1 STACK MPROM-2 e\$A55A	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom. to ram until end. of table setup stack pointer check for extma rom,
0147 0148 0149 0150 0151 0152 0153 0154 0155 0155 0155 0157 0159 0160 0161 0162 0163 0164 0165 0166 0167 0166 0167 0168	03AA 03AC 03AC 93AC F800 F800 F800 F802 F802 F802 F804 F802 F804 F804 F804 F804 F804 F804 F804 F804	0002 00052 00552 00552 00552 00552 00552 00552 00553 00563 FF555 FF555 F755 F755 F755 F755 F755 64	USRSTN TEMP BUFFEF * * 2 RESET 6 3 RESET 6 6 RST1 6 6 8 7 3 6 6 4 3 3 6 6 4 3 3	RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDU LDA CMPX BNE LDS LDS CMPX BNE	2 2 BUFLEN MPROM \$E000 hardware reset 0,00P 0000P	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start or ram until end or ram until end of table setup stack pointer check for aktra rom, not there.
0147 0148 0149 0150 0151 0152 0153 0155 0155 0155 0155 0155 0155 0156 0162 0164 0165 0166 0167 0168 0179	03AA 03AC 03AE F800 F800 F800 F800 F800 F800 F800 F80	0002 0002 00052 E000 63 88 0003 FF51 F755 6358 80 C0 FF51 F77E A55A 04 97 F7FC 1	USRSTR TEMP BUFFER * * 2 RESET 6 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	RMB RMB RMB ORG PUT PUT LDA SETOP LDX LDU LDA STA CMPX BNE LDX CMPX BNE LDX CMPX BNE	2 2 BUFLEN MPROM \$E000 hardware reset 0HONDP 0ND 0ND 0ND 0ND 0ND 0ND 0ND 0ND	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table start ram table start ram table start ram table start copy rom. . to ram until end. . of table setup stack pointer check for. . extra rom, not there. else call it</pre>
0147 0148 0149 0150 0152 0152 0153 0155 0155 0155 0156 0157 0158 0164 0164 0166 0166 0166 0166 0167 0167	03AA 03AC 03AC 03AC F800 F800 F800 F800 F804 85 F804 A6 F804 F804 F804 F804 F804 F804 F804 F804	0002 00053 00053 00053 00053 00053 00053 00053 00053 00053 00053 00053 00053 00053 0005 000000	USRSTR TEMP BUFFER * * * 2 2 2 3 6 6 8 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 4 3 2 2 3 5 5 7 11	RMB RMB ORG PUT UDA TFR SETDP LDX LDU LDA STA STA STA STA STA STA LDX LDX LDX LDX LDX LDX LDX LDX LDX LDX	2 2 BUFLEN MPROM \$E000 hardware reset MONDP	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. ., direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. o f table setup stack pointer check for. axtra rom, not there. else call it put monitor return.
0147 0148 0149 0150 0151 0153 0155 0155 0155 0155 0155	03AA 03AC 03AC 03AC 03AC 03AC F800 F800 F800 F802 F802 F802 F802 F802	0002 00052 00550 0055 0055 0055 0055 005 00	USRSTR TEMP BUFFER * * 2 RESET 6 6 8 8 6 6 6 6 4 3 3 6 6 4 3 3 5 7 7	RMB RMB RMB ORG PUT UDA TFR SETDP LDX LDA STA SETDP LDX LDA STA STA STA SETDP LDX LDA STA STA STA STA STA STA STA STA STA ST	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 RTAB1 , X+ , U- WPTAB2 RST1 STACK MPROM-2 \$AST556 STRT1 [MPROM-4] #BACK U	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table start copy rom. . to ram until end. . of table setup stack pointer check for. . extra rom, not there, else call it put monitor return. . onto tack</pre>
$\begin{array}{c} 0147\\ 0148\\ 0150\\ 0150\\ 0152\\ 0152\\ 0153\\ 0155\\ 0155\\ 0156\\ 0157\\ 0156\\ 0160\\ 0160\\ 0160\\ 0160\\ 0160\\ 0160\\ 0160\\ 0167\\ 0171\\ 0172\\ \end{array}$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00052 00052 00052 00052 00052 00052 00052 00053 00003 00003 00003 00003 00003 00003 00003 00003 00052 00053 00053 00053 00053 00053 00053 00053 00053 00053 0005 000000	USRSTR TEMP BUFFER * * * 2 2 2 2 3 3 6 6 8 5 7 3 3 5 5 7 7 3	RMB RMB RMB ORG PUT LDA TFR SETOP LDU LDU LDU LDU LDU LDS CMPX BNR LDS RNR LDS LDS LDS LDS LDS LDS LDS LDS LDS LDS	2 2 BUFLEN MPROM \$E000 hardware reset #HONOP #PTAB1 #RTAB1 #RTAB1 ,X+ ,U+ #PTAB2 RST1 STACK MPROM-2 #ADDP #PTAB2 RST1 STACK MPROM-2 #BACK U #BACK U #12	saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. ., direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. o f table setup stack pointer check for. axtra rom, not there. else call it put monitor return.
$\begin{array}{c} 0147\\ 0149\\ 0150\\ 0157\\ 0152\\ 0153\\ 0153\\ 0155\\ 0155\\ 0155\\ 0157\\ 0159\\ 0160\\ 0162\\ 0163\\ 0163\\ 0163\\ 0167\\ 0163\\ 0167\\ 0163\\ 0172\\ 0173\\$	03AA 03AC 03AC 03AC 03AC 03AC F800 F800 F800 F800 F800 F800 F800 F80	0002 00052 0052 0055 0055 0055 0055 005	USRSTR TEMP BUFFER * * 2 RESET 6 6 RST1 6 6 6 6 4 3 3 STRT1 7 3 STRT1 7 3 STRT2	RMB RMB RMB ORG PUT LDA LDA LDA LDA LDA LDA STA CMPX BANE LDX CMPX BANE JSR LDD SHS LDD PSHS	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 RTAB1 , X+ , U- WPTAB2 RST1 STACK MPROM-2 \$AST556 STRT1 [MPROM-4] #BACK U	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for. axtra rom, not there, else call it put monitor return. on stack put dummy.</pre>
$\begin{array}{c} 0147\\ 0148\\ 0149\\ 0150\\ 0151\\ 0152\\ 0152\\ 0152\\ 0155\\ 0155\\ 0155\\ 0156\\ 0157\\ 0161\\ 0162\\ 0164\\ 0166\\ 0166\\ 0166\\ 0166\\ 0166\\ 0167\\ 0173\\ 0173\\ 0173\\ 0175\\ \end{array}$	63AA 63AC 63AE 75800 75800 75800 75800 75800 75800 75800 75800 75811 75811 75812 75812 75815 7585 758	0002 00053 00003 0055 00053 00053 00053 00053 00053 00053 00053 00053 00053 00053 0005000000	USRSTR TEMP BUFFER * * * 2 2 2 2 3 3 3 5 7 3 3 5 8 5 7 3 3 5 7 3 3 5 7 7 3 3 5 7 7 3 5 5 7 7 3 5 5 7 7 1 9 5 7 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDA STA SETOP LDX LDA STA SHOS LDX SME JSR SME JSR SMS LDD SHS LDD SHS DDCCB	2 2 BUFLEN MPROM \$E000 hardware reset #HONOP #PTAB1 #RTAB1 #RTAB1 ,X+ ,U+ #PTAB2 RST1 STACK MPROM-2 #ADE STACK MPROM-2 #BACK U #BACK U #BACK U #12 A	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start rom table start rom table start ram table setup stack pointer check for extra rom, not there. else call it put monitor return. registers.</pre>
$\begin{array}{c} 0147\\ 0148\\ 0150\\ 0151\\ 0152\\ 0153\\ 0154\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0156\\ 0166\\ 0166\\ 0166\\ 0166\\ 0167\\ 0166\\ 0167\\ 0172\\ 0172\\ 0177\\ 0177\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\ 0176\\ 0175\\$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00553 00553 00553 00553 00553 00553 00553 00553 00553 00553 0055 0055 0055 0055 0055 0055 0055 0055 0055 0055 0057 0057 0057 0057 0057 0057 0057 0057 0057 0057 0057 005 00 00 00 00 00 00 00 00 00 00 00 00	USRSTR TEMP BUFFER * * 2 2 3 3 6 6 4 3 3 6 6 4 3 3 6 6 4 3 3 5 7 7 3 5 7 7 3 5 7 7 3 5 7 7 3	RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDA STA CMPX BNE JSR LDD CMPX BNE JSR LDD SNE JSR LDD SNE SNE SNE SNE SNE SNE SNE	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP WPTAB1 %TTAB1 ,X+ ,U+ %PTAB1 %TTAB1 ,X+ ,U+ %PTAB2 RST1 STACK MPROM-2 %3755A STRT1 (MPROM-4] WBACK U %12 A STRT2	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table start copy rom. . to ram until end. . of table setup stack pointer check for. . attra rom, not there. else call it put monitor return. . onto stack put dummy. . registers</pre>
$\begin{array}{c} 0147\\ 01149\\ 0150\\ 0150\\ 0151\\ 0152\\ 0154\\ 0155\\ 0155\\ 0156\\ 0156\\ 0156\\ 01661\\ 01665\\ 01665\\ 01668\\ 01668\\ 01668\\ 01671\\ 00177\\ 001775\\ 01777$	03AA 03AC 03AE F800 F800 F800 F800 F800 F800 F800 F80	0002 0002 00052 00552 00552 00552 00550 00003 FF755 00558 000 FF755 F7755 F7755 A55A 04 9F F7755 1 F85A 04 0000 02 F8 F8 F8	USRSTR TEMP BUFFER * * 2 RESET 6 RST1 6 RST1 6 6 4 3 3 STRT1 7 3 STRT1 7 3 STRT2 2 2 3 STRT2 2 3	RMB RMB RMB ORG PUT LDA TFR SETOP LDU LDA STA CMPX BNE LDU STA LDU LDA STA CMPX BNE LDS LDX STA STA STA STA STA STA STA STA STA STA	2 2 BUFLEN MPROM \$E000 hardware reset 0HONDP 0NDP 0NDP 0NDP 0NDP 0HTAB1 0HTAB1 0HTAB1 0HTAB2 0H	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table setup stack pointer check for. . extra rom, not there, else call it put monitor return. . onto stack put dummy. . registers. . onto stack</pre>
$\begin{array}{c} 0147\\ 0149\\ 0151\\ 0152\\ 0152\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0156\\ 0161\\ 0161\\ 0162\\ 0161\\ 0167\\ 0168\\ 0167\\ 0168\\ 0177\\ 01774\\ 01776\\ 01778\\ 01788\\ 01788\\ 01788\\ 01788\\ 01788\\ 01788\\ 0$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00552 00552 00552 00552 00552 00552 00552 00562 0058 00003 FF51 F755 0058 F755 0058 F755 00 95 F755 1 F8EA 40 6090C 92 FB FA AA 603AE	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB ORG PUT LDA TFR SETO LDA LDA LDA LDA LDA STA CHPX BNE LDS LDA SHE JSR LDD SHE JSR LDD SHE STS LDA LDA LDA LDA LDA LDA LDA LDA LDA LDA	2 2 BUFLEN MPROM \$E000 hardware reset A, DP A, DP MONOP MONOP MONOP MONOP MONOP MONOP MPROM \$FTAB1 \$FT	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler nom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there. else call it put monjtor return. onto stack put dummy. registers. onto stack save stack pointer get start of buffer.</pre>
$\begin{array}{c} 0.147\\ 0.148\\ 0.150\\ 0.151\\ 0.151\\ 0.153\\ 0.154\\ 0.155\\ 0.154\\ 0.155\\ 0.156\\ 0.156\\ 0.156\\ 0.156\\ 0.156\\ 0.162\\ 0.163\\ 0.162\\ 0.163\\ 0.$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * 2 RESET 6 RST1 6 RST1 6 RST1 6 RST1 6 STRT1 7 3 STRT1 7 3 STRT1 7 3 STRT2 2 3 STRT2 2 3 STRT2	RMB RMB RMB ORG PUT UDA TFR SETOP LOV LDU LDA LDU LDA LDU LDA LDU LDA LDA LDU LDA STA STS LDD SRU PSHS LDD STS LDA STS STX	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MPTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB2 \$FST1 \$TACK MPROM-2 \$FST1 \$TACK MPROM-2 \$FST1 \$TACK \$FROM-2 \$FST1 \$TACK \$FROM-2 \$FST1 \$TACK \$FROM-2 \$FST1 \$FROM-2 \$FR	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table setup stack pointer check for. . extra rom, not there, else call it put monitor return. . onto stack put dummy. . registers. . onto stack</pre>
0147 0148 0149 0151 0152 0153 0154 0155 0155 0157 0155 0156 0157 0156 0157 0156 0157 0156 0166 0166 0162 0173 0174 0176 0178	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00555 00553 00553 00553 00553 00553 00553 0055 005 005 00 00	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB ORG PUT LDA TFR SETO LDA LDA LDA STA CHPX BNE LDS LDA STA CHPX BNE JSR LDA SHS LDD SHS STS LDD SME STS LDX	2 2 BUFLEN MPROM \$E000 hardware reset A, DP A, DP MONOP MONOP MONOP MONOP MONOP MONOP MPROM \$FTAB1 \$FT	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for extra rom, not there. else call it put monits tack put dummy. registers. onto stack save stack pointer get start of buffer. and setup.</pre>
$\begin{array}{c} 0147\\ 0148\\ 0150\\ 0151\\ 0151\\ 0152\\ 0154\\ 0155\\ 0154\\ 0155\\ 0156\\ 0155\\ 0156\\ 0156\\ 0156\\ 0156\\ 0156\\ 0161\\ 0162\\ 0163\\ 0164\\ 0161\\ 0162\\ 0163\\ 0167\\ 0172\\ 0173\\ 0172\\ 0173\\ 0172\\ 0173\\ 0176\\ 0171\\ 0178\\ 0172\\ 0173\\ 0178\\ 0172\\ 0173\\ 0178\\$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00052 00052 00052 00052 00063 FF25 000603 FF755 00060 FF551 F7755 00 FF55 F7755 04 9 F F755 1 F755 04 9 F F755 1 F755 04 9 F F755 1 F8EA 40 00052 00 5 8 9 5 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 8 9 8 9	USRSTR TEMP BUFFER * * * * 2 2 2 3 3 6 6 7 3 3 5 5 5 5 9	RMB RMB RMB RMB ORG PUT UDA TFR SETOX LDU LDA STA CMPX BME LDX CMPX BME LDX CMPX BME LDX CMPX STX STX	2 2 BUFLEN #PROM #E800 hardware reset #MONDP MONDP #PTAB1 #PTAB1 #RTAB1 ,X+ ,U+ #PTAB2 RST1 STACK MPROM-2 #STA2 STACK MPROM-2 #SA55A STR11 CMPROM-2 #SA55A STR11 CMPROM-2 #SA55A STR12 USRSTK #BOCK USRSTK #BUFFER #EADDY HEADDY	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for extra rom, not there. else call it put monits tack put dummy. registers. onto stack save stack pointer get start of buffer. and setup. pointers initialise crt controller</pre>
$\begin{array}{c} 0147\\ 0148\\ 0149\\ 0150\\ 0151\\ 0152\\ 0153\\ 0154\\ 0155\\ 0156\\ 0156\\ 0156\\ 0156\\ 0157\\ 0156\\ 0157\\ 0156\\ 0157\\ 0168\\ 0161\\ 0168\\ 0161\\ 0173\\ 0172\\ 0173\\ 0175\\ 0178\\$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00052 00052 00052 00052 00063 FF25 000603 FF755 00060 FF551 F7755 00 FF55 F7755 04 9 F F755 1 F755 04 9 F F755 1 F755 04 9 F F755 1 F8EA 40 00052 00 5 8 9 5 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 7 8 8 8 9 8 9	USRSTR TEMP BUFFER * * 2 RESET 6 RST1 6 RST1 6 6 6 RST1 6 6 7 3 STRT1 7 3 STRT1 7 3 STRT2 2 3 STRT2 3	RMB RMB RMB ORG PUT LDA LDA LDA LDA LDA LDA LDA LDA LDA LDA	2 2 BUFLEN MPROM \$E000 hardware reset 0000P 00	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for extra rom, not there. else call it put monits tack put dummy. registers. onto stack save stack pointer get start of buffer. and setup. pointers initialise crt controller</pre>
$\begin{array}{c} 0147\\ 0148\\ 0150\\ 0151\\ 0152\\ 0153\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0155\\ 0162\\ 0162\\ 0163\\ 0165\\ 0161\\ 0172\\$	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * * * 2 2 2 3 3 6 6 7 3 3 5 5 5 5 9	RMB RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDU LDA STFA CMPX BME LDX CMPX BME LDD STA STA STX STX STX STX STX STX STX STX	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 #RTAB1	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there. else call it put monitor return. onto stack put dummy. registers. exts stack pointer get start of buffer. and setup. and setup.</pre>
0147 0148 0151 0152 0154 0155 0154 0155 0156 0165 0160 0161 0162 0162 0162 0162 0161 0162 0177 0172 0174 0173 0174 0173 0177 0174 0172 0174 0173 0174 0173 0174 0173 0174 0173 0174 0173 0174 0173 0174 0173 0174 0174 0173 0174 0174 0173 0174 0174 0173 0174 0174 0173 0174 0173 0174 0174 0173 0174 0174 0173 0174 0174 0173 0190 0191 0192 0190 0191 0192 0174 0192 0190 0191 0192 0190 0191 0192 0190 0191 0192 0190 0191 0190	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * 2 2 3 3 6 6 4 3 3 6 6 8 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	RMB RMB RMB ORG PUT LDA LDA LDA LDA LDA LDA LDA LDA LDA LDA	2 2 BUFLEN MPROM #E000 hardware reset 0000P	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. . direct page tell assembler rom table start ram table start ram table start copy rom. . to ram until end. . of table setup stack pointer check for. . active rom, not there. else call it put monitor return. . onto stack put dummy. . registers. </pre>
0147 0148 0149 0150 0151 0152 0153 0154 0153 0155 0156 0155 0156 0157 0158 0157 0158 0159 0160 0161 0161 0162 0162 0163 0164 0163 0164 0165 0166 0167 0168 0168 0168 0168 0168 0169 0171 0174 0173 0174 0174 0172 0174 0174 0172 0174 0172 0174 0172 0174 0174 0172 0174	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 00052 00052 00052 00052 00052 00052 00052 00052 00052 00052 00052 00003 FF25 00050 80 00003 FF51 FF55 FF55 FF55 FF55 80 0002 0002 0002 0002 97 FB5A 0002 0002 0002 FB 80 0002 0002 FB 80 0002 0002 FB 80 0002 FF55 FF55 FF55 FF55 FF55 FF55 FF5	USRSTR TEMP BUFFER * * * 2 RESET 6 6 6 7 3 3 5 6 6 6 6 6 6 6 6 6 7 3 3 5 5 5 5 5 5 5 5 9 9 5 5 5 5 9 9 5 3 3 7	RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDU LDA STA LDU LDA STA LDU LDA STA STA STX STX STX STX STX STX STX STX STX STX	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB1 \$FTAB2 \$FT1 \$TACK MPROM-2 \$FT1 \$TACK MPROM-2 \$FT1 \$TACK \$FT1 \$FT2 \$FT1 \$FT2 \$FT2 \$FT2 \$FFER \$FER \$FFER	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom. to ram until end., of table setup stack pointer check for extra rom, not there, else call it put monitor return. onto stack put dummy. registers onto stack save stack pointer get start of buffer opinters initialise crt controller and via chips set no saved character</pre>
0147 0148 0148 0151 0152 0154 0155 0155 0155 0155 0155 0155 0155 0155 0155 0156 0155 0156 0155 0156 0156 0156 0167 0162 0162 0162 0162 0164 0165 0164 0165 0164 0165 0164 0165 0164 0165 0164 0165 0164 0165 0164 0162 0164 0162 0164 0174 0172 0172 0174 0172 0172 0174 0172 0174 0172 0174 0172 0174 0172 0174 0172 0172 0172 0174 0172 017	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0003 00052 00052 00052 00052 00052 00052 00052 00052 00052 00052 00052 00003 FFF51 FF752 00003 FF554 000 FF554 000 FBEA 40 0002 003 AE 89 87 88 87 88 87 88 87 88 87 88 87 88 87 84 40 9052 84 40 9052 84 40 9052 84 80 9052 84 80 9052 84 80 9052 84 80 9052 84 80 9052 87 88 87 84 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 88 87 80 80 80 80 80 80 80 80 80 80 80 80 80	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB RMB ORG PUT LDA LDA LDA LDA LDA LDA LDA LDA LDA LDA	2 2 BUFLEN MPROM #E000 hardware reset 0000P	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup . direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there. else call it put monifor return. onto stack put dummy. registers. </pre>
0147 0148 0151 0152 0154 0153 0154 0155 0154 0155 0155 0156 0157 0156 0157 0156 0156 0156 0156 0156 0156 0156 0162 0162 0163 0164 0161 0162 0163 0164 0167 0172 0172 0173 0177 0172 01812 018	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * * 2 RESET 6 6 6 7 3 3 5 6 6 6 6 6 6 6 6 6 7 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	RMB RMB RMB ORG PUT LDA TFR SETOP LDU LDA LDU LDA LDU LDA LDU LDA LDU LDA CMPX BNE LDS LDD STA STA STA STA STA STA STA STA STA STA	2 2 BUFLEN MPROM \$E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 \$FTAB2 \$FTAB2	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there, else call it put monitor return. onto stack put dummy. registers. and stack pointer get start of buffer. and step. opinters initialise crt controller and via chips set no saved character remove if exists then set.</pre>
0147 0148 0149 0150 0151 0152 0153 0154 0155 0155 0155 0155 0156 0157 0155 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0156 0157 0166 0167 0168 0167 0168 0167 0168 0167 0168 0167 0176 0177 0176 0177 0176 0177 0177 0176 0177 0176 0177 0176 0177 0176 0177 0176 0177 0176 0177 0176 0176 0176 0177 0176 0177 0176 0177 0176 0176 0177 0176 0177 0176 0176 0176 0177 0176 0176 0176 0176 0177 0176 0176 0176 0177 0176 0176 0176 0176 0177 0176 0180 01800	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB RMB ORG PUT LDA LTFR SETP LDX LDU LDA STA CMPX BME LDX CMPX BME LDX CMPX BME LDX CMPX BME STA STX STX STX STX STX STX STX STX STX CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR STX CLBSR CLBSR CLBSR	2 9 9 9 9 9 9 9 9 9 9 9 9 9	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler nom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for. aktra rom, not there. else call it put monitor return. onto stack put dummy. registers. onto stack save stack pointer get start of buffer. ak setup. onto stack save stack pointer get start of buffer. ak setup. pointers initialise crt controller and via chips set no saved character remove if exists then set. meg-existing allow interrupts</pre>
0147 0148 0151 0152 0154 01553 0154 01554 01556 01557 01556 01557 01556 01557 01556 01567 016159 0161200 016100 016100 016100 0161000 01610000000 01610000000000000000000000000000000000	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * 2 RESET 6 6 RST1 6 6 6 6 7 3 3 STRT1 7 3 STRT1 7 3 6 6 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	RMB RMB RMB ORG PUT LDA TFR SETOP LDX LDU STPX BME LDS LDU STPX BME LDS LDU STPX BME LDS LDU STPX SETOP LDS LDU STP STS LSS LDDA STA STA STA STA STA STA STA STA STA ST	2 BUFLEN MPROM #E000 hardware reset MONDP MONDP MONDP MONDP MPTAB1 #TAB1 *TAB2 *T	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start rom table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there, else call it put monitor return. onto stack put dummy. registers. and stack save stack pointer get start of Duffer. and setup. pointers initialise crt controller and via chips set no saved character remove if exists then set. momexisting allow interrupts</pre>
0147 0148 0148 0151 0152 0153 0154 0155 0155 0155 0157 0158 0157 0168 0167 0168 0177 0178 0177 0178 0177 0178 0177 0178 0177 0178 0177 0178 0178 0178 0177 0178 0180 0181 0180 0181 0180 01800	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB RMB ORG PUT LDA LTFR SETP LDX LDU LDA STA CMPX BME LDX CMPX BME LDX CMPX BME LDX CMPX BME STA STX STX STX STX STX STX STX STX STX CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR CLBSR STX CLBSR CLBSR CLBSR	2 9 9 9 9 9 9 9 9 9 9 9 9 9	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for extra rom, not there. else call it put monit of tack put dummy. registers. onto stack save stack pointer get start of buffer. and setup. pointers initialise crt controller and via chips set no saved character remove if exists then set. nepexisting allow interrupts send prompt. unless.</pre>
0147 0148 0151 0152 0154 0155 01554 01554 01555 01554 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01568 016169 016162 016162 016162 016162 01721 01723 0174 01772 01772 01772 0179 01723 018180 018180 018180 018190	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * * * * * * * * * * * * * * * * * *	RMB RMB RMB RMB ORG PUT LDA LDA LDA LDA STA LDA STA LDA STA LDA STA LDA STA LDA STA LDA STA LDA STA LDA STA LDA STA STA STA STA STA STA STA STA STA ST	2 9 9 9 9 9 9 9 9 9 9 9 9 9	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start rom table start copy rom. to ram until end. of table setup stack pointer check for. extra rom, not there, else call it put monitor return. onto stack put dummy. registers. and stack save stack pointer get start of Duffer. and setup. pointers initialise crt controller and via chips set no saved character remove if exists then set. momexisting allow interrupts</pre>
0147 0148 0151 0152 0154 0155 01554 01554 01555 01554 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01556 01557 01568 016169 016162 016162 016162 016162 01721 01723 0174 01772 01772 01772 0179 01723 018180 018180 018180 018190	03AA 03AC 03AC 03AC 03AC 03AC 03AC 03AC	0002 0002 00052 0052 0052 0052 0052 005	USRSTR TEMP BUFFER * * 2 2 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	RMB RMB RMB RMB ORG PUT LDA TFR SETOS LDU LDA TFR SETOS LDU LDA STM STA STA STX STX STX STX STX STX STX STX STX STX	2 9 9 9 9 9 9 9 9 9 9 9 9 9	<pre>saved user-stack pointer when user halted temporary storage line input buffer starts at this address setup. direct page tell assembler rom table start ram table start ram table start copy rom. to ram until end. of table setup stack pointer check for extra rom, not there. else call it put monit of tack put dummy. registers. onto stack save stack pointer get start of buffer. and setup. pointers initialise crt controller and via chips set no saved character remove if exists then set. nepexisting allow interrupts send prompt. unless.</pre>

			0744	_	MON1	STA	ONLINE	set no cr yet
	F857		03A1 03		PARSE	LDA	#MONDP	set no cr yet
	F850	1F	8B	6		TFR	A, DP	
0197 0198	F85E	9E	0003 75	5		SETDP	MONDP LINEPT	see if mem input,
0199	F868	27	84	З	:	BEQ	PARSEC	if so, then
	F862		84 41	4		LDA BEQ	,X MEND	see if null yet end if is a null
0202	F866	80	17	7	PARSEC	BSR	CONCHR	get input
	F868		E3	3		BEQ	MON	prompt on cr else command table
	F86A		71 0137	· 9		LDX	CMNDI DISPCH	
0206	F86F		E9	3		BRA	PARSE	
0207					* .		enter here for a	monitor to use memory input line
0209					*		(x) is start of	line, ends with a null.
0210 0211					*		multiple input with a carriage	lines are allowed, each line ends
0212					*		-	
0213 0214					*			zero if all ok, else (a) is \$FF oo early, else (a) is character
0215					*		causing error.	oo eariy, eise (a/ is character
0216			0000	_		SETOP	0	
	F871 F873		06 0380	67	MEMUSE	PSHS STS	DP MSTACK	save stack for return
0219	F877	BF	0375	6		STX	LINEPT	save pointer in memory
	F87A F87D		03A2 CE	73		CLR BRA	LASTC MON	set none saved and call monitor
0222	1010	20	UE	ت	*	BKN	nun	and call monitor
	F87F		ØD	2	CONCHR		#CR	assume or now
	F881 F884		03A1 06	73		TST BEQ	ONLINE CON1	and if found cr then is correct
8226	F886	8D	08	7		BSR	CONIN	get input
	F988 F88A		0D 03	23		CMPA BNE	#CR CON2	if not cr then . done
	F88C		03A1		CON1	CLR	ONLINE	set found cr
	F88F	39		5	CON2	RTS		
0231 0232			0003		*	SETDP	MONOP	
0233					*			
0234 0235					*		buffer or memory	outine, gets character from keyboard y, if finds null in memory then returns to
8236					*		to caller with (
0237 0238	F890	34	18	8	CONIN	PSHS	DP, X	
8239	F892	86	03	2		LDA	#MONDP	setup.
	F894 F896		88 A2	6		TFR LDA	A, DP LASTC	, direct page saved one?
0242	F898	27	04	3		BEQ	CON5	none saved
8243	F89A F89C	ØF	A2 98	6 10		CLR PULS	LASTC	not saved anymore
8245	F890	30	30	10	*	FULS	DP, X, PC	
	F89E		75		CON5	LDX	LINEPT	see if mem input
	F8A0 F8A2		96 80	3		BEQ	CON3 , X+	no, use buffer get mem value
8249	F8A4	26	ØE	3	•	BNE	CON4	ok if not null
0250	F8A6 F8A7	48	80	2	MEND	DECA LDS	MSTACK	set to \$FF
	FBAA		75	6		CLR	LINEPT	clear mem.
	FBAC		76	6		CLR	LINEPT+1	. input
0255	FBAE	30	88	8	*	PULS	DP, PC	and return to caller
0256	F880		9F Ø367		CON3	JSR	[CINADR]	get console input
	F884 F886		75 98	10	CON4	STX PULS	LINEPT DP, X, PC	store new mem pointer
0259	1000				*			
0260					*			and all contrations down the strate of an interest
0261 0262					*		to by usrstk, th	ints the registers from the stack as pointed . Nen prints usrstk itself.
0263					*		also prints 5 by	gtes starting at (pc)
8264 8263	F888	8E	FF96	з	* EXREG	LDX	#TITLES	headings.
0266	F888	8D	2F	7		BSR	STRING	printed first
0267 0269	F9BD F9BF	DE C6	AA 04	52		LDU LDB	USRSTK	is, cc, a, b, dp
	F8C1		Č0		EX1	LDA	, U+	get data
	F8C3		01D1	9		LBSR	OPARSP	output as 2 hex digits
	F8C6 F8C7		F8	23		DECB	EX1	until all 4 output
0273	F8C9	C6	64	2		LDB	#4	ie, x,y,u,pc
	FBCB		C1 0165	9	EX2	LDX LBSR	, U++ OPXREG	get 2 bytes as 4 hex digits
0276	F8D0	5A		2		DECB		-
0277	F8D1 F8D3	26	F8 AA	35		BNE LDX	EX2 USRST-	until all output then put out stack
	F805		019D	9		LIBSR	OPXIREG	then put out stack address
		EE	96	6		LDU	10, X	get user pr value
0280			FFBD	3		LDX BSR	OPCMESS STRING	send title
0281	F8DA		60	- 7				
0281 0282 0283	F8DA F8DD F8DF	8D C6	00 05	2		LDB	#5	then 5 bytes
0281 0282 0283 0284	F8DA F8DD F8DF F8E1	80 C6 A6	05 C0	2 6	EX3	LDA	, U+	then 5 bytes
0281 0282 0283 0284 0285 0285	F8DA F8DD F8DF F8E1 F8E3 F8E6	80 C6 A6 17 5A	05 C0 01B1	2692	EX3	lda LBSR DECB	, U+ OPARSP	then 5 bytes
0281 0282 0283 0284 0285 0286 0286 0287	F8DA F8DD F8DF F8E1 F8E3	80 C6 A6 17 5A	05 C0	2 6 9	EX3	lda LBSR DECB	, U+	then 5 bytes
0281 0282 0283 0284 0285 0285	F8DA F8DD F8DF F8E1 F8E3 F8E6	80 C6 A6 17 5A	05 C0 01B1	2692	EX3 *	lda LBSR DECB	, U+ OPARSP EX3	
0281 0282 0283 0284 0285 0286 0286 0287 0288	F8DA F8DD F8DF F8E1 F8E3 F8E6	80 C6 A6 17 5A	05 C0 01B1	2692	•	lda LBSR DECB	, U+ OPARSP EX3	then 5 bytes ng cr,lf to console 79

0291 E8E9 8E FF93 3 OPCRLF LDX #SCRLF string address of cr. 1f **R292** 0293 output the string pointed to by (x) until a null, leaves x pointing to null+1, other registers intact 0294 8295 6 STRING PSHS 0296 F8EC 34 02 ¥4 get data. 0297 FBEE A6 80 6 STRNG1 LDA FOTYT REO 0298 F8F0 27 05 2 0299 F8F2 17 812C ā LBSR CONOUT .. else output.. 2 BRA STRNG1 ... and repeat 0700 E8E5 20 F7 0301 F8F7 35 ē EOTXT PULS A, PC 82 oet 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 acos s 0307 F8FB D6 8F . LDB CROW **6388 FBFD 30** 11 MUL 90 ADDB CCOL 8389 F8FE DB 4 0310 F900 89 00 ż ADCA #0 CPAGE 6.11 F902 D3 91 6 ADDD 0312 F904 39 ŝ PTS 8313 9314 * calculate real address of cursor in memory space, 0315 . result returned in x 8316 0317 F905 34 7 COLOCN PSHS 86 0318 F907 8D FØ 2 BSR CCOFST apparent address 0319 F909 84 AMDA 03 ż wrap around in 1k page #%11 0320 F908 C3 ADDD 0400 4 #PAGE add start of page 6 TFR 8321 F90E 1F **Ø**1 D, X 0322 F910 35 86 ā PULS D, PC unsave and return 6727 * * **B324** initialise versatile interface adapter. 0325 0326 F912 CC EFD0 з UTAT LDD #IKPCR+256+IKIER 8327 F915 8E 0980 з LDX #KUIA point to via KPCR-KUIA, X 0328 F918 A7 ŝ STA òс peripheral control register interrupt enable register printer output (7 bits) plus, ...cassette input on 'a' side 8329 F91A E7 0E 5 STB KIER-KUIA, X 0330 F91C 86 7F 2 LDA #\$7F 0331 F91E A7 **8**3 5 STA KDORA-KUIA. X KIFR-KUIA, X 5 cancel any. 0332 F920 A6 **PD** I DA 8333 F922 A7 8334 F924 39 5 STA RTS KIFR-KUIA, X ØÐ .. interrupts present 5 0335 8336 * initialise crt controller on vdu card, r or s version 0337 0338 F925 8E FF0D 3 CRTCI LDX **#CRTCSU** table for v version test the page register..
(low byte) for read/write..
...ability **BDAA** LDD 0339 F928 CC З #PAGHI+1+256++AA 0800 STD 0340 F92B FD 65 CRTC 8341 F92E F1 CRTC+1 0342 F931 27 02 2 REO CRTC 11 if so, then v version. CRTCRU-CRTCSU,X...else get r table instead #11 setup 12 registers 8343 F933 I FAX 70 ar 5 9344 E935 86 Й₿ 2 CRTC11 LDA 0345 F937 E6 86 5 CRTCI2 LDB get data. A. X 0346 F939 FD 0800 6 STD CRTC ... and store it in correct register 8347 F93C 46 DECA 2 8348 F930 2A F8 Ξ RPI CRTC12 repeat until all 12 done 8349 0350 * eset display to a blank page with cursor at top left of screen 0351 * 0352 CLRALL CLR CROH row Ø 0353 F93F 0F 8F 6 8354 F941 8F 90 ē CLR CCOL column 0 0355 F943 OF 91 6 CLR CPAGE current page set. CPAGE+1 0356 F945 OF 92 6 CLR .. to start of display memory 0357 F947 17 009C 9 LBSR SETTOP set page in crtc chip 0358 F94A 17 9941 ā LBSR SETCUR and set cursor in crtc chip 0359 * 0360 * clear the display memory to all blanks 8361 #SPACE+256+SPACE 0362 F94D CC 2020 3 CLRSCN LDD 8063 F958 8E 9409 ž LOX **#PAGE** start of displau memoru 0364 F953 ED 8 CLRS1 STD store two blanks 81 . X++ 0365 F955 8C 0800 #PAGE+1024 and repeat. . 4 CMPX 0366 F958 26 ż .. until done all page F9 BNE CLRS1 8367 F95A 39 5 RTS 0368 -0369 * this routine used to put a character to the vdu, 8378 * handling cr, 1-, backspace, and form feed. 8371 * all registers are saved 8372 15 DISPLA PSHS 2 LOB 0373 F958 34 7F DP, CC, D, X, Y, U save registers 8374 F950 C6 83 MONOP 8375 F95F 1F 98 6 TFR B. DP 8376 F961 9E 6F 5 LDX FUNCTI get function table 0377 F963 1A 10 ž #IRQ stop interrupts since not re-entrant ORCC 8378 F965 8D 3F FF BSR DISPCH jump on function table 8379 F967 35 17 PULS DP, CC, D, X, Y, U, PC then restore and return 8388 -0381 this routine puts a character on display and moves cursor, 6382 0383 F969 8D 96 7 SINCHR BSR COLOCN find location in memory. 0384 F968 A7 84 90 .. and store character move cursor across.. STA . × 6385 F96D 0C CCOL 6 INC 0386 E96E 96 90 I DA CCOL .. then done if...

0387	F971	81	28	2		CHPA	#COLS	if all columns
	F973		ØĒ	3		BNE	SIMI	not yet filled
	F975		90	6		CLR	CCOL	do cr
0390					•			
8391					•		move cursor dow	n 1 line, scroll display if required
0392		-			*			
	F977		0F		DOLF	INC	CRON	down a row
	F979 F97B		eF .	4 2		LDA CHPA	CROM IRDNS	last row yet.
	F970		19 04	3		BNE	SIM1	if not then done
	F97F		8F	6		DEC	CRON	else back up.
	.F981		40	7		BSR	SCROLL	and scroll instead
	F983		69		SIMI	BRA	SETCUR	then set new cursor
0400					*			
0401					*		move cursor bac	k erasing last character
8482					+			
	F985		96	7	DORUB	BSR	BSONE	back up cursor
	F987		29	2		LDA	#SPACE	
	F989		00	7		BSR	DISPLA	then blank last character
	F988		82	7		BSR	BSONE	, and back up again
0407	F980	20	SF	З		BRA	SETCUR	then set cursor up
8489					*		back ourses up	allowing line and row underflow
0410					*		Dack CUIDOI OP	allowing line and low sheat tow
	F 98F	86	90	6	BSONE	DEC	CCOL	left move cursor
	F991		ØČ	- 3		BPL	BS1	no underflow.
	F993		27	2		LDA	OCOLS-1	else set to
0414	F995	97	98	4		STA	CCOL	right margin
	F997		8F	6		DEC	CRON	and up one row
	F999		04	3		BPL	BS1	no row underflow
	F99B		8F	- 6		CLR	CROW	
	F99D		90	6		CLR	CCOL	
	F99F	39		5	BS1	RTS		
0420					*			• • • • • • • • • • • •
8421					*		dispatch routin	
8422					+		character in a,	table addr i X.
0423					*			
8424					*		table format is	 first byte, number of entries 1 to 255
0425					*			
0426 0427					*		repeat for ea	ch entry- character to match with,
					*			2 byte offset from start of this
0428					*			table of routine to jump to if
0429					*			characters match.
8438					*			
0431					*		flag buter	appired if an match found :-
0432 0433					*		ting byte-	control if no match found :-
0434					*		0001 \$ 1.000	next word is offset of default routine
0435					*			
							negative-	return to calling program next word is address of another table to search
0436					*		negative-	next word is address of another table to search
0436 0437	F960	EC	84	5	*	100	negative-	next word is address of another table to search
0436 0437 0438	F960 F962		94 62				. negative-	next word is address of another table to search get address of new table.
0436 0437 0438 0439	F9A2	ED		6	*	ldd Std Puls	,X 2,S	next word is address of another table to search get address of new table replace old on stack
0436 0437 0438 0439 0440	F9A2 F9A4	ED 35	62	6 11	* DIS2	STD PULS	negative- ,X 2,S D,X,U	next word is address of another table to search get address of new table. replace old on stack and begin again.
0436 0437 0438 0439 0440 0441	F9A2 F9A4 F9A6	ED 35 34	62 56	6 11	*	STD PULS	negative- ,X 2,S D,X,U D,X,U	next word is address of another table to search get address of new table. replace old on stack and begin again save registers
0436 0437 0438 0439 0440 0441	F9A2 F9A4 F9A6 F9A8	ED 35 34 E6	62 56 56	6 11 11 7	* DIS2	STD PULS PSHS	negative- ,X 2,S D,X,U	next word is address of another table to search get address of new table. replace old on stack and begin again.
0436 0437 0438 0439 0440 0441 0442	F9A2 F9A4 F9A6 F9A8 F9A8	ED 35 34 E6 A1	62 56 56 81	6 11 11 7	т DIS2 DISPCH	STD PULS PSHS LDB	negative- ,X 2,S D,X,U D,X,U ,X++	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset
0436 0437 0438 0439 0440 0441 0442 0443	F9A2 F9A4 F9A6 F9A8 F9AA F9AC	ED 35 34 E6 A1 27	62 56 56 81 1F	6 11 11 7 5	т DIS2 DISPCH	STD PULS PSHS LDB CNPA	negative- ,X 2,S D,X,U D,X,U ,X++ -1,X	next word is address of another table to search get address of new table replace old on stack and begin again save registers' get length and move to offset compare characters
0436 0437 0438 0439 0440 0441 0442 0443 0443 0445 0445	F9A2 F9A4 F9A6 F9A8 F9AA F9AC F9AE F9B0	ED 35 34 E6 A1 27 39 54	62 56 81 1F 0B 03	6 11 11 7 5 3 5 2	т DIS2 DISPCH	STD PULS PSHS LDB CHPA BEQ LEAX DECB	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 01S3 3,X	<pre>next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. repeat until.</pre>
0436 0437 0438 0439 0440 0442 0443 0443 0445 0445 0445 0446 0447	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1	ED 35 34 E6 A1 27 30 50 26	62 56 81 1F 0B 03 F7	6 11 7 5 3 5 2 3	т DIS2 DISPCH	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1	next word is address of another table to search get address of new table. replace old on stack and begins again save registers' get length and move to offset compare characters found it, else. move down. repeat until done
0436 0437 0438 0439 0440 0442 0443 04445 0445 0446 0445 0446	F9A2 F9A4 F9A6 F9A8 F9A8 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B3	ED 35 34 E6 A1 27 39 56 60	62 56 56 81 1F 0B 03 F7 1F	6 11 11 7 5 3 5 2 3 7	т DIS2 DISPCH	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST	negative- ,X 2,S D,X,U D,X,U D,X,U ,X++ -1,X DIS3 3,X DIS1 -1,X	<pre>next word is address of another table to search get address of new tablereplace old on stack and begin again save registers' get length and move to offset compare characters found it, elsemove downrepeat untildone test flag byte</pre>
0436 0437 0438 0439 0440 0442 0443 0444 0445 04445 0446 0447 0448 0449	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B3 F9B3	ED 35 34 E6 A1 27 30 56 60 27	62 56 81 1F 98 93 F7 1F 94	6 11 11 7 5 3 5 2 3 7 3	т DIS2 DISPCH	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X DIS4	next word is address of another table to search get address of new table. replace old on stack and begins again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return
0436 0437 0438 0439 0449 0441 0442 0443 0444 0445 0446 0446 0446 0446 0446 0446	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6	ED 35 34 E6 A1 27 30 56 60 27 28	62 56 81 1F 08 03 F7 1F 0A E7	6 11 17 5 3 5 2 3 7 3 3	* DIS2 DISPCH DIS1	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ BMI	negative- ,X 2,S D,X,U D,X,U D,X,U -1,X DIS2 3,X DIS1 -1,X DIS2 DIS2 DIS2	<pre>next word is address of another table to search get address of new tablereplace old on stack and begins again save registers' get length and move to offset compare characters found it, elsemove downrepeat untildone test flag byte zero means return search new table, else.</pre>
0436 0437 0438 0439 0448 0441 0442 0443 04445 0445 0446 0445 0446 0445 0446 0445 0450 0451	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9B1 F9B3 F9B3 F9B5 F9B7 F9B9	ED 35 34 E6 A1 27 30 5A 6D 27 8C 20 20 20 20 20 20 20 20 20 20 20 20 20	62 56 56 81 1F 08 93 F7 1F 9A E7 84	6 11 11 7 5 3 5 2 3 7 3 3 5	т DIS2 DISPCH	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ BMI LDO	negative- ,X 2,S 0,X,U 0,X,U ,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS4 DIS2 ,X	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and.
0436 0437 0438 0439 0441 0442 0443 0444 0445 0446 0446 0446 0446 0445 0451 0452	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9B1 F9B3 F9B3 F9B5 F9B7 F9B9 F9B8	ED 35 34 E6 127 30 56 60 27 80 20 20 20 20 20 20 20 20 20 20 20 20 20	62 56 56 81 1F 0B 03 F7 1F 0A E7 84 62	6 11 11 7 5 3 5 2 3 7 3 3 5 7	* DIS2 DISPCH DIS1	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ BNI LDO ADDD	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 0152 3,X DIS1 -1,X 0154 DIS2 ,X 2,S	<pre>next word is address of another table to search get address of new table. replace old on stack and begins again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. add start of table, then.</pre>
0436 0437 0438 0439 0448 0441 0442 0443 0444 0445 0445 0446 0445 0448 0445 0450 0451 0453	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B9 F9B8 F9B0 F9B0	ED 354 E6 127 28 A 26 D 27 B CC 32 B C	62 56 56 81 1F 98 83 F7 1F 84 E7 84 64	6 11 17 53 52 37 33 57 6	* DIS2 DISPCH DIS1	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ BNI LDO ADOD STD	negative- ,X 2,S D,X,U D,X,U ,X,U ,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 ,X 2,S 4,S	next word is address of another table to search get address of new table , replace old on stack and begin again save registers get length and move to offset compare characters found it, else repeat until done test flag byte zero means return search new table, else get offset and add start of table, then store in stack
0436 0437 0438 0439 0440 0442 0443 0444 0443 0444 0445 0446 0446 0445 0455 0455 0455	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B7 F9B8 F9B5 F9B5 F9B5	ED 354 E6 11 27 28 A 26 D 27 B CC 32 D 55 ED 35	62 56 56 81 1F 08 03 F7 1F 0A E7 84 62 96	611175352373357611	* DIS2 DISPCH DIS1 DIS3	STD PULS PSHS LDB CMPA BEQ LEAX DECB BNE TST BEQ BNE TST BEQ BMI LDO ADDD STD PULS	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS4 DIS4 DIS4 DIS2 ,X 2,S 4,S 0,X,PC	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine
0436 0437 0438 04439 0444 0443 0444 0445 0445 0446 0447 0448 0447 0448 0447 0451 0451 0451 0452 0453 0454	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B7 F9B8 F9B5 F9B7 F9B8 F9B5	ED 354 E6 127 28 A 26 D 27 B CC 32 D 55 ED 35 ED 35	62 56 56 81 1F 98 83 F7 1F 84 E7 84 64	611175352373357611	* DIS2 DISPCH DIS1 DIS3 DIS4	STD PULS PSHS LDB CNPA BEQ LEAX DECB BNE TST BEQ BNI LDO ADOD STD	negative- ,X 2,S D,X,U D,X,U ,X,U ,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 ,X 2,S 4,S	next word is address of another table to search get address of new table , replace old on stack and begin again save registers get length and move to offset compare characters found it, else repeat until done test flag byte zero means return search new table, else get offset and add start of table, then store in stack
0436 0438 0439 0448 0442 0442 0443 04442 0443 0444 0445 0446 0446 0452 0452 0452 0455 0456	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B7 F9B8 F9B5 F9B7 F9B8 F9B5	ED 354 E6 127 28 A 26 D 27 B CC 32 D 55 ED 35 ED 35	62 56 56 81 1F 08 03 F7 1F 0A E7 84 62 96	611175352373357611	* DIS2 DISPCH DIS1 DIS3	STD PULS PSHS LDB CMPA BEQ LEAX DECB BNE TST BEQ BNE TST BEQ BMI LDO ADDD STD PULS	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 D	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller
0436 0438 0439 0448 0449 0442 0443 04443 04443 04443 04445 04448 04451 0452 0453 0455 0455 0455	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B7 F9B8 F9B5 F9B7 F9B8 F9B5	ED 354 E6 127 28 A 26 D 27 B CC 32 D 55 ED 35 ED 35	62 56 56 81 1F 08 03 F7 1F 0A E7 84 62 96	611175352373357611	* DIS2 DISPCH DIS1 DIS3 DIS4	STD PULS PSHS LDB CMPA BEQ LEAX DECB BNE TST BEQ BNE TST BEQ BMI LDO ADDD STD PULS	negative- ,X 2,S D,X,U D,X,U -1,X DIS2 3,X DIS1 -1,X DIS2 ,X 2,S 4,S D,X,U,PC scroll the disp	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. add start of table, them. store in stack restore and go to routine return to caller lay up one line, leave cursor at same
0436 0438 0439 0448 0442 0442 0443 04442 0443 0444 0445 0446 0446 0452 0452 0452 0455 0456	F9A2 F9A4 F9A6 F9A8 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B1 F9B1 F9B5 F9B7 F9B7 F9B8 F9B5 F9B7 F9B8 F9B5	ED 354 E6 127 28 A 26 D 27 B CC 32 D 55 ED 35 ED 35	62 56 56 81 1F 08 03 F7 1F 0A E7 84 62 96	611175352373357611	* DIS2 DISPCH DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CMPA BEQ LEAX DECB BNE TST BEQ BNE TST BEQ BMI LDO ADDD STD PULS	negative- ,X 2,S D,X,U D,X,U -1,X DIS2 3,X DIS1 -1,X DIS2 ,X 2,S 4,S D,X,U,PC scroll the disp	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller
0436 0438 0439 0449 0441 0442 0443 0444 0445 0444 0445 0445 0451 0452 0451 0452 0453 0454 0455 0456 0457 0458 0459	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9B1 F9B1 F9B1 F9B3 F9B5 F9B7 F9B5 F9B7 F9B5 F9B7 F9B5 F9B7 F9B5 F9B7 F9B5 F9B7 F9B5	ED 334 EA1230 A 260 27 B C 3 0 3 5 2 6 0 27 B C 3 0 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3	62 56 56 81 1F 08 03 F7 1F 0A E7 84 62 96	6 11 17 5 3 5 2 3 7 3 3 5 7 6 1 13	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CMPA LEAX DECB BME TST BEQ BME TST BEQ BMI LD0 ADD0 STD PULS PULS	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 01S3 3,X DIS1 -1,X 01S4 DIS4 DIS4 DIS4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,U,PC scroll the disp position on scr	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact
0436 0437 0438 0439 0441 0443 0444 0445 0445 0445 0445 0445 0445	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9A7 F9B3 F9B3 F9B3 F9B5 F9B5 F9B5 F9B5 F9B5 F9B5 F9B5 F9B5	ED 354 E A17 36 A 260 77 B CC 3 5 5 5 DC	62 556 81 F BB 8 F F F BB 8 F F F BB 8 56 56 9 56 9 56 9 56 9 56 9 56 9 56 9	6 11 17 5 3 5 2 3 7 3 3 5 7 6 1 13	* DIS2 DISPCH DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CNPA BEQ BEQ BME TST LDD ADDD STD PULS PULS	negative- ,X 2,S D,X,U D,X,U ,X,U ,X,U ,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 ,X 2,S D,X,PC D,X,PC scroll the disp position on scr CPMOE	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page.
0436 0437 0438 0439 0441 0442 0443 0444 0445 0445 0445 0445 0445 0455 045	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6	ED 354 E A17 20 A 26 D 7 B C 3 D C 3 D C 3	62 556 81 198 93 77 196 27 46 26 46 26 96 D6	6 11 17 5 3 5 2 3 7 3 3 5 7 6 1 1 3 5	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CMPA LEAX DECB BME TST BEQ BME TST BEQ BMI LD0 ADD0 STD PULS PULS	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 01S3 3,X 01S1 -1,X 01S4 01S4 01S4 01S4 01S4 01S4 01S5 0,X,VC 0,X,VC 0,X,V,PC scroll the disp position on scr CPAGE eROMS+COLS ePAGE	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address
0436 0437 0438 0439 0441 0443 0444 0445 0445 0445 0445 0445 0445	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6 F9A6	ED 354 E A17 20 A 26 D 7 B C 3 D C 3 D C 3	62 556 81 F ØB 90 57 F F ØA 284 62 96 D6 91 8358	6 11 11 7 5 3 5 2 3 7 3 3 5 7 6 11 3 5 4	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CNPA BEQ BEQ BMI LDD STD PULS LDD ADDD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS4 DIS4 DIS4 DIS2 ,X 2,S 0,X,PC 0,X,U,PC scroll the disp position on scr CPMGE eROMS+COLS	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address
0436 0437 0438 0439 0441 0442 0443 0444 0445 0445 0445 0445 0445 0455 045	F9A2 F9A4 F9A6 F9A8 F9A7 F9B8 F9B8 F9B8 F9B8 F9B8 F9B8 F9B8 F9B8	ED 3346 A1730 5260 728 CE 35 3 DC 38 CE	62 556 81 F 08 63 F7 F7 F7 84 62 64 96 D6 91 83E8 80488 80488 80488 804888	6 11 17 5 3 5 2 3 7 3 3 5 7 6 11 3 5 4 3	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 *	STD PULS PSHS LDB CMPA BEQ BEQ BHI LEAX BNE BEQ BHI LDO ADDD STD PULS PULS PULS	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 01S3 3,X 01S1 -1,X 01S4 01S4 01S4 01S4 01S4 01S4 01S5 0,X,VC 0,X,VC 0,X,V,PC scroll the disp position on scr CPAGE eROMS+COLS ePAGE	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address
0436 0437 0438 0448 0443 04443 04443 04443 04443 04443 04443 04449 04451 04453 04551 04553 04554 04553 04556 04558 04556 04559 04561 04563 04664 04664 04664	F9A2 F9A4 F9A6 F9A6 F9A6 F9A6 F9A6 F9B0 F9B0 F9B0 F9B0 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7	ED 3534 E6 A1 239 5A 26 D 27 28 C 23 25 DC 3 8E C 298 E 23 25 DC 3 9E C 298 E	62 556 81 F 08 63 F7 F7 F7 64 64 64 62 64 64 06 04 83 E8 8488 8488 8488 8488 8488 8488 84	6111753523733576113 543342	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 *	STD PULS PSHS LDB ECAPA BEQ LEAX DECB BMC LEAX DECB BMC STD PULS STD PULS LDD LDD LDV ANDA	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 01S3 3,X DIS1 -1,X DIS3 3,X DIS1 -1,X DIS4 DIS4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,V,PC 0,X,U,PC scroll the disp position on scr CPAOE eROUS+COLS ePAOE eSPACE=236+SPACI eC0LS/2	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until.
0436 0437 0438 04438 04442 04443 04442 04442 04442 04442 04453 04455 04453 04555 04553 04555 04559 04559 04560 04564 04560 04550 04560 04550 04560 04550 04560 04550 04550 04560 04550 04560 04550 04560 04550 04560 04550 04560 04550 04560 04550 04550 04550 04550 04560 045500000000	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9B1 F9B1 F9B1 F9B3 F9B5 F9B5 F9B5 F9B5 F9C1 F9C3 F9C5 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8	ED 35 34 E6 1 27 39 5 26 60 27 28 EC 33 35 DC 38 EC 198 EF	62 556 81 F 88 F F F F 84 64 90 81 88 88	6111753523733576113 5433429	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PSHS LDB CMPA BEQ CMPA BEQ BEQ BHI LDA STD LDD ADDD LDX LDY ANDA STU	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 01S3 3,X DIS1 -1,X DIS3 3,X DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and store in stack restore and go to routine return to caller lay up one line, leave cursor at same ean, leave registers intact get start of page. then move address E double space number to blank wrap around on 1k and put 2 blanks
0436 0437 0438 0448 0443 04443 04443 04443 04443 04443 04443 04449 04451 04453 04551 04553 04554 04553 04556 04558 04556 04559 04561 04563 04664 04664 04664	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9B1 F9B1 F9B1 F9B3 F9B5 F9B5 F9B5 F9B5 F9C1 F9C3 F9C5 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8	ED 35 34 E6 1 27 39 5 26 60 27 28 EC 33 35 DC 38 EC 198 EF	62 556 61 F BB 03 F7 F7 B4 62 64 96 D6 91 03E0 90400 20200 0014 88 80 8022	6111753523733576113 54334294	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PSHS LDB ECMPA BEQ LEAX DECB BNE LEAX DECB BNE LEAX DECB BNE LEAX DECB BNE LEAX LDD ADDD ADDD LDV ADDD STD LDV ADDD STD LDV ADDD STD LDV ADDD STD STD ADDD ADDD ADDD STD STD ADDD ADD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0153 3,X DIS1 -1,X DIS3 3,X DIS1 -1,X DIS4 DIS4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,V,PC 0,X,U,PC scroll the disp position on scr CPAOE eROUS+COLS ePAOE eSPACE=236+SPACI eC0LS/2	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. repeat until.
0436 0437 0438 0439 0440 04412 0442 0442 0442 0442 0442 044	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9B3 F9B3 F9B3 F9B3 F9B3 F9B5 F9B3 F9B5 F9B5 F9B5 F9B5 F9C1 F9C3 F9C3 F9C3 F9C3 F9C3 F9C3 F9C4 F9C4 F9C4 F9C4 F9C4 F9C4 F9C4 F9C4	ED 35 34 E6 1 27 39 5 26 D 27 28 E C 3 3 5 5 5 C 3 8 E C 3 1 28 C	62 556 81 F 08 03 F 7 F F 64 64 90 00 00 00 00 00 00 10 88 00 90 14 88 00 90 28 56 91 91 88 80 20 91 93 56 91 91 91 91 91 91 91 91 91 91 91 91 91	6111753523733576113 543342945	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PSHS LDB EQ CMPA BEQ ECAPA BEQ BHI LDA DECB TST BEQ BHI LDA DECB TST BEQ BHI LDA DECB STD LDA DECB STD LDA DECB STD LDA STD LDA DECB STD STD LDA DECB STD STD LDA DECB STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 01S1 -1,X 01S3 3,X 01S1 -1,X 01S4 01S4 01S4 01S4 01S4 01S4 01S4 01S4	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller ley up one line, leave cursor at same een, leave registers intact get start of page. then move address E double space number to blank wrap around on ik and put 2 blanks move up and. repeat until
0436 0437 0438 0439 0439 0440 0440 0440 0440 0440 0440	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9B0 F9B0 F9B0 F9B0 F9B7 F9B0 F9B7 F9B0 F9B7 F9C1 F9C2 F9C2 F9C2 F9C2 F9C2 F9C2 F9C2 F9C2	ED3534E617239526D728CE35550038EE1984FC3326	62 556 61 F 08 0 7 F F F 04 0 62 64 96 0 8 8 8 62 64 96 0 91 0 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	6111753523733576113 5433429453	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PSHS LDB ECAPA BEQ DECB BNE LEAX DECB BNE DECB BNE LEAX DECB BNE LEAX LDD STD STD STD STD STD STD STD STD LDA ADDO STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 2,S 0,X,V 0,X 1,X 0,X 1,X 0,X,V 0,X 0,X,V 0,X 0,X 0,X,U 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. move down. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. tine done
0436 0437 0438 0439 0440 0441 0442 0442 0442 0442 0442 0444 0445 0444 0445 0444 0445 0444 0451 0445 0444 0455 0445 0455 045	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 35 34 E6 1 27 39 5 26 60 7 28 EC E ED 35 35 CC 28 EC E 10 4 FC 3 12 63	62 556 81 F 08 07 F7 F7 F7 F7 84 64 64 64 90 06 91 08 88 80 90 92 92 92 92 93 F7 F5 F5 63 E8 90 92 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7	6111753523733576113 54334294534	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PSHS LDB EQ PSHS LDB EQ PSHS BEG BHI LDD ECS BHI LDD BHI LDD BHI LDD LDD LDD LDD LDD STD ADDO LDD STD STD ADDO STD STD SUBO	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 01S3 3,X DIS1 -1,X 01S3 3,X DIS1 -1,X 01S4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,U,PC 5croll the disp position on scr CPAGE eSPACE=236+SPACI eCOLS/2 eX11 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. .then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. .repeat until. .ine done move time now
0436 0437 0438 0439 0448 0449 0442 0442 0442 0442 0442 0442 0442	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9B0 F9B0 F9B0 F9B1 F9B5 F9B5 F9B5 F9B5 F9B5 F9B5 F9C1 F9C2 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8 F9C8	ED3534 E6127395260728EE3ED3535 DC38EE808 E6127395260728EE3ED3535 DC38EE808 E6127326384	62 556 81 F 88 87 F F 88 64 90 90 91 88 80 80 80 80 80 80 80 80 80 80 80 80	6111753523733576113 543342945342	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PULS CHPA BEQ ECAX BECAX DEC6B BNE ECAX DEC6B BNE TST BEQ TST BEQ TST BEQ TST BEQ PULS LD0 D CMPA BEG X DEC6B STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wore up and. tine done move to second line now wrap around and.
0436 0437 0438 0439 0440 0442 0442 0442 0442 0442 0442 044	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B0 F9B0 F9B0 F9B0 F9B0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C	ED 334 E61 279 56 260 27 28 CE 32 DC 38 E CE 98 E CE 33 35 DC 38 E CE 98 E CE	62 556 81 F 08 F7 F1 F 09 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1	6111753523733576113 5433429453425	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PULS LDB CMPA BEQ CMPA BEQ BEQ BNE TST BENI LDO ADDO STD LDU LDU LDU LDU LDU LDU LDU LDU STD STD	negative- ,X 2,S 0,X,U 0,X,U ,X++ -1,X 01S3 3,X DIS1 -1,X DIS3 3,X DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4 DIS4	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. .then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. .repset until. .ist an ew page
0436 0437 0438 0449 04471 0472	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B0 F9B0 F9B0 F9B0 F9B0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C	ED 334 E61 279 56 260 27 28 CE 32 DC 38 E CE 98 E CE 33 35 DC 38 E CE 98 E CE	62 556 81 F 88 87 F F 88 64 90 90 91 88 80 80 80 80 80 80 80 80 80 80 80 80	6111753523733576113 543342945342	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PULS CHPA BEQ ECAX BECAX DEC6B BNE ECAX DEC6B BNE TST BEQ TST BEQ TST BEQ TST BEQ PULS LD0 D CMPA BEG X DEC6B STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X 0,X	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wore up and. tine done move to second line now wrap around and.
0436 0437 0438 0439 0449 0442 0442 0442 0442 0442 0442 044	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B0 F9B0 F9B0 F9B0 F9B0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C	ED 334 E61 279 56 260 27 28 CE 32 DC 38 E CE 98 E CE 33 35 DC 38 E CE 98 E CE	62 556 81 F 08 F7 F1 F 09 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1	6111753523733576113 5433429453425	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PULS LDB CMPA BEQ CMPA BEQ BEQ BNE TST BENI LDO ADDO STD LDU LDU LDU LDU LDU LDU LDU LDU STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 01S1 -1,X 01S3 3,X DIS1 -1,X 01S4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,V,PC 0,X,U,PC scroll the disp position on scr CPAGE eROM5+COLS ePAGE eX11 0,X 02 2,1 97 01 2,5 4,5 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until.
0436 0437 0438 0439 0439 0440 0442 0442 0442 0442 0442 0442 044	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B0 F9B0 F9B0 F9B0 F9B0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C	ED 334 E61 279 56 260 27 28 CE 32 DC 38 E CE 98 E CE 33 35 DC 38 E CE 98 E CE	62 556 81 F 08 F7 F1 F 09 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1 F1	6111753523733576113 5433429453425	* DISPCH DISPCH DIS1 DIS3 DIS4 * * \$CROLL	STD PULS PULS LDB CMPA BEQ CMPA BEQ BEQ BNE TST BENI LDO ADDO STD LDU LDU LDU LDU LDU LDU LDU LDU STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 01S1 -1,X 01S3 3,X DIS1 -1,X 01S4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,V,PC 0,X,U,PC scroll the disp position on scr CPAGE eROM5+COLS ePAGE eX11 0,X 02 2,1 97 01 2,5 4,5 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2 0,2	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. .then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. .repset until. .ist an ew page
0436 0439 0439 0439 0440 0442 0442 0442 0442 0442 0442 044	F9A2 F9A6 F9A6 F9A6 F9A6 F9A7 F9B1 F9B1 F9B1 F9B1 F9B1 F9B7 F9B7 F9B7 F9B7 F9B7 F9C1 F9C3 F9C3 F9C3 F9C5 F9C4 F9C6 F9C6 F9C6 F9C6 F9C6 F9C6 F9C7 F9C6 F9C7 F9C6 F9C7 F9C6 F9C7 F9C7 F9C7 F9C7 F9C7 F9C7 F9C7 F9C7	ED 334 E6 A1 279 56 26 D27 28 E C2 35 35 DC 38 E C4 10 27 28 C C2 8 C C2	62 556 81 81 86 86 87 7 1 80 86 86 86 80 80 80 80 80 80 80 80 80 80 80 80 80	611175352373357613 54334294534257	* DIS2 DISPCH DIS1 DIS3 DIS4 * * * SCROLL SCR1	STD PLLS PSHS LDB EQ HPA BEQ HEAX DECB BHI LEAX DECB BHI ADDO ADDO STD LDU LDU LDU LDU LDU LDU LDU LDU LDU STD BSR STD BSR	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0153 3,X DIS1 -1,X DIS3 3,X DIS4 DIS2 ,X 2,S 0,X,PC 0,X,U,PC scroll the disp position on scr CPAGE SCPACE	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. move down. repeat until. repeat until.
0436 0437 0438 0449 0437 0445 04464 0445 0442 0445 0442 0445 0442 0445 0445	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B0 F9B0 F9B0 F9B0 F9B0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C0 F9C	ED 334 E6 127 278 56 27 28 EC 32 57 C 38 EC 98 EC 312 88 C 312 C 312	62 556 81 F 08 03 F7 F7 F7 F7 84 62 64 96 0400 0400 0400 0400 0400 0400 03 F F5 63 E8 03 F F5 91 08 92 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	611175352373357613 54334294534257 5	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PSHS LDB BEQ CMPA BEQ ECAPA BEQ BHI LEAX DECS BHI LEAX DECS BHI LEAX DECS BHI LOD ADDO LDX LDY ANDA STD BR LCAY ANDA STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,U,V,PC 0,X,U,P	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until repeat until done test flag byte zero means return search new table, else get offset and add start of table, then store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on 1k and put 2 blanks move to second line now wrap around and. stat as new page put cursor back in position s into crt controller get page
0436 0437 0438 0439 0439 0440 0440 0440 0440 0440 0440	F9A2 F9A6 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 3 3 4 6 6 1 27 20 6 27 20 E E B 3 3 3 5 6 6 27 20 E E B 3 3 3 5 C 3 8 E E C 3 1 2 6 3 8 4 D 0 8 0 9 E 6 6 1 2 7 8 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	62 556 61 F BB 02 F F F F B4 64 64 96 C 91 88 80 90 91 88 90 91 80 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	6111753523733576113 54334294534257 52	* DIS2 DISPCH DIS1 DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PULS CHPA BEQ ECAX DECB BNE ECAX DECB BNE TST BEQ TST BEQ TST BEQ TST BEQ TST BEQ ADDO STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0,X,U 0153 3,X DIS1 -1,X DIS3 3,X DIS4 DIS2 ,X 2,S 0,X,PC 0,X,U,PC scroll the disp position on scr CPAGE sCPAGE sCPAGE sCPAGE sCPAGE sCCLS cCPAGE cCPAGE sCCLS cCPAGE cCPAGE sCCLS cCPAGE sCCLS cCPAGE sCCLS cCPAGE cCP	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. ist as new page put cursor back in position s into crt controller get page point to page start register
0436 0437 0438 0439 0440 0442 0442 0442 0442 0442 0442 044	F9A2 F9A6 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 3 3 4 6 6 1 27 20 6 27 20 E E B 3 3 3 5 6 6 27 20 E E B 3 3 3 5 C 3 8 E E C 3 1 2 6 3 8 4 D 0 8 0 9 E 6 6 1 2 7 8 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	62 556 81 F 08 03 F7 F7 F7 F7 84 62 64 96 0400 0400 0400 0400 0400 0400 03 F F5 63 E8 03 F F5 91 08 92 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	611175352373357613 54334294534257 5	* DISPCH DISPCH DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PSHS LDB BEQ CMPA BEQ ECAPA BEQ BHI LEAX DECS BHI LEAX DECS BHI LEAX DECS BHI LOD ADDO LDX LDY ANDA STD BR LCAY ANDA STD STD STD STD STD STD STD STD STD STD	negative- ,X 2,S 0,X,U 0,X,U 0,X,U ,X++ -1,X 0IS3 3,X DIS1 -1,X 0IS3 3,X DIS1 -1,X 0IS4 DIS2 ,X 2,S 4,S 0,X,PC 0,X,U,V,PC 0,X,U,P	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until repeat until done test flag byte zero means return search new table, else get offset and add start of table, then store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on 1k and put 2 blanks move to second line now wrap around and. stat as new page put cursor back in position s into crt controller get page
0436 0437 0438 0449 0437 0438 0449 0449 0449 0449 0449 0449 0449 044	F9A2 F9A6 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 3 3 4 6 6 1 27 20 6 27 20 E E B 3 3 3 5 6 6 27 20 E E B 3 3 3 5 C 3 8 E E C 3 1 2 6 3 8 4 D 0 8 0 9 E 6 6 1 2 7 8 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	62 556 61 F BB 02 F F F F B4 64 64 96 C 91 88 80 90 92 80 80 90 91 80 90 80 91 80 80 91 80 80 80 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	6111753523733576113 54334294534257 52	* DISPCH DISPCH DIS1 DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PULS CHPA BEQ ECAX DECB BNE ECAX DECB BNE TST BEQ TST BEQ TST BEQ TST BEQ TST BEQ ADDO STD STD STD STD STD STD STD STD STD STD	negative= ,X 2,S D,X,U D,X,U J,X,U J,X,U J,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 J,X 2,S D,X,U PC Scroll the disp position on scr CPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SETCUR put page addres: CPAGE SETCUR	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blenks move up and. stat a new page put cursor back in position s into crt controller get page point to page start register then enter parauters
0436 0437 0438 0439 0449 0442 0442 0442 0442 0442 0442 044	F9A2 F9A6 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 3 3 4 6 6 1 27 20 6 27 20 E E B 3 3 3 5 6 6 0 7 20 E E B 3 3 3 5 C 3 8 E E C 3 1 2 6 3 8 4 D 0 8 0 9 E 6 6 1 2 7 8 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	62 556 61 F BB 02 F F F F B4 64 64 96 C 91 88 80 90 92 80 80 90 91 80 90 80 91 80 80 91 80 80 80 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	6111753523733576113 54334294534257 52	* DISPCH DISPCH DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PULS CHPA BEQ ECAX DECB BNE ECAX DECB BNE TST BEQ TST BEQ TST BEQ TST BEQ TST BEQ ADDO STD STD STD STD STD STD STD STD STD STD	negative= ,X 2,S D,X,U D,X,U J,X,U J,X,U J,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 J,X 2,S D,X,U PC Scroll the disp position on scr CPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SETCUR put page addres: CPAGE SETCUR	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. ist as new page put cursor back in position s into crt controller get page point to page start register
0436 0437 0438 0449 0448 0447 0478 0448 0445 0447 0448 0455 0456 0457 0457 0456 0457 0456 0457 0457 0456 0457 0455 055 055 055 055 055 055 055 055 05	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 334 E6 A1 27 39 56 26 D2 28 C 13 25 5 C 13 8 E C 198 4 E C 31 26 8 3 4 0 0 9 8 6 29 28 C 198 4 E C 31 26 8 3 4 0 0 9 8 6 29 28 28 29 28	62 556 81 81 80 80 87 71 84 64 90 80 80 80 80 80 80 80 80 80 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	611175352373357613 54334294534257 523	* DISPCH DISPCH DIS1 DIS3 DIS3 SCR0LL SCR1	STD PULS PSHS LDB BEQ CMPA BEQ ECAS BED TST BEQ TST BEQ TST BEN BMI LDD BME DCCB TST BEN TST BEN BMI LDD ADDD LDX LDY ANDD STD STD S BEN S TST BEQ LDS ECAS STD S DCCB TST BEQ S DCCB TST BEQ S DCCB TST BEQ S DCCB TST BEQ S DCCB TST BEQ S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB TST S DCCB S DCCB TST S DCCB S S DCCB S DCCB S DCCB S DCCB S DCCB S S DCCB S DCCB S DCCB S DCCB S DCCB S S DCCB S S S S S S S S S S S S S S S S S S	negative- ,X 2,S D,X,U D,X,U D,X,U D,X,U D,X,U D,X,U D,X,U D,X,U D,X,U D,X,U DIS1 -1,X DIS3 3,X DIS4 DIS2 ,X 2,S D,X,VC D,X,VC D,X,VC D,X,V,PC Scroll the disp position on scr CPAGE SPAGE SPAGE SPAGE SPAGE SETCUR put page addres: CPAGE SETPAR do a carriage r	next word is address of another table to search get address of new table. replace old on stack and begin again save registers get length and move to offset compare characters found it, else. repeat until. repeat until. repeat until. done test flag byte zero means return search new table, else. .get offset and. .store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blenks move up and. stat a new page put cursor back in position s into crt controller get page point to page start register then enter parauters
0436 0437 0438 0439 0449 0442 0442 0442 0442 0442 0442 044	F9A2 F9A4 F9A6 F9A6 F9A6 F9A7 F9A7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B7 F9B	ED 334 E6 A1 27 39 56 26 D2 28 C 13 25 5 C 13 8 E C 198 4 E C 31 26 8 3 4 0 0 9 8 6 29 28 C 198 4 E C 31 26 8 3 4 0 0 9 8 6 29 28 28 29 28	62 556 61 F BB 02 F F F F B4 64 64 96 C 91 88 80 90 92 80 80 90 91 80 90 80 91 80 80 91 80 80 80 80 91 80 80 80 80 80 80 80 80 80 80 80 80 80	611175352373357613 54334294534257 523	* DISPCH DISPCH DIS1 DIS1 DIS3 DIS4 * * SCR0LL SCR1	STD PULS PULS CHPA BEQ ECAX DECB BNE ECAX DECB BNE TST BEQ TST BEQ TST BEQ TST BEQ TST BEQ ADDO STD STD STD STD STD STD STD STD STD STD	negative= ,X 2,S D,X,U D,X,U J,X,U J,X,U J,X++ -1,X DIS3 3,X DIS1 -1,X DIS4 DIS2 J,X 2,S D,X,U PC Scroll the disp position on scr CPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SPAGE SETCUR put page addres: CPAGE SETCUR	next word is address of another table to search get address of new table. replace old on stack and begin again save registers' get length and move to offset compare characters found it, else. move down. repeat until. done test flag byte zero means return search new table, else. get offset and. store in stack restore and go to routine return to caller lay up one line, leave cursor at same een, leave registers intact get start of page. then move off end actual memory address E double space number to blank wrap around on ik and put 2 blanks move up and. stars new page put cursor back in position s into crt controller get page point to page start register then enter parameters eturn by setting column to 0

8496 * set cursor register in crt controller * 0485 0486 get cursor address, no wrap around FERO 9 SETCUR LBSR CCOFST 0487 F9EE 17 TFR n. x into x 0488 F9F1 1F 01 6 #CURHI point to cursor register LDA 0489 F9F3 86 ØF 2 0490 put 2 byte value into crt controller, value in X, 8491 high byte register number in a Q492 8493 0494 F9F5 34 0495 F9F7 35 10 7 SETPAR PSHS v get high byte PILS B 04 6 set register and data STD CRTC 0496 F9F9 FD 0000 6 move to low byte INCA 2 0497 F9FC 40 6 PULS R get low byte 64 0498 E9ED 35 2499 F9FF FD 0690 STD CRTC set register and data 6 š RTS 9500 FAR2 39 0501 this routine puts keyboard input into line buffer 0502 if no room then ignores character, else echoes to display * 0503 if echo is switched on handles rubout unless line buffer is switched off. 0504 * 0505 ىد 0506 0507 FA03 D6 60 4 HAVCHR LDB ECHOE HAUZ buffer off, so no rubout BPL 0508 FA05 2A 04 7F з 0509 FA07 5 CMPA #RUBCH 81 BEQ BSP1 go de rubout 0510 FA09 27 23 ž 0511 FA0B 8D 37 7 HAU3 BSR PLITCHR put into buffer no room, do not echo BEQ PUT1 0512 FA0D 27 43 2 2 TSTB if buffer off, then no.. 8513 FA8F 5D . line feed on cr input Z BPL HAU4 0514 FA10 2A Ø8 0515 FA12 81 ž CMPA #CR ØD KBECHO not cr, so no.. echo of a line.. 0516 FA14 26 0517 FA16 8D BNF 86 37 BSR KBECHO **P**4 04 . feed 2 LDA #L.F 0518 FA18 86 HEADST move static pointer up 5 HAV4 STX 9519 FALA 9F 87 0520 put character in a to console output if echo on 6521 ب 0522 ECHOF see if echo on need bit 6 KRECHO LOB 0523 FA1C D6 60 4 LSLB 0524 FA1E 56 0525 FA1F 2A 2 PUT1 not on if zero 3 BPL 31 8526 console output routine, sends to printer also **05**27 0528 if switched on . 0529 6530 0000 SETDP ß PFLAG if printer off then 7 CONOUT TST 0531 FA21 7D 0361 CANOP 1 value is zero 04 BEQ 0532 FA24 27 з 0533 FA26 AD 9F 036D 12 [PRINTI] else call printer then console output JSR 0534 FA2A 6E 9F 0365 8 CANOP1 JMP [COPADR] 0535 0003 SETDP MONDP 0536 end of line, if at. 0537 FA2E 9E 5 BSP1 LDX HEADDY 89 ... start of line. HEADST 0538 FA30 9C 87 6 CMPX , then nothing to rubout 0539 FA32 27 BEQ PHIT 1 1E 3 CMPX #BUFFER do cyclic. 02AF 0540 FA34 8C 4 BNE BSP2 0541 FA37 26 67 2 BUFLEN, X .. decrement of pointer 88 52 LEAX 0542 FA39 30 5 BSP2 LEAX 9543 FA3C 7.0 1F 5 -1, X STX HEADDY set new end of line 0544 FA3E 9F 89 5 0545 FA40 96 5F 4 LDA BSECHO and echo a backspace. 0546 FA42 20 08 3 RPA VRECHO 0547 * × 0548 0549 put character into buffer if room, return z=0, * else return z=1 0550 * 0551 * 89 5 PUTCHR LDX HEADDY get pointer 9552 FA44 9E 7 BSR BUMPU if no room, then 0553 FA46 80 **0**B 0554 FA48 9C 88 6 CMPX TAIL ...pointers equal... 0555 FA4A 27 06 з BEQ PUT1 . so done store it and.. .. set new pointer 0556 FA4C A7 84 4 STA . × 0557 FA4E 9F 89 5 STX HEADEY #255-2FR0 0558 FA50 10 FB ž ANDCC 5 PUT1 0559 FA52 39 RTS 0560 0561 * cuclic increment of buffer pointers 0562 0563 FA53 30 5 BUMPU LEAX Ø1 1. X **#BUFFER+BUFLEN** 0564 FA55 8C 0400 4 CMPX 0565 FA58 26 03 з BNE ANRTS 0566 FA5A 8E 93AE 3 אח ו #RI IFFFR 5 ANRTS 8567 FASD 39 RTS 8568 * get character from buffer, if none then clears interrupt 0569 * mask and waits, all registers saved, including cc 0570 * 0571 * 0000 0572 0573 FA5E 34 SETDE 0 8 GETCHR PSHS čc, x save and. . 11 0574 FA60 20 BRA GETCH1 ... skip wait 02 з wait for an interrupt get tail pointer #255-TR0 0575 FA62 30 FF 20 GETCH2 CWAI 038R 0576 F664 BF 6 GETCH1 LDX TAIL if equals static head ... then no characters 0577 FA67 BC 0387 7 CMPX HEADST 0578 FA6A 27 F6 E5 BEQ GETCH2 37 0529 FA6C 8D RSP BUMP11 .. else move up 0580 FASE AS 1 DA .. and get it , X 84 4 '82

0581 FA70 BF	0388	6	STX	TAIL	set new tail
0582 FA73 35 0583		0	PULS	CC, X, PC MONDP	
0584	0005	*	36.707		
0385 0586		*		all registers a	ter as 4 hex digits to console, saved.
0587 0588 FA75 34	06	7 OPXREG	DCUC	0	
0589 FA77 1F	10	6	TFR	X, D	
0590 FA79 8D 0591 FA7B 1F		7 6	BSR TFR	opareg B, A	
0592 FA7D 8D	18	7	BSR	OPARSP	
0593 FA7F 35 0594	86	9 *	PULS	D, PC	
0595 8596		*		output 'a' regi all registers s	ster as 2 hex digits to console,
0597		*			even except e
0598 FA81 34 0599 FA83 44		6 op ar eg 2	PSHS LSRA	A	
0600 FA84 44 0601 FA85 44		2 2	LSRA		
0602 FA86 44		2	LSRA LSRA		
0603 FA87 BD 0604 FA89 A6		7 6	BSR LDA	HEXOUT , S+	left nibble get byte back and .
0605 FA3B 84		2	ANDA	#15	. do right nibble
0606 0607		*		output a as a s	single hex digit
0608 0609 FABD 8B	30	2 HEXOUT	ADDA	*'9	
0610 FA8F 81	39 .	2	CMPA	#19	
0611 FA91 23 0612 FA93 8B		3	BLS ADDA	HEX2 #'A-'9-1	
0613 FA95 20 0614		3 HEX2	BRA	CONOUT	output and return
0615		*		output a hex di	git followed by a space
0616 0617 FA97 8D	E8	7 OPARSP	BSR	OPAREG	
0618 FA99 86	20	2	LDA	#SPACE	
0619 FA9B 20 0620	F8 -	3 *	BRA	HEX2	
0621 0622		*		printer routine parallel interf	b) this interfaces to anadex or centronics are printers.
9623	0000		SETOP	0	
0624 0625 FA9D 34	46	* 9 PRINT	PSHS	D, U	
0626 FA9F CE 0627 FAA2 B1		3	ldu Cmpa	#KUIA PNEW	point to via
8628 FAA5 27	0E 3	3	BEQ	PEXIT	if specid symbol then . do not send
0629 FAA7 E6 0630 FAA9 28		5 PWAIT1 3	LDB BMI	KORA2-KVIA,U PWAIT1	check if busy if so then wait
0631 FAAB A7	41	5	STA	KORA-KVIA, U	store data
0632 FAAD C6 0633 FAAF E7	4C !	2 5	LDB STB	#IKPCR-PSTRB KPCR-KVIA, U	then low strobe.
0634 FAB1 C6 0635 FAB3 E7		2 5	LDB STB	#IKPCR KPCR-KVIA, U	., and high . strobe
0636 FAB5 35	C6 1	PEXIT	PULS	D, U, PC	
0637 0638	0003	*	SETDP	MONDP	
0639 0640		*			e the decision tables for the and change function.
0641	~	* .			
0642 FAB7 0643 FAB8	08 56	MTABA	FCB FCB	MTABAE-MTABA/3-	
0644 FAB9 0645 FABB	002C 47		FDB FCB	Vaddr-Mtaba 'G	modify break address and memory
0646 FABC	0034		FDB	GADDR~MTABA	modify go address and memory
0647 FABE 0648 FABF	50 003C		FCB FDB	'P PADDR-MTABA	modify proceed address and memory
0649 FAC1 0650 FAC2	52 0042		FCB FDB	'r Raddr-Mtaba	modify register locations
0651 FAC4			FCB	SPACE	
0652 FAC5 0653 FAC7	20				
9654 FAC8 9655 FACA	20 0051 2C		FDB FCB	SPACEA-MTABA COMMA	
0633 FRUN	0051 2C 007A		FCB FDB	Comma Commaa-Mtaba	
0656 FACB	0051 2C 007A 3B 008A		FCB FDB FCB FDB	Comma Commaa-Mtaba SEMIC SEMICA-MTABA	
9657 FACD	0051 2C 007A 3B 008A 2D		FCB FDB FCB FDB FCB	COMMA COMMAA-MTABA SEMIC SEMICA-MTABA MINUS	
0657 FACD 0658 FACE 0659 FAD0	0051 2C 007A 38 008A 2D 008E 01		FCB FDB FDB FDB FCB FDB FCB	Comma Commaa-Mtaba Semic Semica-Mtaba Minusa Minusa-Mtaba 1	
0657 FACD 0658 FACE 0659 FAD0 0660 FAD1 0661	0051 2C 007A 3B 008A 2D 008E	MTABAE	FCB FDB FCB FCB FDB FCB FCB FCB FDB	Comma Commaa-Mtaba Semic Semica-Mtaba Minus Minusa-Mtaba	
0657 FACD 0658 FACE 0659 FAD0 0660 FAD1 0661 0662	0051 2C 007A 38 008A 2D 008E 01 0071 FAD3	*	FCB FDB FCB FDB FCB FCB FCB FCB FCB FCB FCB FDB EQU	Comma Commaa-MTABA SEMIC SEMICA-MTABA MINUSA-MTABA 1 NOTA-MTABA *	1
9657 FACD 9658 FACE 9659 FAD9 9660 FAD1 9661 9662 9663 FAD3 9664 FAD4	0051 2C 007A 3B 008A 2D 008E 01 0071 FAD3 04 20	*	FCB FDB FCB FDB FCB FCB FCB FCB FCB FCB FCB FCB	Contria Contria-MTABA SEMIC SEMICA-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA * MTABBE-MTABB/3- SPACE	1
9657 FACD 9658 FACE 9659 FAD9 9660 FAD1 9661 9662 9663 FAD3	0051 2C 007A 38 008A 2D 008E 01 00871 FAD3 04	* MTABB	FCB FDB FCB FDB FCB FCB FCB FCB FCB FCB FCB FCB	Conma Comma Semic Semic Semica-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA * MTABBE-MTABB/3-	1
9657 FACD 9658 FACE 9659 FAD 9660 FAD1 9661 9662 9663 FAD3 9664 FAD4 9665 FAD5 9664 FAD7 9667 FAD8	0051 2C 907A 38 006A 2D 008E 01 0671 FAD3 04 20 005E 2C 0068D	* MTABB	FC8 FD8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC	Comma Comma-Mtaba Senic SenicA-Htaba Minusa Minusa-Mtaba Minusa-Mtaba * Mtabbe-Mtaba/3- SpaceB-Mtabb Comma-Ttabb	1
0537 FACD 0659 FAD0 0659 FAD0 0660 FAD1 0661 0662 0663 FAD3 0664 FAD4 0665 FAD5 0666 FAD7 0666 FAD7 0666 FAD8	0051 2C 007A 38 008A 20 008E 01 0871 FAD3 04 20 04 22 0035E 2C 006D 38 00659	* MTABB	FC8 FD8 FC8 FC8 FC8 FD8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC	Comma Comma-Mtaba Senic SenicA-Mtaba MinusS MinusS-Mtaba 1 Nota-Mtaba * Mtabbe-Mtaba Space Space Comma CommaB-Mtabb Senic SenicS SenicB-Mtabb	1
9657 FACD 9658 FADE 9659 FADE 9660 FAD1 9661 9663 FAD2 9663 FAD2 9664 FAD4 9665 FAD5 9666 FAD7 9666 FAD7 9666 FAD7 9667 FADB	0051 2C 007A 38 008A 2D 008E 01 0071 FAD3 04 20 04 20 045E 2C 0085E 2C 0085E 2C 0085E 3B	* MTABB	FC8 FD8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC	COMMA COMMA-MTABA SENIC SENICA-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA 1 NOTA-MTABA * MTABBE-MTABA SPACE SPACEB-MTABA COMMA COMMA-TTABB SENIC	1
0537 FACD 0659 FADB 0659 FADB 0660 FAD1 0661 0663 FAD3 0664 FAD3 0664 FAD3 0665 FAD3 0665 FAD5 0665 FAD5 0665 FADB 0667 FADB 0669 FADB 0669 FADB 0670 FADD 0671 FADE	0051 2C 0007A 3B 008A 2D 008E 01 FAD3 04 20 005E 2C 0085E 2C 0085E 2C 00659 2D 0072 01	* MTABB	FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8 FC8	Comma Comma-mtaba Senic Senica-mtaba Minus Minusa-mtaba 1 Nota-mtaba * Mtabbe-mtaba Space SpaceB-mtaba Comma-mtaba Senic Senica-mtaba Minusa-mtaba 1	ı
0537 FACD 0658 FADE 0658 FADE 0660 FAD1 0661 0662 0663 FAD3 0663 FAD3 0664 FAD4 0665 FAD5 0665 FAD5 0665 FAD5 0665 FADB 0667 FADB 0667 FADB 0671 FADE 0671 FADE 0672 FAE0 0672 FAE1 0674	0051 2C 007A 3B 008A 2D 008E 01 0071 FAD3 04 20 04 22 005E 22 006D 3B 0069 2D 0072	* MTABB	FC8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD	COMMA COMMA-MTABA SEMIC SEMICA-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA * MTABBE-MTABA SPACE SPACEB-MTABB COMMA-MTABB SEMIC SEMICB-MTABB MINUSA-MTABB	ı
0637 FACD 0638 FAD6 0669 FAD1 0661 6661 0662 FAD3 0664 FAD3 0665 FAD3 0664 FAD3 0665 FAD3 0666 FAD7 0667 FAD8 0668 FAD4 0669 FAD8 0669 FAD8 0669 FAD8 0668 FAD4 0669 FAD8 0667 FAD8 0671 FAD8 0672 FAE1 0674 FAE1 0675 FAE1	0051 2C 007A 38 008A 2D 008E 61 0071 FAD3 04 20 04 20 04 20 04 20 005E 2C 066D 38 00659 2D 00072 01 007B	* MTABB	FC8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD	COMMA COMMA-MTABA SEMIC SEMICA-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA * MTABBE-MTABA SPACE SPACEB-MTABB COMMA-MTABB SEMIC SEMICB-MTABB 1 NOTB-MTABB *	1 and change routine
0537 FACD 0658 FADE 0658 FADE 0660 FAD1 0661 0662 0663 FAD3 0663 FAD3 0664 FAD4 0665 FAD5 0665 FAD5 0665 FAD5 0665 FADB 0667 FADB 0667 FADB 0671 FADE 0671 FADE 0672 FAE0 0672 FAE1 0674	0051 2C 007A 38 008A 2D 008E 61 0071 FAD3 04 20 04 20 04 20 04 20 005E 2C 066D 38 00659 2D 00072 01 007B	* MTABB MTABBE *	FC8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD8 FD	COMMA COMMA-MTABA SEMIC SEMICA-MTABA MINUS MINUSA-MTABA 1 NOTA-MTABA * MTABBE-MTABB/3- SPACE SPACE SPACE-MTABB SPACE-MTABB SCMIC SEMIC SEMIC SEMICB-MTABB MINUSB-MTABB 1 NOTB-MTABB * memory examine	

0678 FAE3	3 17	0160	9	VADDR	LBSR	BRKOUT	take out any break
0679 FAE6		03A3	3		LDU	#CBREAK	point to break address store
0680 FAE9		03	ž		BRA	ADDR1	point to break address store
0681 FAEB		0395		GADDR		#GSAU	
0682 FAEE							point to go address store
				ADDR1	LDY	, U	get initial value
0683 FAF1		3E	3		BRA	DATA	
0684 FAF3	DE	AA	5	PADDR	LDU	USRSTK	
0685 FAF5	33	4 A	5		LEAU	10, U	point to user pc
0686 FAF7	20	F5	3		BRA	ADDR1	
0687 FAF9				RADDR		USRSTK	
							get address off 'cc' register
0688 FAFC		03AC	3			#TEMP	dummy location for new value
0689 FAFF	20	30	3		BRA	DATA	
0690				*			
0691 FB01	CE	0393	3	MEM	LDU	#MSAU	point to memory address store
0692 FB04	10AF	C4	6		LDY	, U	get intial value
0693 FB07			2		CLRB	,0	
		FD74				000101	set status zero
0694 FB08				SPACE		CONCHR	get input
0695 FB0B		0B	3		BEQ	CRA	no address given
0696 F00D		FAB7	3		LDX	#MTABA	search address.
0697 FB10	16	FE93	5		LBRA	DISPCH	table
0698			-	*		510.00	
0699 FB13	50		2	CRD	TSTB		· • • •
		200		CRD			if status
0700 FB14 0701 FB16	÷.	02 21	3		BLE	CRA	-1 or 0 then no change
	51	« 1	5		LEAY	1, Y	else up one
0782				*			
0703 FB18		21	6	CRA	TFR	Y, X	print out.
0704 FB1A		FF58	9		LBSR	OPXREG	address
0705 FB1D		84	4		LDA	, x	then.
0706 FB1F		FF75	و		LBSR	OPARSP	data
0707 FB22		01	2				
		A1			LDB	#1	set status +1
0708 FB24			4		STB	ONLINE	allow next line
0709 FB26	20	Ø9	з		BRA	DATA	and continue
0710				*			
0711 FB28	97	A2	4	NOTA	STA	LASTC	save for re-use
0712 FB2A		40	7		BSR	NUMB	get numberwith.
0713 FB2C	29	38	ż		BUS	MERR	
0714 FB2E			6				. error if none
0714 FB2E	11	62			TFR	D, Y	is new address
0715 FB30			- 2	DAT1	CLRB		set status 0
0716		FB31		COMMAA		*	
0717		FB31		SPACEE	EQU	*	
0718 FB31	17	FD4B	9	DATA	LBSR	CONCHR	get input
0719 FB34	27	DD	з		BEQ	CRD	no data, found cr
0720 FB36		FAD3	3		LDX	MITABE	else search.
0721 FB39		FE6h	5		LBRA	DISPCH	
8722		. 2011	•	-	LDKN	DISPCH	data table
0723 FB3C	50		2	SEMICE			
0724 FB30		6 2		SEALCB			test status,
0725 FB3F		02 70	3		BGE	SEMICA	if -1 then.
0723 7835	31	3F	5		LEAY	-1, Y	dec before
0726 FB41		C4		SEMICA		,U	save new address
0727 FB44	39		5		RTS		and exit
8728				*			
0729		FB45		MINUSA	EQU	*	
0730 FB45	31	3F	5	MINUSB	LEAY	-1, Y	back down 1,
0731 FB47	50		2		TSTB		but if status.
8732 F848	20	82	3		BGE	MIND2	is -1 then.
0733 FB4A		3F	5		LEAY	-1, Y	
0734 FB4C		E2		MIND2	BRA	DAT1	back down 2
0735			•	*	Dirit.	DH11	then continue
8736 FB4E	97	A2		NOTB	STA	1 4070	
0737 F850				NOTE		LASTC	save for re-use
		IA	?		BSR	NUMB	in number
0738 FB52		12	3		BVS	MERR	should have number
0739 FB54 (94	4		STB	,Υ	store data then.
0740 FB56	E1 (14	4		CMPB	ÿΨ	check it
0741 FB58 2		36	3		BEQ	COMMAB	
9742 F85A		FF8F	ž		LOX		is ok.
0743 FB5D		Dec				MODV	is ok,
0744 5060			۹) DCD	#MORY STRING	else tell
	31 '		9	-	LBSR	STRING	else tell user
0744 FB60		21	5	COMMAB	LEAY	STRING 1, Y	else tell user go up 1
0745 FB62 (C6 I	21 FF	5 i 2	Commab	LEAY LDB	STRING 1, Y #\$FF	else tell user
0745 FB62 (0746 FB64 ;	C6 I	21	5	Commab	LEAY	STRING 1, Y	else tell user go up 1
0745 FB62 (0746 FB64 ; 0747	C6 20 (21 FF CB	523	ŧ.	leay Lob Bra	STRING 1, Y ##FF DATA	else tell user go up 1 with status —1
0745 FB62 (0746 FB64) 0747 0748 FB66)	C6 20 (17	21 FF CB FD16	5 2 3 9	Commab * MERR	LEAY LOB BRA LBSR	STRING 1, Y #\$FF	else tell user go up 1 with status —1 then continue
0745 FB62 (0746 FB64 ; 0747 0748 FB66 ; 0749 FB69 ;	C6 20 (17	21 FF CB	523	ŧ.	leay Lob Bra	STRING 1, Y ##FF DATA CONCHR	else tell. user go up 1 with status —1 then continue get wrong symbol
0745 FB62 (0746 FB64 ; 0747 0748 FB66 ; 0749 FB59 ; 0750	C6 20 (17	21 FF CB FD16	5 2 3 4 3 9 5	ŧ.	LEAY LOB BRA LBSR	STRING 1, Y ##FF DATA	else tell user go up 1 with status —1 then continue
0745 FB62 (0746 FB64 ; 0747 0748 FB66 ; 0749 FB69 ;	C6 20 (17	21 FF CB FD16	523 95	* MERR *	LEAY LOB BRA LBSR	STRING 1, Y ##FF DATA CONCHR BADCMD	else tell. user go up 1 with status —1 then continue get wrong symbol and tell user
0745 FB62 (0746 FB64 ; 0747 0748 FB66 ; 0749 FB59 ; 0750	C6 20 (17	21 FF CB FD16	523 95 7	* MERR *	LEAY LOB BRA LBSR	STRING 1.9 #SFF DATA CONCHR BADCMD get hex number	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spares
0745 FB62 (0746 FB64 2 0747 0748 FB66 2 0749 FB69 2 0750 0751 0752	C6 20 (17	21 FF CB FD16	52395	* MERR * *	LEAY LOB BRA LBSR	STRING 1, Y ##FF DATA CONCHR BADCMD get hex number and stop on f:	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 (0746 FB64 ; 0747 0748 FB66 ; 0749 FB59 ; 0750 0751 0752 0753	C6 20 (17	21 FF CB FD16	52395	* MERR * *	LEAY LOB BRA LBSR	STRING 1, Y ##FF DATA CONCHR BADCMD get hex number and stop on f:	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spares
0745 FB62 0 0746 FB64 2 0747 0 0748 FB66 2 0749 FB59 2 0759 0 0751 0 0752 0 0752 0 0753 0 0754 0 0754 0 0754 0 0754 0 0754 0 0754 0 0754 0 0 0754 0 0 0754 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	C6 29 (17 16 (21 FF CB FD16 911D	52395	* MERR * * *	LEAY LOB BRA LBSR LBRA	STRING 1, Y ##FF DATA CONCHR BADCMD get hex number and stop on f:	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 (0746 FB64) 0747 FB64) 0748 FB66) 0750 0750 0751 0752 0753 0753 0754 0755	C6 29 (17 16 (21 FF CB FD16	52395	* MERR * * *	LEAY LOB BRA LBSR	STRING 1, Y ##FF DATA CONCHR BADCMD get hex number and stop on f:	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 (0746 FB64) 0746 FB64) 0749 FB65) 0750 0751 0752 0753 0754 0755 0756	C6 29 (17 16 (8	21 FF CB FD16 911D	523 95 , , , , , , , , , , , , , , , , , , ,	* MERR * * *	LEAY LOB BRA LBSR LBRA	STRING 1,Y estFF DATA CONCHR BADCMD get hex number and stop on f: with v=0, if n	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 (0746 FB64 (0747 BFB65 (0748 FB66 (0758 BF55 (0758 BF55 (0752 BF55 (0753 BF55 (0755 BF55 4 0756 BF55 4 0757 FB65 4	C6 20 (17 16 (8	21 FF CB FD16 911D	523 95 , , , , , , , , , , , , , , , , , , ,	* MERR * * *	LEAY LOB BRA LBSR LBRA	STRING 1,Y estFF DATA CONCHR BADCMD get hex number and stop on f: with v=0, if n	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 (0746 FB64 ; 0747 8 FB66 ; 0748 FB66 ; 0759 0751 0752 0753 0754 0755 0755 0755 0755 FB6C 4 0758 FB6C 1	C6 20 (17 16 (8 8 8 8	21 FF CB FD16 911D	523 95	* MERR * * * *	LEAY LDB BRA LBSR LBRA	STRING 1,Y estFF DATA CONCHR BADCMD get hex number and stop on f: with v=0, if n	else tell user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d
0745 FB62 1 0746 FB64 2 0747 0748 FB66 2 0749 FB65 2 0759 0751 0752 0753 0754 0755 0755 0755 0755 0755 0755 0755	C6 20 (17 16 (9 9 9 9 7 9 4 0	21 FF CB FD16 911D	523 95	* MERR * * * *	LEAY LDB BRA LBSR LBRA SETOP CLRA CLRB	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with v=0, if i 0	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and v=1.
0745 FB62 1 0746 FB64 2 0747 0748 FB66 2 0749 FB65 2 0759 0751 0752 0753 0754 0755 0755 0755 0755 0755 0755 0755	C6 20 (17 16 (9 9 9 9 7 9 4 0	21 FF CB FD16 3110	523 95 · · · · · · · · · · · · · · · · · ·	* MERR * * * *	LEAY LDB BRA LBSR LBRA SETDP CLRA CLRB PSHS	STRING 1.Y esFF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value
0745 FB62 0 0746 FB64 2 0748 FB66 2 0749 FB66 2 0750 0 0751 0 0754 0 0755 0 0755 0 0755 0 0755 FB6C 2 0759 FB6C 2 0759 FB6C 2 0759 FB6C 2	C6 20 (17 16 (9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	21 FF 5016 30110 99900 68	523 95 ···································	* MERR * * * *	LEAY LDB BRA LBSR LBRA SETDP CLRA CLRA CLRA BSR	STRING 1,Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if n 0 D D GETHXS	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value
0745 FB62 1 0746 FB64 2 0747 B786 5 0749 FB65 2 0750 0 0751 0 0752 0 0753 0 0755 0 0755 0 0755 0 0755 FB6C 2 0759 FB6C 2 0759 FB6C 2 0760 FB70 0 0760 FB70 0	C6 20 () 17 16 () %F %F %F %4 @ %0 1 29 1	21 FF 5016 5016 50110 6000 6 8 8 4	523 95 22773	* MERR * * * *	LEAY LDB BRA LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value
0745 FB62 0 0746 FB64 2 0747 0 0748 FB66 2 0750 0 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 4 0759 FB6C 2 0759 FB6C 2 0760 FB78 0 0761 FB72 2 0760 FB78 0 0761 FB72 2 0760 FB78 0 0	C6 20 () 17 16 () %F %F %F %A @ %D 1 1 %C 0	21 FF 5016 30110 99900 68	523 95 227732 227732	* MERR * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETOP CLRA CLRB PSHS BSR BVS LDB	STRING 1,Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if n 0 D D GETHXS	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value
0745 FB62 0 0746 FB64 2 0747 FB65 2 0749 FB65 2 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 3 0759 FB6C 3 0759 FB6C 3 0760 FB70 0 0761 FB72 2 0762 FB74 0 0762 FB76 4	C6 220 (17 16 (16 (16 (16 (16 (17 (16 (16 (17 (17 (16 (16 (16 (17 (17 (16 (16 (16 (16 (16 (16 (16 (16	21 FF 5016 5016 50110 6000 6 8 8 4	52395 2277322 2277322	* MERR * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRA CLRB BSR BSR BVS LDB ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value
0745 FB62 1 0746 FB64 2 0747 0748 FB66 2 0749 FB65 2 0759 0750 0750 0751 0 0752 0 0755 0 0755 0 0755 FB62 2 0758 FB62 2 0759 FB62 2 0760 FB76 0 0761 FB76 2 0762 FB74 0 0763 FB76 4 0764 FB74 0 0764 FB74 0	C6 20 () 17 16 () 4 5 5 5 6 6 7 6 7 7 7 7 7 7 7 7 7 7 7 7 7	21 FF 5016 5016 50110 6000 6 8 8 4	52395 22773222 22773222	* MERR * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETOP CLRA CLRB PSHS BSR BVS LDB	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex
0745 FB62 0 0745 FB64 2 0747 0 0748 FB66 2 0749 FB65 2 0750 0 0751 0 0752 0 0754 0 0755 0 0755 0 0755 FB6C 2 0759 FB6E 2 0760 FB70 0 0761 FB72 2 0762 FB76 4 0763 FB76 4 0763 FB76 4 0764 FB77 4 0764 FB77 4	C6 220 (17 16 (16 (16 (16 (16 (16 (16 (16 (21 FF 5016 5016 50110 6000 6 8 8 4	52395 2277322 2277322	* MERR * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS LDB ASLA ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex
0745 FB62 1 0746 FB64 2 0747 B748 FB66 2 0749 FB65 2 0750 0751 0752 0753 0751 0755 0755 0755 0755 0755 0755 0755 0755	C6 229 () 17 16 () 57 () 57 () 57 () 57 () 57 () 57 () 57 () 58 () 5	21 FF 5016 5016 50110 6000 6 8 8 4	52395 22773222 22773222	* MERR * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS LDB ASLA ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value wasn't hex move to
0745 FB62 0 0745 FB64 2 0747 0 0748 FB66 2 0749 FB66 0 0750 0 0751 0 0751 0 0753 0 0754 0 0755 FB60 2 0759 FB66 2 0760 FB70 0 0761 FB72 2 0762 FB76 0 0763 FB76 0 0763 FB76 0 0763 FB76 0 0763 FB76 0 0763 FB76 0 0765 FB79 0 0765 FB79 0 0765 FB79 0	C6 220 (17 16 (16 (16 (17) 16 (16) 16 (16) 16 (16) 16 (16) 16 (16) 16 (16) 16 (17) 17 17 17 17 17 17 17 17	21 FF 5016 5016 50110 6000 6 8 8 4	52395 2277322222 2277322222	* MERR * * * MMB MMB3	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BSS BSS BSS ASLA ASLA ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D DETHXS NUMB1	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to., high nibble
0745 FB62 1 0746 FB64 2 0747 P748 FB66 2 0749 FB65 2 0750 0 751 0 752 0 753 0 754 0 755 FB6C 2 0757 FB6C 3 0759 FB6C 3 0759 FB6C 3 0760 FB70 0 0761 FB72 2 0762 FB74 0 0763 FB76 4 0764 FB77 4 0765 FB79 4 0765 FB79 4 0766 FB79 4 0766 FB79 4 0766 FB79 4	C6 C29 () () () () () () () () () () () () ()	21 FF 58 5016 5110 8090 6 8 4 4	523 95 22773222222 227732222222	* # # # # # # # # # # # # # # # # # # #	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRA BSR LDB ASLA ASLA ASLA ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with u=0, if i 0 D GETHORS NUMB1 #4	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value wasn't hex move to
0745 FB62 1 0746 FB64 2 0747 P748 FB66 2 0749 FB65 2 0750 0 751 0 752 0 753 0 754 0 755 FB6C 2 0757 FB6C 3 0759 FB6C 3 0759 FB6C 3 0760 FB70 0 0761 FB72 2 0762 FB74 0 0763 FB76 4 0764 FB77 4 0765 FB79 4 0765 FB79 4 0766 FB79 4 0766 FB79 4 0766 FB79 4	C6 C29 () () () () () () () () () () () () ()	21 FF 28 5016 5110 6000 6 8 4 4	523 95 227732222227 N N N	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBRA LBRA LBRA SETDP CLRA CLRB PSHS BSR BSR BSR BSR BSR ASLA ASLA ASLA ASLA ASLA	STRING i.v esFF DATA CONCHR BADCHD get hex number and stop on f with v=0, if i 0 D GETHXS NUMB1 04 1,5	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to high nibble rotate into
0745 FB62 0 0746 FB64 2 0747 0 0748 FB66 2 0750 0 0750 0 0751 0 0754 0 0755 0 0755 0 0755 0 0755 FB60 2 0759 FB66 2 0760 FB78 0 0761 FB72 0 0761 FB72 0 0761 FB72 0 0763 FB76 0 0764 FB78 0 0764 FB78 0 0764 FB78 0 0764 FB78 0 0766 FB78 0 0769 FB78 0 0 0769 FB78 0 0 0769 FB78 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	C6 C20 C	21 FF 508 5016 5110 60900 66 68 64 4	523 95 2277322222277 N N	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBRA LBRA LBRA SETDP CLRA CLRB BSR BVS LDB SVS LDB ASLA ASLA ASLA ASLA ASLA ASLA ASLA	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with u=0, if i 0 D GETHORS NUMB1 #4	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to., high nibble
0745 FB62 0 0746 FB64 2 0747 FB65 2 0750 FB65 2 0750 0 0751 0 0752 0 0753 0 0755 0 0755 FB6C 3 0759 FB6C 3 0759 FB6C 3 0760 FB79 0 0763 FB76 0 0763 FB76 4 0764 FB77 4 0765 FB78 4 0765 FB78 4 0766 FB79 4 0766 FB79 4 0766 FB79 4 0766 FB79 4 0766 FB78 4	C6 C29 C	21 FF 28 5016 5110 6000 66 8 4 4	523 95 22773222222772 N N	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSS BSS LDB BSSR BVS ASLA ASLA ASLA ASLA ROL ROL DECB	STRING 1.Y ##FF DATA CONCHR BADCMD get hex numbe and stop on f with v=0, if 0 D GETHXS NUMB1 #4 1,S 0,S	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to high nibble rotate into value
0745 FB62 0 0746 FB64 0 0747 0 0748 FB66 0 0749 FB59 0 0750 0 0751 0 0752 0 0754 0 0755 0 0755 0 0755 0 0756 FB70 0 0761 FB72 0 0762 FB74 0 0764 FB74 0 0764 FB74 0 0764 FB74 0 0764 FB75 0 0766 FB79 4 0766 FB79 4 0766 FB79 4 0767 FB75 1 0770 FB75 1 0770 FB75 1 0771 FB76 0 0771 FB75 0 0770 FB75 0 0770 FB75 0 0770 FB75 0 0770 FB75 0 0 0770 FB75 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	C6 C20 (220 (117) 116 (1	21 FF CB 0016 011D 0000 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	523 95 227732222227723 N N	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETOP CLRA CLRA PSHS BSR BVS LDB SVS LDB ASLA ASLA ASLA ASLA ASLA ASLA ASLA ASL	STRING 1,Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if n 0 D GETHOS NUMB1 #4 1,S 0,3 NUMB2	else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d-0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to., high nibble rotate into. value do 4 times
0745 FB62 0 0746 FB64 2 0747 FB66 2 0749 FB69 2 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 3 0755 FB6C 3 0759 FB6C 3 0760 FB70 0 0761 FB72 2 0762 FB76 4 0764 FB77 4 0766 FB78 4 0766 FB78 4 0766 FB78 4 0766 FB78 6 0769 FB76 5 0771 FB60 2 0771 FB60 2 0772 FB62 0 0771 FB60 2 0772 FB62 0 0772 FB62 0	C6 20 (117 117 116 (117 116 (116 116 (116 116	21 FF 28 5016 5110 5000 68 8 4 4 4	523 95 2277322222277237	* * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BSR ASLA ASLA ASLA ASLA ASLA ASLA ASLA BSR BNE BSR BSR	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with v=0, if i 0 D QETHONS NUMB1 04 1,S 0.S NUMB2 GETHEX	<pre>else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, inst non-hex character. return number in d, no number then d=0 and v=1. put initial zero value get first non-blank as hex value wasn't hex move to high nibble rotate into value do 4 times get a hex digit from console</pre>
0745 FB62 1 0746 FB64 2 0747 0 0748 FB66 2 0750 0 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 2 0755 FB6C 2 0760 FB76 0 0761 FB72 0 0761 FB72 0 0762 FB74 0 0764 FB75 0 0770 FB75 0 0770 FB75 0 0771 FB96 2 0772 FB96 2 0772 FB96 2	C6 C6 C9 17 16 17 16 16 16 16 16 16 17 16 17 16 17 16 16 17 16 17 16 16 17 16	21 FF CB 5016 5110 60000 65 65 64 4 4	523 95 22773222222772373	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS LDB ASLA ASLA ASLA ASLA ASLA ASLA BSR ROL CLCB BINE BSNE	STRING 1,Y ##FF DATA CONCHR BADCHD get hex number and stop on f with v=0, if n 0 D GETHOS NUMB1 #4 1,S 0,3 NUMB2	<pre>else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial Zero value get first non-blank as hex value wasn't hex move to., high nibble rotate into. value do 4 times get a hex digit from console was hex se use</pre>
0745 FB62 0 0746 FB64 2 0747 FB66 2 0749 FB69 2 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 3 0755 FB6C 3 0759 FB6C 3 0760 FB70 0 0761 FB72 2 0762 FB76 4 0764 FB77 4 0766 FB78 4 0766 FB78 4 0766 FB78 4 0766 FB78 6 0769 FB76 5 0771 FB60 2 0771 FB60 2 0772 FB62 0 0771 FB60 2 0772 FB62 0 0772 FB62 0	C6 C6 C9 17 16 17 16 16 16 16 16 16 17 16 17 16 17 16 17 16 17 16 16 17 16 16 17 16	21 FF CB 5016 5110 60000 65 65 64 4 4	523 95 22773222222772373	* * * UHB3 UHB3	LEAY LDB BRA LBSR LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BSR ASLA ASLA ASLA ASLA ASLA ASLA ASLA BSR BNE BSR BSR	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with v=0, if i 0 D GETHCS NUMB1 #4 1,S 0.S NUMB2 GETHEX NUMB2	<pre>else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move ta., high nibble rotate into value do 4 times get a hex digit from console was hex so use else (nish with u clear</pre>
0745 FB62 1 0746 FB64 2 0747 0 0748 FB66 2 0750 0 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 2 0755 FB6C 2 0760 FB76 0 0761 FB72 0 0762 FB76 0 0764 FB77 0 0764 FB79 0 0770 FB75 0 0771 FB96 2 0772 FB97 0 0772 FB96 2	C6 C6 C9 17 16 17 16 16 16 16 16 16 17 16 17 16 17 16 17 16 17 16 16 17 16 16 17 16	21 FF CB 5016 5110 60000 65 65 64 4 4	523 95 22773222222772373	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS LDB ASLA ASLA ASLA ASLA ASLA ASLA BSR ROL CLCB BINE BSNE	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with v=0, if i 0 D QETHONS NUMB1 04 1,S 0.S NUMB2 GETHEX	<pre>else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move ta., high nibble rotate into value do 4 times get a hex digit from console was hex so use else (nish with u clear</pre>
0745 FB62 1 0746 FB64 2 0747 0 0748 FB66 2 0750 0 0750 0 0751 0 0752 0 0753 0 0754 0 0755 FB6C 2 0755 FB6C 2 0755 FB60 2 0760 FB76 0 0761 FB72 0 0764 FB76 0 0764 FB77 0 0764 FB79 0 0764 FB79 0 0764 FB79 0 0764 FB79 0 0764 FB79 0 0764 FB79 0 0770 FB76 0 0771 FB96 2 0772 FB97 0 0772 FB96 2	C6 C6 C9 17 16 17 16 16 16 16 16 16 17 16 17 16 17 16 17 16 17 16 16 17 16 16 17 16	21 FF CB 5016 5110 60000 65 65 64 4 4	523 95 22773222222772373	* * * * * * * * * * * * * * * * * * *	LEAY LDB BRA LBSR LBSR LBRA SETDP CLRA CLRB PSHS BSR BVS LDB ASLA ASLA ASLA ASLA ASLA ASLA BSR ROL CLCB BINE BSNE	STRING 1.Y ##FF DATA CONCHR BADCHD get hex number and stop on f: with v=0, if i 0 D GETHCS NUMB1 #4 1,S 0.S NUMB2 GETHEX NUMB2	<pre>else tell. user go up 1 with status -1 then continue get wrong symbol and tell user r from input stream, allow leading spaces, irst non-hex character. return number in d, no number then d=0 and u=1. put initial zero value get first non-blank as hex value wasn't hex move to., high nibble rotate into. value do 4 times get a hex digit from console was hex so use else finish with u clear</pre>

				-				
	F888	35	86	9	NUMB1	PULS	D, PC	
0776 0777					-		gethys - get a l	hex digit ignoring leading spaces
0778							gethex - get a l	hex digit
0779					*		both return valu	ue in a, with um0, else set u=1 if non-hex
0780								
	FB8A		FCF2		GETHXS		CONCHR	get input
	F880		22	3		BEQ	GETH5	on cr, no number
	FB8F FB91		20	23		CMPA	#SPACE	if space.
	FB93		F7 05	3		BEQ BRA	GETHXS GETH2	., then ignore change to hex
0786	1055	20	00	0	•	E-WAT 1	Call I Fran	change vo nex
	FB95	17	FCE7	9	GETHEX	LBSR	CONCHR	get input
	F898		17	З		BEQ	GETH5	on cr, no number
	FB9A		30		GETH2	CMPA	* '0	
	FB9C		ec	3		BLO	GETH1	illegal hex
	FB9E		39 14	2		CMPA BLS	#19 GETH3	number hex
	FBA8 FBA2		41	20		CMPA	#'A	nomber nes
	FBA4		84	ž		BLO	GETHI	illegal hex
	FBAG		46	2		CMPA	#'F	•
	FBAS		ØA	З		BLS	GETH4	alpha hex
	FBAA		2C		GETH1	CMPA	#COMMA	
	FBAC		03	3		BEQ	GETH5	absorb comma
	FBAE		03A2	5	OFTHE	STA SEU	LASTC	else re-use bad hex
	FBB1 FBB3		82	5	GETH5	RTS		Ded nex
0802	1000	55			*	K. 0		
	FBB4	80	07	2	GETH4	SUBA	#7	alpha offset
0804	FBB6	80	38	2		SUBA	₩'0	number offset
0605	FBB8	39		5		RTS		with v clear
9896					*			
0807			9983			SETDP	MONDP	
0808 0809					*			gram using stack as stands
0810					-		Lesone azer biot	gram using stack as stands
	F889	32	62	5	RESUME	LEAS	2, 8	strip return address
	FBBB		AF	7		BSR	NUMB	get number or zero
0 813	FBBD	00	A5	5		STD	NBREAK	and set break ignore count
0814	FOOF	80	60	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 interru	upt handler, come here on breakpoint
0818					*		either stops and	displays registers or traces past
0819					*		breakpoint and r	
6826					*		•	
0821	FBC3	86	0 3	2	SWIHMN		#MONDP	setup
0.077	FBC5	1 5	88	6		TER		
	FBCJ	11		•			A, DP	direct page
882 3			8883	-		SETDP	MONDP	tell assembler
0823 0824	FBC7	Æ	8993 6A	6		SETDP	MONDP 10, S	tell assembler back up
0823 0824 0825	FBC7 FBC9	AE 30	8883 6A 1F	6		SETDP LDX LEAX	MONDP 10,5 -1,X	tell assembler back up user
0823 0824 0825 0826	FBC7 FBC9 FBCB	AE 30 AF	8083 6A 1F 6A	656		SETDP LDX LEAX STX	MONDP 10, S -1, X 10, S	tell assembler back up user program counter
9823 9824 9825 9826 9827	FBC7 FBC9 FBC8 FBC0	AE 30 AF 80	8883 6A 1F	6		SETDP LDX LEAX	MONDP 10,5 -1,X	tell assembler back up user
9823 9624 9825 9826 9827 9828	FBC7 FBC9 FBC8 FBC0	AE 30 AF	8883 6A 1F 6A 77	656753		SETDP LDX LEAX STX BSR LDX BEQ	HONDP 10, S ~1, X 10, S BRKOUT	tall assembler back up., ., User,. ., program counter remove breakpoint get count stop if zero, else.
8823 8824 8825 8826 8827 8828 8829 8829 8838	FBC7 FBC9 FBCB FBCD FBC7 FBC1 FBC1 FBC3	AE 30 AF 90 92 27 30	8003 6A 1F 6A 77 A5 08 1F	6567535	-	SETDP LDX LEAX STX BSR LDX BEQ LEAX	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X	tell assembler back up ., user ., program counter remove breakpoint get count stop if zero, else. ., decrement,
8823 9824 9825 9826 9827 9828 9829 9839 9839 9830	FBC7 FBC9 FBC8 FBC0 FBC7 FB01 FB03 FB03 FB05	AE 30 AF 90 927 30 95	8003 6A 1F 6A 77 A5 88 1F A5	65675355		SETDP LDX LEAX STX BSR LDX BEQ LEAX STX	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement and restore
8823 9824 9825 9826 9827 9828 9829 9839 9839 9830 9831 9832	FBC7 FBC9 FBC8 FBC0 FBC7 FB01 FB03 FB05 FB07	AE 30 AF 90 92 30 95 95 95	8003 6A 1F 6A 77 A5 08 1F A5 76	656753557	-	SETDP LDX LEAX STX BSR LDX BEQ LEAX STX BSR	MONDP 10,5 5 10,5 BRKOUT BRKOUT NBREAK RES1 -1,X NBREAK TUSER1	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break
8823 9824 9825 9826 9827 9828 9829 9839 9830 9831 9832 9833	FBC7 FBC9 FBCB FBCD FBCF FB01 FB03 FB05 FB07 FB09	AE 30 AF 90 92 39 50 92 90 20	8003 6A 1F 6A 77 A5 08 1F A5 76 E4	6567535573	RES1	SETDP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BSR BRA	MONOP 10,5 -1,X 10,5 BRKOUT NBREAK RES1 -1,X NBREAK TUSER1 RES2	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement and restore
9823 9824 9825 9826 9827 9828 9829 9830 9830 9831 9832 9833 9834	FBC7 FBC9 FBC8 FBC0 FBCF FB01 FB03 FB05 FB07 FB09 FB09 FB08	AE 30 AF 80 9E 27 39 80 20 100F	8003 6A 1F 6A 77 77 A5 88 1F 45 76 E4 AA	65675355736	RES1	SETDP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BSR BSR STS	MONDP 10,5 5 10,5 BRKOUT BRKOUT NBREAK RES1 -1,X NBREAK TUSER1	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement, ., and restore trace past break. ., then resume again
9823 9824 9825 9826 9827 9828 9829 9838 9838 9838 9831 9832 9833 9834 9835	FBC7 FBC9 FBC8 FBC0 FBC7 FB01 FB03 FB03 FB03 FB05 FB07 FB09 FB08 FB08 FB06	AE 30 AF 80 9E 27 38 9F 80 20 100F 17	8003 6A 1F 6A 77 A5 08 1F A5 76 E4	6567535573	RES1	SETDP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BSR BRA	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break
9823 9824 9825 9826 9826 9828 9829 9830 9831 9832 9833 9834 9835 9834 9835 9835 9835 9836 9836 9836	FBC7 FBC9 FBC8 FBC0 FBCF FB01 FB03 FB05 FB07 FB09 FB09 FB08	AE 30 AF 80 9E 27 38 9F 80 20 100F 17	8093 6A 1F 6A 77 A5 88 1F A5 76 E4 AA FCD7	656753557369	*	SETOP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BRA STS LBSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RESI -1, X -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor
9823 9824 9825 9826 9827 9828 9829 9838 9831 9832 9833 9834 9833 9834 9835 9836 9835 9838	FBC7 FBC9 FBC8 FBC0 FBC7 FB01 FB03 FB03 FB03 FB05 FB07 FB09 FB08 FB08 FB06	AE 30 AF 80 9E 27 38 9F 80 20 100F 17	8093 6A 1F 6A 77 A5 88 1F A5 76 E4 AA FCD7	656753557369	*	SETOP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BRA STS LBSR	MONDP 10, S -1, X 10, S BRKDUT BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers
9823 9824 9825 9826 9827 9828 9829 9839 9830 9831 9832 9833 9834 9835 9836 9836 9838 9838 9838 9838	FBC7 FBC9 FBC8 FBC0 FBC7 FB01 FB03 FB03 FB03 FB05 FB07 FB09 FB08 FB08 FB06	AE 30 AF 80 9E 27 38 9F 80 20 100F 17	8093 6A 1F 6A 77 A5 88 1F A5 76 E4 AA FCD7	656753557369	*	SETOP LDX LEAX STX BSR LDX BEQ LEAX STX BSR BRA STS LBSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RESI -1, X -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor
9823 9824 9825 9826 9826 9829 9829 9839 9831 9831 9833 9834 9833 9834 9835 9836 9837 9838 9838 9838 9838 9838 9838	FBC7 FBC9 FBC8 FBC7 FB03 FB03 FB03 FB03 FB03 FB03 FB05 FB09 FB08 FB08 FB08 FB08 FB08	AE 30 AF 80 9E 27 30 9F 80 20 100F 17 20	9903 64 1F 64 77 45 88 1F 45 76 88 1F 26 44 40 FCD7 19	6567535573693	* * *	SETDP LDX LEAX STX BSR LDX BSR BEQ LEAX STX BSR BRA STS LBSR BRA	MONDP 10,5 -1,X 10,5 BRKDUT BRKDUT NBREAK RES1 -1,X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor
9823 9824 9825 9826 9826 9829 9839 9839 9833 9834 9835 9834 9835 9834 9835 9834 9835 9838 9839 9844	FBC7 FBC9 FBC8 FBC07 FB03 FB03 FB03 FB05 FB05 FB09 FB08 FB08 FB08 FB08 FB08 FB08 FB08 FB08	AE 30 AF 80 9E 27 38 9F 80 100F 17 20 80	8083 64 1F 65 77 75 88 87 85 76 85 76 87 87	6567535573693 7	*	SETDP LDAX LEAX STX BSR LDX BEG LEAX STX BSR BRA STS LBSR BRA BRA BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 Change number if destroys d NUMB	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor
9823 9824 9825 9826 9826 9829 9839 9839 9833 9833 9833 9835 9835 983	FBC7 FBC9 FBC8 FBC7 FB03 FB03 FB03 FB03 FB03 FB03 FB05 FB09 FB08 FB08 FB08 FB08 FB08	AE 20 AF 80 9E 27 39F 80 200 100F 17 20 80 29	9903 64 1F 64 77 45 88 1F 45 76 88 1F 26 44 40 FCD7 19	6567535573693	* * *	SETDP LDX LEAX STX BSR LDX BSR BEQ LEAX STX BSR BRA STS LBSR BRA	MONDP 10, S -1, X 10, S BRKDUT BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d	tell assembler back up., user. program counter remove breakpoint get count stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream,
9823 9824 9825 9826 9826 9829 9831 9832 9831 9832 9834 9835 9834 9835 9836 9836 9836 9836 9836 9836 9836 9836	FBC7 FBC9 FBCB FBC0 FBC7 FB01 FB03 FB07 FB09 FB08 FB07 FB09 FB08 FB07 FB09 FB08 FB07 FB09 FB08 FB07 FB03 FB07 FB03 FB07 FB03 FB03 FB03 FB03 FB03 FB03 FB03 FB03	AE 30 AF 80 9E 27 39F 80 200 100F 17 20 80 91 17 20	89903 6A 1F 6A 77 A5 88 88 1F A5 66 E4 FCD7 19 87 82 82 82 82	6567535573693 73	* * *	SETDP LDX LEAX STX BSR BSR BRA BSR BRA BSR BSR BSR BRA BSR FFR	MONDP 10,S -1,X 10,S BRKOUT NBREAK RES1 -1,X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMB	tell assembler back up., user. program counter remove breakpoint get count stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream,
6823 9824 98255 9826 9826 9829 9831 9832 9831 9832 9833 9834 9835 9839 9839 9838 9839 9834 9835 9839 9834 9835 9834 9835 9834 9834 9834 9845	FBC7 FBC9 FBC9 FBC0 FBC7 FB03 FB03 FB03 FB07 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB07 FB09 FB09 FB09 FB09 FB09 FB09 FB09 FB09	AE 30 AF 80 9E 27 39F 80 200 100F 17 20 80 91 17 20	89903 6A 1F 6A 77 A5 88 88 1F A5 66 E4 FCD7 19 87 82 82 82 82	6567535573693 736	* * * NUMBX NUMBX1	SETDP LDX LEAX STX BSR BSR BRA BSR BRA BSR BSR BSR BRA BSR FFR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. dacrement and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream.
8823 9824 98255 9826 9826 9828 9839 9839 9839 9831 9833 9834 9833 9834 9833 9834 9833 9834 9833 9834 9833 9834 9834	FBC7 FBC9 FBC9 FBC0 FBC7 FB03 FB03 FB03 FB07 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB09 FB08 FB07 FB09 FB09 FB09 FB09 FB09 FB09 FB09 FB09	AE 30 AF 80 9E 27 39F 80 200 100F 17 20 80 91 17 20	89903 6A 1F 6A 77 A5 88 88 1F A5 66 E4 FCD7 19 87 82 82 82 82	6567535573693 736	* * * NUMBX NUMBX1 *	SETDP LDX LEAX STX BSR BSR BRA BSR BRA BSR BSR BSR BRA BSR FFR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X	tell assembler back up., user. program counter remove breakpoint get count stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream,
823 9823 9825 9826 9826 9828 9829 9839 9839 9831 9832 9833 9834 9833 9834 9833 9834 9833 9834 9834	FBC7 FBC7 FBC8 FBC0 FBC7 FB03 FB05 FB03 FB05 FB09 FB08 FB09 FB08 FB09 FB08 FB04 FB04 FB04 FB05 FB09 FB08 FB09 FB08 FB09 FB09 FB09 FB09 FB09 FB09 FB09 FB09	AE 3 AF 80 927 338 950 238 950 240 17 20 80 957 80 80 957 80 957 80 80 957 80 80 957 80 80 957 80 80 957 80 80 80 80 80 80 80 80 80 80	8993 64 64 77 65 66 77 65 17 76 68 15 76 19 87 87 89 89 89 89	6567535573693 7365	* * * NUMBX NUMBX1 * *	SETDP LDX LEAX STX BSR LDX BEQ LEAX STX BBRA STS LBSR BRA BRA BSR BRA	MONDP 10, S -1, X 10, S BRKDUT BRKDUT NBREAK TUSER1 RES1 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. dacrement and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream.
8823 9824 9825 9826 9826 9828 9829 9839 9832 9832 9832 9833 9834 9834 9832 9832 9833 9834 9834 9834 9834 9834 9844 8844 8	FBC7 FBC8 FBC0 FBC1 FB03 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB07	AE 30 AF 80 95 27 38 95 20 100 F 20 80 29 17 20 80 29 17 39 39 39 39 39 39 39 39 39 39	8993 6A 1F 6A 77 A5 08 1F 77 A5 77 A5 77 48 76 87 87 82 81 75	6567535573693 7365 5	* * * NUMBX NUMBX1 *	SETDP LDX LEAX STX BSR LDX BEQ SSR BSR STS BRA STS BRA BRA BRA BRA BRA LESSR BRA LLASSR LLASSR BRA LLASSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 Change number if destroys d NUMB NUMEX1 D, X user program ref -2, S	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream, no number turns here if rts done make room for new return address
8823 9824 9825 9826 9827 9828 9829 9839 9833 9833 9833 9833 9833	FBC7 FBC9 FBC9 FBC0 FBC1 FB03 FB03 FB03 FB07 FB09 FB08 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB07	AE 30 30 59 50 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 64 77 45 66 17 45 77 45 98 15 19 87 87 82 81 87 82 81	6567535573693 7365 57	* * * NUMBX NUMBX1 * *	SETDP LDX LEAX STX BSR LDX BESR LDX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 100, S -1, X 10, S BOKDUT BOKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB C, A CC, A, B, DP, X, Y, U,	tell assembler back up., user program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break then resume again display registers and stay in monitor n x if one given in input stream.
8823 9824 9825 9826 9826 9827 9828 9829 9839 9832 9833 9832 9833 9834 9833 9834 9833 9834 9834 9834	FBC7 FBC9 FBC8 FBC7 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB0	AE 80 95 27 39 95 20 95 20 107 20 80 95 17 20 80 95 32 48	8983 6A 1F 6A 77 A5 08 1F 76 4A A5 76 4A A6 707 19 87 82 81 7E FF FF FF FF FF FF FF FF FF FF FF FF FF	6567535573693 7365 5	* * * NUMBX NUMBX1 * *	SETDP LDX LEAX STX BSR LDX BEQ SSR BSR STS BRA STS BRA BRA BRA BRA BRA LESSR BRA LLASSR LLASSR BRA LLASSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 Change number if destroys d NUMB NUMEX1 D, X user program ref -2, S	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream, no number turns here if rts done make room for new return address
8823 9824 9825 9826 9827 9828 9829 9839 9839 9832 9832 9832 9833 9834 9835 9834 9835 9834 9835 9834 9834 9834 9834 9846 9842 9844 9842 9844 9845 9845 9845 9845 9845 9845 9845	FBC7 FBC9 FBC8 FBC7 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB0	AE 320 AF 90 27 33 95 20 2100F 117 20 80 29 57 32 46 F	8997 64 64 77 45 68 17 45 76 87 76 19 87 87 82 81 87 87 87 87 87 87 87 87	6567535573693 7365 573	* * * NUMBX NUMBX1 * *	SETDP LDX LEAX STX BSR LDX BSR LDX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBNUMBX1 D, X user program ref -2, S CC, A, B, DP, X, Y, U, #BACK	tell assembler back up user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor and stay in monitor an
8823 9824 9825 9826 9827 9828 9827 9828 9832 9833 9834 9832 9833 9834 9835 9834 9835 9834 9835 9834 9835 9834 9834 9834 9834 9844 9845 9844 9845 9844 9845 9844 9845 9844 9845 9845	FBC7 FBC9 FBC8 FBC7 FBC1 FBC7 FBC7 FBC7 FBC9 FBC7 FBC9 FBC7 FBC9 FBC7 FBC9 FBC7 FBC9 FBC7 FBC9 FBC7 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9	AE 320 AF 90 927 39F 80 1100 F 324 80 927 39F 80 80 929 179 324 80 AF 60 F	8993 6A 1F 6A 77 A5 68 1F 77 A5 88 1F 77 A5 88 1F 77 76 4A 7007 19 87 82 81 87 82 81 87 82 81 87 82 81 87 88 81 87 88 81 87 88 81 87 83 84 84 85 86 86 86 86 86 86 86 86 86 86 86 86 86	6567535573693 7365 573626	* * * NUMBX NUMBX1 * *	SETOP LDX LEAX STX BSR LDX LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMEX1 D, X user program ref -2, S CC, A, B, DP, X, Y, U, #BACK 12, S #MONDP B, DP	tell assembler back up user. user. user. stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream, no number turns here if rts done make room for new return address PC and save all user registers set return address. again
8823 9824 9825 9825 9825 9825 9827 9828 9827 9828 9827 9828 9827 9828 9823 9823 9823 9823 9823 9823 9823	FBC7 FBC9 FBCB FBC0 FBC7 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB0	AE 320 AF 89E 273 9F 920 1000F 120 80 29 F 39 324 8E F 60 F 80	8003 64 64 77 45 66 17 66 17 77 45 77 45 77 45 77 45 77 45 77 45 77 45 87 82 81 72 75 75 82 81 75 75 82 81 75 75 82 81 75 77 77 77 77 77 77 77 77 77 77 77 77	6567535573693 7365 5736267	* * * NUMBX NUMBX1 * *	SETDP LDX LEAX STX BSR LDX LEAX STX BSR BSR BSR BSR BSR BBSR BBSR BBSR BBS	MONDP 10, S -1, X 10, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. dacrement and restore trace past break. den resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints
8823 9824 9825 9826 9827 9828 9827 9828 9827 9828 9832 9833 9834 9832 9833 9834 9833 9834 9835 9834 9835 9834 9835 9834 9834 9846 9842 9844 9844 9844 9844 9844 9844 9844	FBC7 FBC8 FBC8 FBC7 FBC7 FBC7 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9 FBC9	AE 339 AF 80 952 233 9F 223 950 200 1000 F 120 80 29 1F 39 324 8E AF 6 1F 0 100 F	8993 6A 1F 6A 77 A5 68 1F 77 A5 68 1F 77 77 45 77 77 77 77 89 87 87 82 81 75 87 82 81 98 64 98 84 64 64 77 77 77 77 77 77 77 77 77 77 77 77 77	6567535573693 7365 57362676	* * NUMBX NUMBX1 * * BACK	SETOP LDX LEAX STX BSR LDX LEAX STX BSR BRA SSR BRA BSR BRA BSR BRA BSR BRA SSR BSR BSR BSR BSR BSR BSR BSR BSR STS LDS LEAX LDX LDX LEAX LDX LEAX LDX LEAX STX BSR BSR BSR BSR BSR BSR BSR SSR BSR BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMEX1 D, X user program ref -2, S CC, A, B, DP, X, Y, U, #BACK 12, S #MONDP B, DP	tell assembler back up user. user. user. stop if zero, else. decrement, and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream, no number turns here if rts done make room for new return address PC and save all user registers set return address. again
8823 9824 9825 9826 9825 9826 9827 9828 9827 9828 9827 9828 9827 9828 9832 9834 9832 9834 9832 9834 9832 9834 9834 9834 9834 9834 9834 9834 9834	FBC7 FBC9 FBCB FBC0 FBC0 FBC0 FBC0 FB00 FB00 FB00	AE 320 AF 595 27 395 20 395 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 20 295 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 64 64 77 65 77 65 77 65 77 65 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 69 77 77 68 17 69 77 77 68 17 77 77 68 17 77 77 68 17 77 77 78 50 77 77 77 77 77 78 50 77 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 78 50 77 78 78 78 78 78 78 78 78 78 78 78 78	6567535573693 7365 573626763	* * * NUMBX NUMBX1 * *	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 Change number if destroys d NUMB NUMB NUMB NUMB CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S BRKDUT USRSTK	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. den resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer
8823 9824 9825 9826 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9832 9832 9832 9832 9832 9832 9832	FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	AE 320 AF 595 27 395 20 395 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 20 295 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 6A 1F 6A 77 A5 68 1F 77 A5 68 1F 77 77 45 77 77 77 77 89 87 87 82 81 75 87 82 81 98 64 98 84 64 64 77 77 77 77 77 77 77 77 77 77 77 77 77	6567535573693 7365 57362676	* * NUMBX NUMBX1 * * BACK	SETOP LDX LEAX STX BSR LDX LEAX STX BSR BRA SSR BRA BSR BRA BSR BRA BSR BRA SSR BSR BSR BSR BSR BSR BSR BSR BSR STS LDS LEAX LDX LDX LEAX LDX LEAX LDX LEAX STX BSR BSR BSR BSR BSR BSR BSR SSR BSR BSR	MONDP 10, S -1, X 10, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK	tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. dacrement and restore trace past break. den resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints
8823 9824 9825 9826 9826 9827 9828 9827 9828 9827 9828 9827 9828 9828	FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	AE 320 AF 595 27 395 20 395 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 20 295 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 64 64 77 65 77 65 77 65 77 65 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 69 77 77 68 17 69 77 77 68 17 77 77 68 17 77 77 68 17 77 77 78 50 77 77 77 77 77 78 50 77 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 78 50 77 78 78 78 78 78 78 78 78 78 78 78 78	6567535573693 7365 573626763	* * NUMBX NUMBX1 * * BACK	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB NUMB NUMB NUMB	tell assembler back up user. user. user. get counts stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor in x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem. optional address specified
8823 9824 9825 9826 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9832 9832 9832 9832 9832 9832 9832	FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	AE 320 AF 595 27 395 20 395 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 20 295 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 64 64 77 65 77 65 77 65 77 65 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 69 77 77 68 17 69 77 77 68 17 77 77 68 17 77 77 68 17 77 77 78 50 77 77 77 77 77 78 50 77 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 78 50 77 78 78 78 78 78 78 78 78 78 78 78 78	6567535573693 7365 573626763	* * NUMBX NUMBX1 * BACK1 *	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BKCDUT BKCDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S MONDP B, DP BRKOUT USRSTK PARSE go to user program	<pre>tell assembler back up. . user. . program counter remove breakpoint get count stop if zero, else. . decrement . and restore trace past break. . then resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. . , again remove breakpoints save user stack pointer and resume monitor functions ram, optional address specified en is reset, but the register contents</pre>
8823 9824 9825 9826 9827 9828 9827 9828 9827 9828 9827 9828 9823 9832 9833 9833 9833 9833 9833	FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	AE 320 AF 595 27 395 20 395 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 295 20 20 295 20 20 20 20 20 20 20 20 20 20 20 20 20	8993 64 64 77 65 77 65 77 65 77 65 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 77 68 17 69 77 77 68 17 69 77 77 68 17 77 77 68 17 77 77 68 17 77 77 78 50 77 77 77 77 77 78 50 77 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 77 78 50 78 50 77 78 78 78 78 78 78 78 78 78 78 78 78	6567535573693 7365 573626763	* * NUMBX BACK1 *	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BKCDUT BKCDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S MONDP B, DP BRKOUT USRSTK PARSE go to user program	tell assembler back up user. user. user. get counts stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor in x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem. optional address specified
823 9823 9824 9825 9825 9825 9825 9827 9828 9827 9828 9832 9834 9832 9834 9833 9834 9835 9834 9835 9834 9835 9834 9835 9834 9834 9834 9834 9834 9834 9844 9845 9844 9845 9844 9845 9844 9845 9855 985	FBC7 FBC9 FBC0 FBC0 FBC7 FB07 FB07 FB07 FB07 FB07 FB07 FB07 FB0	AE 320 AF 60 99E 27 328 99F 328 220 200 177 220 229 220 229 220 229 220 229 220 229 220 220	8993 6A 1F 6A 77 A5 68 1F 77 A5 76 4A 76 76 76 76 76 87 82 87 82 81 75 75 87 82 81 75 75 87 82 81 75 75 87 82 81 87 82 81 87 82 81 87 85 85 85 86 85 86 86 86 86 86 86 86 86 86 86 86 86 86	6567535573693 7365 5736867635	* * NUMBX * BACK BACK * *	SETOP LDX LEAX STX BSR LDX LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB NUMB NUMB C, A C, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S #MONDP B, DP BRKOUT USRSTK PARSE go to user program the stack point are maintained of	<pre>tell assembler back up . user. . program counter remove breakpoint get count stop if zero, else. decrement,. and restore trace past break. then resume again display registers and stay in monitor and stay in monitor turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified ar is reset, but the register contents as listed by the e command.</pre>
823 9823 9824 9825 9825 9825 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9823 9823 9823 9823 9823 9823 9823	FBC7 FBC8 FBC9 FBC0 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	AF 329 AF 80 99E 27 339 580 2100F 12 20 229 229 229 229 229 229 229 229 229	8003 64 64 77 45 66 17 77 45 77 45 77 45 77 45 77 45 77 45 77 45 87 87 87 80 81 72 75 75 80 81 75 75 82 80 81 75 75 75 40 82 80 81 75 77 77 77 77 77 77 77 77 77 77 77 77	6567585878698 7365 5786867685 6	* * NUMBX NUMBX1 * BACK1 *	SETDP LDX LEAX STX BSR LDX LEAX SSR BSR BSR BSR BSR BBSR BBSR BBSR BBS	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK USRSTK PARSE go to user program the stack points are maintained of NBREAK	<pre>tell assembler back up. user. user. user. user. user. get counts stop if zero, else. decrement and restore trace past break. dear restore trace past break. dear resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified ar is reset, but the register contents as listed by the e command. put zero in.</pre>
8823 9824 9825 9826 9827 9825 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9823 9823 9823 9823 9823 9823 9823 9823	FBC7 FBC8 FBC0 FBC0 FBC0 FBC0 FBC0 FB00 FB00 FB00	AF 329 AF 80 99 27 339 99 200 27 339 200 27 100 F 200 200 200 200 200 200 200 200 200	8993 6A 1F 6A 77 A5 68 1F 77 A5 68 1F 77 4A 76 4A 77 49 87 87 82 81 7 7 7 7 87 82 81 7 87 82 81 7 87 82 81 87 82 81 87 84 84 84 84 84 85 84 85 85 85 85 85 85 85 85 85 85 85 85 85	6567535573693 7365 5736867635 66	* * NUMBX * BACK BACK * *	SETDP LDX LEAX STX BSR LDAG LEAX SSR BSR SSR BSR SSR BSR SSR BSR SSR BSR SSR BSR SSR S	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMB NUMB NUMB NUMB NUMB NUMB NUMB	<pre>tell assembler back up . user. . program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified er is reset, but the register contents as listed by the e command. put zero in. break count</pre>
823 9823 9824 9825 9825 9825 9827 9828 9825 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9829 9823 9823 9823 9823 9823 9823	FBC7 FBC9 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0	AF 329 AF 800 SPE 27 339 580 190F 17 20 20 21F 39 324 860 F 100 F 100 F 100 F 10 0 F	8993 64 64 77 45 66 17 77 45 77 45 77 45 77 45 77 45 98 15 77 45 98 97 87 89 91 75 75 80 91 75 75 80 91 75 75 80 91 87 83 940 40 85 85 85 85 85 85 85 85 85 85 85 85 85		* * NUMBX * BACK BACK * *	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 BACK1 Change number in destroys d NUMB NUMB NUMB NUMB NUMB CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S HOMOP B, DP B, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 14, S CC, A, B, DP, X, Y, U, #BACK 14, S CC, A, B, CP, X, Y, U, #BACK 14, S CC, A, B, CP, X, Y, U, #BACK 14, S CC, A,	<pre>tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. detrestore trace past break detrestore detrestore turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified er is reset, but the register contents as listed by the e command. put zero in. break count get pointer</pre>
8823 9824 9825 9825 9825 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9823 9823 9823 9823 9823 9823 9823 9823	FBC7 FBC9 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FB00 FB00	AF 329 AF 329 S9E 239 S9E 239 S92 239	8983 6A 1F 6A 77 A5 68 1F 77 A5 87 87 87 87 87 87 87 87 87 87 87 87 87		* * NUMBX * BACK BACK * *	SETDP LDX LEAX STX BSR LDAG LEAX SBSR SBSR SBSR SBSR SBSR SBSR SBSR SBS	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number if destroys d NUMB NUMBNUMBEN D, X user program ref -2, S CC, A, B, DP, X, Y, U, WBACK 12, S CC, A, B, DP, X, Y, U, WBACK PARSE go to user program the stack point are maintained of NBREAK NBREAK NBREAK STACK	<pre>tell assembler back up. user. user. user. get counts stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor nx if one given in input stream, no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions ram, optional address specified er is reset, but the register contents as listed by the e command. put zero in break count get pointer </pre>
8823 9823 98245 9825 9826 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9828	FBC7 FBC9 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0	AE 329 AF 800 SPE 2733 SPE 00 SPE 1173	8993 64 64 77 76 66 17 77 76 98 17 77 76 98 17 77 76 98 17 77 77 76 98 17 77 77 76 98 17 77 77 76 98 17 77 77 77 76 98 17 77 77 77 77 77 77 77 77 77 77 77 77		* * NUMBX * BACK BACK * *	SETDP LDA LEAX STX BSR LDA LEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 100, S -1, X 100, S BRKDUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 BACK1 Change number in destroys d NUMB NUMB NUMB NUMB NUMB CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S HOMOP B, DP B, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 14, S CC, A, B, DP, X, Y, U, #BACK 14, S CC, A, B, CP, X, Y, U, #BACK 14, S CC, A, B, CP, X, Y, U, #BACK 14, S CC, A,	<pre>tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement and restore trace past break. detrestore trace past break detrestore detrestore turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified er is reset, but the register contents as listed by the e command. put zero in. break count get pointer</pre>
8823 9824 9825 9825 9825 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9828 9827 9823 9823 9823 9823 9823 9823 9823 9823	FBC7 FBC9 FBC0 FBC0 FBC0 FBC0 FBC0 FBC0 FB00 FB00	AF 329 AF 80 99E 739 800 2180 F 120 80 92 F 39 32 46 F 66 1 F 80 F 66	8983 6A 1F 6A 77 A5 68 1F 77 A5 87 87 87 87 87 87 87 87 87 87 87 87 87		* * NUMBX * BACK BACK * *	SETDP LIDAX LEAX STX BSR LDX LEAX SSR SSR SSR SSR SSR SSR SSR SSR SSR SS	MONDP 10, S -1, X 10, S BRKDUT BRKDUT BRKDUT NBREAK TUSER1 RES1 -1, X NBREAK TUSER1 RES2 USRSTK EXREG BACK1 change number in destroys d NUMB NUMBX1 D, X USER program ref -2, S CC, A, B, DP, X, Y, U, #BACK 12, S #MONDP BRKOUT USRSTK PARSE go to user program the stack point are maintained of NBREAK NBREAK BREAK BREAK BREAK BREAK BREAK BREAK BREAK BREAK BREAK	<pre>tell assembler back up. user. program counter remove breakpoint get count stop if zero, else. decrement decrement decrement decrement detrestore trace past break detrestore trace past break detrestore trace past break detrestore trace past break detrestore and stay in monitor n x if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address again remove breakpoints save user stack pointer and resume monitor functions rem, optional address specified er is reset, but the register contents as listed by the e command. put zero in pushed first.</pre>
8823 98234 98255 98248 98265 98279 98299 9829 9829 98231 98232 98244 98252 98255 98252 98255 98555 98555 98555 98555 98555 98555 98555 98555 98555 98555 98555 985	FBC7 FBC7 FBC9 FBC0 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7 FBC7	A 29 A 60 99 27 39 50 00 12 12 80 29 17 39 32 4 6 6 F 60 18 10 16 6 6 6 18 10 16 16 16 16 16 16 16 16 16 16 16 16 16	8003 64 64 77 76 66 77 76 87 77 75 87 80 81 75 76 40 87 80 81 75 75 80 81 75 75 80 81 75 75 80 81 75 75 80 81 75 75 80 81 75 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 76 80 77 77 75 80 76 77 76 80 77 77 76 80 77 76 76 76 76 76 76 76 76 76 76 76 76		* * NUMBX1 NUMBX1 * * BACK BACK1 * * * GOUSER	SETDP LLDX LLEAX STX BSR LD20 LLEAX STX BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	MONDP 10, S -1, X 10, S BRKOUT NBREAK RES1 -1, X NBREAK TUSER1 RES2 USRSTK USRSTK EXREG BACK1 change number if destroys d NUMB NUMBX1 D, X CC, A, B, DP, X, Y, U, #BACK 12, S CC, A, B, DP, X, Y, U, #BACK 12, S B, DP BRKOUT USRSTK PARSE go to user program the stack point: are mainted of NBREAK NBREAK NBREAK NBREAK STACK #BACK	<pre>tell assembler back up. user. user. user. user. get counts stop if zero, else. decrement and restore trace past break. then resume again display registers and stay in monitor hx if one given in input stream. hx if one given in input stream. hx if one given in input stream. no number turns here if rts done make room for new return address PC and save all user registers set return address. again remove breakpoints save user stack pointer and resume monitor functions ram, optional address specified er is reset, but the register contents as listed by the e command. put zero in. break count get pointer return address. pushed first 12 registers plus return</pre>

0871 FC12 5A 0872 FC13 2A 23 DECB for all registers new stack, ignore return address GO 1 F9 BPL 0873 FC15 32 42 5 LEAS 2, U 5 LDX 0874 FC17 9E 95 GSAD saved address 0875 FC19 8D C8 7 BSR SHIMPY change if given. ... and restore 5 0876 FC1B 9F 95 STX 0SAU 6 10,S BRKIN put as user pc insert breakpoint 0877 FC1D AF 66 STX er. 7 0878 FC1F 6D RSP 17 CC, A, B, DP, X, Y, U, PC and begin user program FF PHIS 0879 FC21 35 0880 change user breakpoint position 0881 0882 ensure old is out value for no break change if given 0883 FC23 8D 21 7 BRKSET BSR BRKOUT 37 0884 FC25 8E FFFF LDX #SFFFF 0885 FC28 80 89 RSR NUMBX 0886 FC2A 9F A3 5 STX CBREAK and save 0887 FC2C 39 5 PTS 0888 put breakpoint in if one exists 0889 0890 0891 FC2D 8D 69 7 BRKIN BSR BRKTST see if exists 0892 FC2F 27 66 ż BEQ BRK1 already a swi, so done 0693 FC31 97 A7 ā STA CINST else save it 0E94 FC33 86 35 2 LDA #SWI . × 0895 FC35 A7 84 4 STA and insert a sw1 instead 5 BRK1 0896 FC37 39 RTS **0897** * check break required, if not does rts twice **6898** * **A899** 5 BRKTST LDX CBREAK 0900 FC38 9E нЗ get address \$FFFF means none 0901 FC3A 8C FFFF 4 CMPX #\$FFFF 0902 FC3D 27 05 ÷ DED BRK10 not one 84 3E get instruction 0903 FC3F A6 4 L DA × CMPA #SWI 0904 FC41 81 2 see if swi 5 RTS then exit exit twice 0905 FC43 39 96 ā BRK10 PULS X, PC 0906 FC44 35 0907 0908 remove a breakpoint if one present in code 0909 7 BRKOUT BSR 0910 FC46 8D FØ REFIST see if exists 0911 FC48 26 04 87 3 RNE BRK2 not swi, leave alone get saved instruction LDA 4 0912 FC4A 96 CINST 0913 FC4C A7 4 STA . X and restore in code 84 0914 FC4E 39 5 BRK2 RTS 0915 * trace one instruction of user code 0916 * 0917 0918 FC4F 35 10 7 TUSER1 PULS get return address. × 0919 FC51 9F 5 STX IRORTS and save it 73 L DA ANDA 0920 FC53 86 ΞF 24 #255-IRQ clear irg mask.. F4 , s . in user. 0921 FC55 A4 STA 0922 FC57 A7 F۵ 4 C condition codes З 0500 LDD #INTDEL 0923 FC59 CC delay before interrupt 0924 FC5C FD 0984 6 STD KTICL 0925 FC5F 33 CC, A, B, DP, X, Y, U, PC FF 17 PULS 0926 0927 * set number of instructions to trace on each command **8928** FFØ8 TRACEN LBSR 0929 FC61 17 9 NUMB get 0 if no number 0930 FC64 DD 0931 FC66 39 50 5 STD NTRACE save result 5 TRACEL RTS 0932 trace required number of instructions then display register contents and halt user 0933 0934 0935 0936 FC67 9E 50 5 TRACE LDX NTRACE get number to trace ignore command if zero TRACE 1 0937 FC69 27 FB з BEQ 0938 FC68 32 62 5 LEAS 2, 5 strip return address CTRACE 0939 FC60 9F **A**8 5 TRACE2 STX save number left 0940 FC6F 8D DE 2 BSR TUSER1 trace one instruction 0941 FC71 9E 88 LDX CTRACE get number left.. 5525 30 and decrement 0942 FC73 1F I EAX -1,X TRACE2 0943 FC75 26 F6 BNE repeat if required FF61 LBRA RES1 0944 FC77 16 else show registers and halt user 0945 * 0946 turn printer echo of console output on or off 0947 0948 FC7A 17 FC02 9 PONTL LBSR CONCHR get input if cr then off if plus. 0949 FC7D 27 96 3 BEQ POFF CMPA 0950 FC7F 81 28 2 #1+ 2 0951 FC81 27 63 8FQ PON .. then switch on LASTC re-use if not + switch off value set flag 0952 FC83 97 A2 4 STA CLRA 0953 FC85 4F 2 POFF 4 PON PFLAG 0954 FC86 97 61 STA 0955 FC88 39 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 FC88 1026 FC18 0961 FC8F 8E FFC3 6 3 LBNE MEND . then exit LDX #COPV output query. 0962 FC92 17 FC57 9 LBSR STRING ... message 0963 FC95 17 FD89 ŝ LBSR CONOUT and character followed by cr, lf 0964 FC98 17 FC4E 9 LBSR OPCRLF 0965 FC9B 17 FBE1 9 BAD1 CONCHR get input and., if not or then ignore it LBSR 0366 FC9E 26 FB 3 RNE BAD1 0967 FCA0 39 ŝ RTS then carry on

	*			
8968 8969	*		cassette file 1	oad routine, this searches for named file
0970	*		followed by dat	a.
0971 0972 FCA1 17 0		O LBSR	NAMEIN	get file name
0973 FCA4 17 F	EC5 9	LBSR	NUMB	get offset
	85 5	STD	OFFSET #NAME	and save
	3397 3 LOA 30 2 LOA		#rem⊏	get name file.
0977 FCAE 8D 4	NE 7	BSR	GETHOR	, . header
	A 3	BNE	LÜAD4	ignore others
	% 2 ℃ 4	LDB STB	#6 TEMP	name length save it
0981 FCB6 8D 6	56 7 LOA		CBIN1	get input character
	30 6 XF 2	· LDB CMPB	,X+ ♦′?	and name character if wildcard
	Ar ∡ 34 3	BEQ	LOADI	., then matches
8985 FCBE A1 1	F 5	CMPA	-1, X	else compare
	17 3 NG 61.04	BNE	LOAD2	wrong name
	NC 61.0A 70 3	01 DEC BNE	TEMP LOAD3	count name length repeat for all 6
0989 FCC6 80 5	56 7	BSR	CBIN1	ignore checksum byte
0990 0991	*		anten hans for	match without file name check
8992	*		whiter Here for	match without file mame check
	31 2 LOA		#1	get data.
	27 34 3	BSR BNE	GETHOR LOAD5	.,header wrong header
	.7 5	BSR	CBIN2	get start address
0997 FCD0 1F 0	6 6	TFR	D, X	and save while.
0998 FCD2 8D 3 0999 FCD4 34 0	7 167	BSR PSHS	CBIN2 D	., getting end address put end on stack
	6 7 LOA		CBIN1	get data item.
1001 FCD8 A7 8	6 6	STA	. X+	and store it
	4 6 8 3	CMPX	, S LOADE	if not done.
	E 7	BLS BSR	CBIN1	then repeat get checksum byte
1005 FCE0 35 1	87	PULS	×	get old end address
	D90 9	LBSR	OPXREG	show address so far
1008 FCE7 53	906 2	TFR	U, D	check summed in u lower byte only
1009 FCE8 27 D		BEQ	LOAD7	if ok, then repeat
	FCD 3 BFC 9	LDX	#LORY	else message, ,
1011 FCED 17 F 1012 FCF0 20 0		LBSR BRA	STRING KEYON	., output
1013	*			
1014 FCF2 81 3			#19	if not 'x9' then.
1015 FCF4 26 D 1016 FCF6 86 9		BNE DN L.DA	LOAD7 #CB1FLG+\$80	ignore turn on
	98E 5	STA	KIER	. keyboard
1018 FCFB 16 F	BEB 5	LBRA	OPCR_F	then exit
1019 1020	*			
1021	*			om the tape, if the expected one status, else return non-zero status
1021 1022			then set a zero	
1021 1022 1023	*	HDR PSHS	then set a zero	: status, else return non-zero status : checksum in u to 0
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 10	* * 2 6 Geti 8 2	HDR PSHS LDA	then set a zero initialises the A #CBIFLG	i status, else return non-zero status i checksum in u to 0 save character turn off.
1021 1022 1023 1024 FCFE 34 02 1025 FD00 86 11 1026 FD02 B7 02	* * 2 6 GETI 8 2 98E 5	LDA STA	then set a zero initialises the A #CBIFLG KIER	status, else return non-zero status checksum in u to 0 save character turn off ., keyboard
1021 1022 1023 1024 FCFE 34 02 1025 FD00 86 10 1026 FD02 87 02 1027 FD05 80 12	* * 2 6 GETI 0 2 98E 5 7 7 GETI	LDA STA ID1 BSR	then set a zero initialises the A #CBIFLG KIER CBIN1	istatus, else return non-zero status :checksum in u to 0 save character turn off. keyboard get from tape
1021 1022 1023 1024 FCFE 34 0 1025 FD09 86 11 1026 FD02 B7 0 1027 FD05 80 11 1028 FD07 81 D0 1029 FD09 26 Ff	* * 9 6 GETI 9 2 98E 5 7 7 GETI 8 2 A 3	LDA STA HD1 BSR CMPA BNE	then set a zero initialises the A #CBIFLG KIER	: status, else return non-zero status : checksum in u to 0 ; sure character turn off keyboard get from tape., .and if not 'x' then .try again
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 11 1026 FD05 80 11 1028 FD05 80 11 1028 FD07 81 D0 1029 FD09 80 1	* 2 6 GETI 9 2 98E 5 7 7 GETI 8 2 A 3 1 7	LDA STA HD1 BSR CMPA BNE BSR	then set a zero initialises the MCBIFLG KIER CBIN1 #'X+\$80 GETHD1 CBIN1	istatus, else return non-zero status ichecksum in u to 0 save character turn off. keyboard get from tape. and if not 'x' then try again get next character
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 11 1026 FD02 87 00 1027 FD05 80 11 1028 FD07 81 00 1029 FD09 26 FI 1030 FD06 80 1 1031 FD06 80	* * 9 2 6 Geti 9 8 5 5 7 7 Geti 8 2 A 3 1 7 000 3	LDA STA HD1 BSR CMPA BNE BSR LDU	then set a zero initialises the MCBIFLG KIER CBINI 4'X+800 GETHD1 CBINI #0	istatus, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape., and if not 'x' then., try again get next character setup checksum
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 11 1026 FD05 80 11 1028 FD05 80 11 1028 FD07 81 D0 1029 FD09 80 1	* + 2 6 GET 9 8 5 7 7 6 ET 8 2 A 3 1 7 8000 3	LDA STA HD1 BSR CMPA BNE BSR	then set a zero initialises the MCBIFLG KIER CBIN1 #'X+\$80 GETHD1 CBIN1	istatus, else return non-zero status ichecksum in u to 0 save character turn off. keyboard get from tape. and if not 'x' then try again get next character
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 11 1026 FD02 87 00 1027 FD02 87 00 1028 FD07 81 00 1029 FD09 26 FT 1030 FD08 80 1 1031 FD00 CE 00 1032 FD12 39 1034	* * 2 6 GETI 98E 5 7 7 GETI 8 2 A 3 1 7 8000 3 8000 5 5 5	LDA STA 101 BSR CMPA BNE BSR LDU CMPA	then set a zero initialises the A WCBIFLG KIER CBIN1 #'X+\$80 GETHO1 CBIN1 #0 , S+	: status, else return non-zero status : checksum in u to 0 save character turn off. keyboard get from tape., .and if not 'x' then., .try again get next character setup checksum and compare with required then return
1021 1022 1023 1024 FCFE 34 01 1025 FD00 86 11 1026 FD02 87 01 1027 FD09 26 F7 1028 FD07 81 D0 1029 FD09 26 F7 1030 FD08 6D 11 1031 FD00 CE 01 1032 FD10 A1 E1 1033 FD12 39 1034	* 2 6 GETI 9 2 58E 5 7 7 GETI 8 2 1 7 000 3 000 3 0 6 5 * *	LDA STA 101 BSR CMPA BNE BSR LDU CMPA	then set a zero initialises the A #CBIFLG KIER CBIN1 #'X+\$90 GETHD1 CBIN1 #0 , S+ get 2 bytes and	<pre>i status, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d</pre>
1021 1022 1023 1024 FCFE 34 00 1025 FD00 86 11 1026 FD02 87 00 1027 FD09 86 11 1028 FD07 81 00 1029 FD09 26 F1 1038 FD09 26 F1 1031 FD00 CE 00 1032 FD10 A1 E1 1033 FD12 39 1034 1035 1036	* 2 6 GET 9 2 96E 5 7 7 0FT 8 2 96E 5 7 7 0FT 8 2 96E 9 6 6 5 * * *	LDA STA HD1 BSR CHPA BNE BSR LDU CHPA RTS	then set a zero initialises the A #CBIFLG KIER CBIN1 #'X+\$80 GETHD1 CBIN1 #0 , S+ get 2 bytes and add offset sinc	istatus, else return non-zero status ichecksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address
1021 1022 1023 1025 FD00 85 11 1025 FD00 86 11 1026 FD02 87 0 1027 FD00 86 11 1028 FD07 81 0 1029 FD09 26 FT 1030 FD08 8D 1 1031 FD00 CE 00 1032 FD10 A1 E1 1033 FD12 39 1034 1035 1037	* 2 6 GET 9 2 6 GET 96E 5 7 7 GET 8 2 A 3 1 7 8008 3 0 6 5 * * * 9 7 CBI	LDA STA 5TA BSR CMPA BNE BSR LDU CMPA RTS	then set a zero initialises the A #CBIFLG KIER CBIN1 @CBIN1 @GETHD1 CBIN1 #0 get 2 bytes and add offset sinc CBIN1	<pre>istatus, else return non-zero status checksum in u to 0 sue character turn off keuboard get from tape., . and if not 'x' then., . try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD06 86 11 1026 FD02 87 67 1028 FD07 81 07 1028 FD07 81 07 1028 FD07 81 07 1030 FD08 80 1 1031 FD00 CE 07 1032 FD16 A1 E1 1033 FD12 39 1034 1035 1036 1037 FD13 80 67 1038 FD13 80 67	* 2 6 GET 9 6E 5 7 7 6FT 8 2 6 7 7 6FT 8 2 6 1 7 8 6 8 6 5 * * * 9 7 CBI 9 6	LDA STA 4D1 BSR CMPA BNE BSR LDU CMPA RTS 42 BSR TFR	then set a zero initialises the A WCBIFLG KIER CBIN1 4'X+\$800 GETHD1 CBIN1 #0 , S+ get 2 bytes and add offset sinc CBIN1 A,B	<pre>i status, else return non-zero status checksum in u to 0 save character turn off. keuboard get from tape. . and if not 'x' then. tru again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while.</pre>
1021 1022 1023 1025 FD00 85 11 1025 FD00 86 11 1026 FD02 87 0 1027 FD00 86 11 1028 FD07 81 0 1029 FD09 26 FT 1030 FD08 8D 1 1031 FD00 CE 00 1032 FD10 A1 E1 1033 FD12 39 1034 1035 1037	* 2 6 GETI 9 2 56E 5 7 7 GETI 8 2 1 7 8000 3 6 6 5 8 4 * 9 6 6 7 7 CBII 9 6 7	LDA STA 5TA BSR CMPA BNE BSR LDU CMPA RTS	then set a zero initialises the A #CBIFLG KIER CBIN1 @CBIN1 @GETHD1 CBIN1 #0 get 2 bytes and add offset sinc CBIN1	<pre>istatus, else return non-zero status checksum in u to 0 sue character turn off keuboard get from tape., . and if not 'x' then., . try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte</pre>
1021 1022 1023 1024 FCFE 34 07 1025 FD00 86 11 1026 FD02 87 07 1028 FD07 81 D1 1028 FD07 81 D1 1029 FD09 26 F1 1032 FD109 26 07 1033 FD12 39 1034 1035 1023 1036 1037 1038 FD13 8D 07 1039 FD13 8D 07 1049 FD17 8D 07 1041 FD19 1E 07 1041 FD19 18 07 1042 FD18 D3 87	* 2 6 GET 9 2 986 5 7 7 GET 8 2 986 5 7 7 GET 8 2 986 3 9 6 6 5 * * 9 6 5 7 9 7 5 6	LDA STA 101 BSR BSR BSR LDU CHPA RTS V2 BSR TFR BSR EXG ADDD	then set a zero initialises the A WCBIFLG KIER CBIN1 */X+\$90 GETHD1 CBIN1 #0 , S+ get 2 bytes and add offset sinc CBIN1 A, B CBIN1	<pre>i status, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. get second</pre>
1021 1022 1022 5000 86 11 1025 5000 86 11 1025 5000 86 11 1026 502 87 81 01 1028 5007 81 01 1029 5009 26 F1 1030 5006 80 1 1031 5000 CE 00 1032 5012 39 1034 1035 1033 5013 80 81 1036 5013 80 81 1041 5013 16 81 1041 5013 16 81 1042 5018 03 81 1041 5013 16 81 1042 5018 03 81 1043 5013 91 16 81 1043 5013 91 16 81 1044 5019 18 81 1043 5013 91	* 2 6 GET 9 96E 5 7 7 6 GET 8 2 6 1 7 7 6 GET 8 3 6 0 6 5 5 * * 5 7 9 7 CBII 9 6 5 7 9 7 7	LDA HD1 BSR CMPA BNE BSR LDU CMPA RTS V2 BSR TFR BSR EXG	then set a zero initialises the A WCBIFLG KIER CBIN1 4'X+\$800 GETHO1 CBIN1 40 ,S+ get 2 bytes and add offset sinc CBIN1 A, B CBIN1 A, B	<pre>istatus, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second wrong order, swap over</pre>
1021 1022 1023 1024 FCFE 34 07 1025 FD00 86 11 1026 FD02 87 07 1028 FD07 81 D1 1028 FD07 81 D1 1029 FD09 26 F1 1032 FD109 26 07 1033 FD12 39 1034 1035 1023 1036 1037 1038 FD13 8D 07 1039 FD13 8D 07 1049 FD17 8D 07 1041 FD19 1E 07 1041 FD19 18 07 1042 FD18 D3 87	* 2 6 GET 9 2 986 5 7 7 GET 8 2 986 5 7 7 GET 8 2 986 3 9 6 6 5 * * 9 6 5 7 9 7 5 6	LDA STA 101 BSR BSR BSR LDU CHPA RTS V2 BSR TFR BSR EXG ADDD	then set a zero initialises the A WCBIFLG KIER CBIN1 * ***500 GETHD1 CBIN1 #0 , S+ get 2 bytes and add offset sinc CBIN1 A, B CBIN1 A, B OFFSET	<pre>istatus, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second wrong order, swap over</pre>
1021 1022 1024 1025 1024 FCFE 34 (2) 1025 FD00 86 11 1026 FD02 87 (2) 1028 FD07 81 (2) 1028 FD07 81 (2) 1029 FD08 80 (1) 1030 FD08 80 (1) 1031 FD00 CE (2) 1032 FD12 39 1033 FD12 39 1033 FD12 39 1035 1036 FD13 (8) (2) 1037 FD15 (1) 1037 FD15 (1) 1037 FD15 (1) 1044 FD17 80 (2) 1044 FD19 (1) (2) 1044 FD19 (1) (2) 1044 FD10 39 1044 FD10 39 1044 FD10 39 1044 FD10 39	* 2 6 GET 9 2 966 5 7 7 GET 8 2 0 6 7 7 GET 8 2 9 6 0 6 5 * * 9 7 CBI 9 6 5 7 9 7 5 6 5 5 7 9 7 5 6 5 * * * * * * * * * * * * * * * * * *	LDA STA D1 BSR BSR BSR CMPA BSR CMPA RTS V2 BSR TFR BSR ESG A000 RTS	then set a zero initialises the A WCBIFLG KIER CBIN1 #'X+\$800 GETHD1 CBIN1 #0 ,S+ get 2 bytes and add offset sinc CBIN1 A,B CBIN1 A,B OFFSET get 1 byte from	<pre>istatus, else return non-zero status checksum in u to 0 save character turn off. keuboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second urong order, swap over move by offset tape, modifying checksum to suit</pre>
1021 1022 1023 1025 FD06 86 11 1025 FD06 86 11 1026 FD02 87 0 1027 FD06 80 11 1028 FD07 81 D0 1029 FD08 8D 1 1028 FD07 81 D0 1034 FD06 8D 1 1034 FD06 8D 1 1035 1037 1037 1037 1 1038 FD13 8D 0 1049 FD13 1F 0 1044 FD19 1E 0 1044 FD19 1E 0 1044 FD19 10 3 1044 FD19 10 1045 1045 1045 1045 1045 1045 1045 1045	* 2 6 GETI 996E 5 7 7 GETI 8 2 GATI 8 2 GATI 8 3 1 9068 3 9068 3 9 9 6 5 7 9 7 CBII 9 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 7 9 7 5 6 5 7 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	LDA STA FD1 BSR BNE BNE LDU CMPA RTS V2 BSR TFR BSR EXG A0000 RTS	then set a zero initialises the A MCBIFLG KIER CBIN1 eX+\$80 GETHD1 CBIN1 00 , S+ get 2 bytes and add offset sinc CBIN1 A, B OFFSeT get 1 byte from [CASINA]	<pre>i status, else return non-zero status i checksum in u to 0 save character turn off keuboard get from tape., . and if not 'x' then., try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while., and save while., and save while. and save while. </pre>
1021 1022 1023 1025 FD06 86 11 1025 FD06 86 11 1026 FD02 87 61 1028 FD07 81 01 1028 FD07 81 01 1028 FD07 81 01 1029 FD08 8D 1 1030 FD06 8D 1 1031 FD00 CE 00 1032 FD16 A1 E1 1033 FD12 39 1034 1035 1037 1038 FD13 8D 81 1037 1038 FD13 8D 81 1048 FD13 8D 81 1044 FD19 1E 61 1044 FD19 18 03 1044 FD19 32 81 1045 1045 FD1E AD 99 1044 FD12 33 C1	* 2 6 GETI 996E 5 7 7 GETI 8 2 GATI 8 2 GATI 8 3 1 9068 3 9068 3 9 9 6 5 7 9 7 CBII 9 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 7 9 7 5 6 5 7 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 7 9 9 7 5 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	LDA STA D1 BSR BSR BSR CMPA BSR CMPA RTS V2 BSR TFR BSR ESG A000 RTS	then set a zero initialises the A WCBIFLG KIER CBIN1 #'X+\$800 GETHD1 CBIN1 #0 ,S+ get 2 bytes and add offset sinc CBIN1 A,B CBIN1 A,B OFFSET get 1 byte from	<pre>istatus, else return non-zero status checksum in u to 0 save character turn off. keuboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second urong order, swap over move by offset tape, modifying checksum to suit</pre>
1021 1022 1022 1024 FCFE 34 03 1025 FD00 86 11 1025 FD00 86 11 1026 FD02 87 03 1027 FD00 86 11 1028 FD07 81 00 1029 FD09 26 FF 1030 FD08 80 1 1031 FD00 CE 04 1033 FD12 39 1034 1035 1036 FD08 80 1 1035 1033 FD13 80 03 1043 FD13 80 03 1043 FD13 1F 03 1044 FD17 90 03 1044 FD17 90 03 1044 FD19 1E 03 1045 1044 FD1 39 1045 1045 104 1045 1044 FD1 39 1046 1047 FD1E AD 94 1048 FD22 33 C1 1049 FD24 39	* 2 6 GET 9 6 E 5 7 7 6 ETI 8 2 6 1 7 7 6 ETI 8 3 1 7 8 600 3 0 6 5 7 9 7 7 5 6 5 7 9 7 CBII 9 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 7 5 7 9 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	LDA STA HD1 BSR BNE BSR CMPA RTS V2 BSR TFR BSR ADDD RTS V1 JSR LEAU	then set a zero initialises the A WCBIFLG KIER CBIN1 *X+\$800 GETHD1 CBIN1 0 (CBIN1 0 0 (CBIN1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre>status, else return non-zero status checksum in u to 0 save character turn off. keuboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then. add to checksum</pre>
1021 1022 1023 1024 FCFE 34 02 1025 FD00 86 11 1026 FD02 87 02 1027 FD00 86 11 1028 FD07 81 02 1029 FD09 26 F1 1038 FD08 80 1 1031 FD00 CE 02 1033 FD12 39 1034 1035 1023 10 1037 FD13 10 02 1036 FD13 80 02 1044 FD17 80 02 1041 FD19 1E 62 1044 FD17 80 02 1044 FD19 18 1044 FD19 18	* 2 6 GET 2 56E 5 7 7 GET 8 4 3 7 600 3 6 5 * * 9 7 CBI 9 6 5 7 5 6 5 7 5 6 5 7 5 7 5 6 7 5 7 5 6 7 5 7 5	LDA STA HD1 BSR BNE BSR CMPA RTS V2 BSR TFR BSR ADDD RTS V1 JSR LEAU	then set a zero initialises the A MCBIFLG KIER CBIN1 eX+\$500 GETHD1 CBIN1 H0 A get 2 bytes and add offset sinc CBIN1 A, B OFFSeT get 1 byte from [CASINA] A, U software asynch	<pre>i status, else return non-zero status i checksum in u to 0 save character turn off keuboard get from tape., . and if not 'x' then., try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while., and save while., and save while. and save while. </pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1026 FD02 87 67 1028 FD07 81 07 1028 FD07 81 07 1028 FD07 81 07 1030 FD08 80 1 1031 FD00 CE 07 1032 FD16 A1 E1 1033 FD12 39 1034 FD12 39 1035 1036 FD13 80 67 1037 FD15 1F 68 1044 FD17 80 67 1044 FD17 80 67 1044 FD19 1E 69 1044 FD19 18 97 1044 FD19 38 1044 FD19 38 1044 FD10 39 1044 FD12 33 1049 FD22 33 1051 1052 1053 1053	* * * 2 6 GEU 96E 5 7 7 GET 8 2 6 1 7 8 9 7 GET 9 6 5 7 9 7 CBI 9 6 5 7 9 7 CBI 9 7 CBI 5 7 5 7 5 7 5 7 5 7 5 7 5 7 5 7	LDA STA HD1 BSR BNE BSR CMPA RTS V2 BSR TFR BSR ADDD RTS V1 JSR LEAU	then set a zero initialises the A #CBIFLG KIER CBINI @CBINI @GETHDI CBINI #0 , S+ get 2 bytes and add offset sinc CBINI A, B OFFSET get 1 byte from [CASINA] A, U software asynch as start bit, B by count in del	<pre>status, else return non-zero status checksum in u to 0 sue character turn off keyboard get from tape .and if not 'x' then .try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then ad to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cht</pre>
1021 1022 1023 1024 FCFE 34 03 1025 FD00 86 11 1026 FD02 87 03 1027 FD00 86 11 1028 FD07 81 05 1029 FD09 26 FF 1030 FD06 80 1 1031 FD00 CE 00 1032 FD12 39 1034 FD00 CE 00 1033 FD12 39 1034 FD12 39 1034 FD12 80 03 1035 1035 1F 83 1040 FD17 8D 03 1044 FD19 1E 83 1044 FD19 1E 83 1044 FD19 1E 83 1044 FD19 3 1045 1045 1045 1045 FD1E AD 99 1045 1053 1055 1055 1055	* 2 6 GET 9 6 E 5 7 7 6 ETI 8 2 6 1 7 7 6 ETI 8 3 1 9 6 6 9 6 5 7 7 9 7 6 5 7 9 7 6 5 7 9 7 5 5 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	LDA STA HD1 BSR BNE BSR CMPA RTS V2 BSR TFR BSR ADDD RTS V1 JSR LEAU	then set a zero initialises the A WCBIFLG KIER CBINI #X+\$80 GETHOI CBINI #0 GETHOI CBINI A B CBINI A,B CCASINI A,B CCASINI A,B CCASINI A,B CCASINI A,B CCASINI A,B CCASINA A,B CCASINA A,B CCASINA A,B CCASINA A,C CCASINA CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA A,C CCASINA CCA	<pre>status, else return non-zero status checksum in u to 0 sue character turn off keyboard get from tape .and if not 'x' then .try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then ad to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cht</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1026 FD02 87 67 1028 FD07 81 07 1028 FD07 81 07 1028 FD07 81 07 1030 FD08 80 1 1031 FD00 CE 07 1032 FD16 A1 E1 1033 FD12 39 1034 FD12 39 1035 1036 FD13 80 67 1037 FD15 1F 68 1044 FD17 80 67 1044 FD17 80 67 1044 FD19 1E 69 1044 FD19 18 97 1044 FD19 38 1044 FD19 38 1044 FD10 39 1044 FD12 33 1049 FD22 33 1051 1052 1053 1053	2 6 GE 2 6 GE 2 96E 5 7 7 GET 8 2 6 1 7 8 9 7 GE 9 7 CB 9 7 CB 9 7 CB 9 7 CB 5 7 5 7 5 7 5 7 5 7 5 7 5 7 5 7	LDA STA STA BSR DI BSR BNE BSR LDU CMPA RTS V2 BSR TFR BSR EXG A0000 RTS V1 JSR LEAU RTS	then set a zero initialises the A #CBIFLG KIER CBINI @CBINI @GETHDI CBINI #0 , S+ get 2 bytes and add offset sinc CBINI A, B OFFSET get 1 byte from [CASINA] A, U software asynch as start bit, B by count in del	<pre>status, else return non-zero status checksum in u to 0 sue character turn off keyboard get from tape .and if not 'x' then .try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then ad to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cht</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1025 FD00 86 11 1026 FD02 87 67 1028 FD07 81 07 1028 FD07 81 07 1029 FD09 26 FT 1030 FD08 80 1 1031 FD00 CE 07 1032 FD12 39 1034 FD12 39 1034 FD12 39 1035 1035 FD13 90 67 1043 FD13 90 67 1044 FD17 80 67 1044 FD17 80 67 1044 FD17 80 67 1044 FD17 80 67 1045 FD23 33 1044 FD12 39 1044 FD12 33 1044 FD12 39 1044 FD12 33 1045 1055 1055 FD27 1A 57 1057 FD27 1A 57 1057 FD27 1A 57 1055 FD27 FD27 FD27 FD27 FD27 1055 FD27 FD27 FD27 FD27 FD27 1055 FD27 FD27 FD27 FD27 1055 FD27 FD27 FD27 1055 FD27 FD27 FD27 1055 FD27 FD27 1055 FD27 FD27 FD27 1055 FD27 FD27 1055 FD27 FD27 1055 FD27 FD27 1055 FD27 FD27 1055 FD27 FD27 1055 FD27 1055 FD27 FD27 1055 FD2	* 2 6 GET 2 96E 5 7 7 GETI 8 2 GET 8 7 6 GET 8 6 5 8 7 6 GET 9 7 CBII 9 6 5 5 7 9 7 CBII 9 6 5 5 7 9 7 CBII 9 6 5 5 7 9 7 7 5 6 5 5 * * * 7 10 MCAt 8 3	LDA STA STA BSR DI BSR BSR LDU CMPA RTS V2 BSR TFR BSR EXG ADDD RTS V1 JSR LEAU RTS SEIF	then set a zero initialises the A WCBIFLG KIER CBIN1 #X+\$800 GETHD1 CBIN1 #6 ,S+ get 2 bytes and add offset sinc CBIN1 A,B CBIN1 A,B OFFSET get 1 byte from [CASINA] A,U software asynch as start bit, 8 by count in del saves all regis CC,D,X	<pre>status, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then. add to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cnt ters</pre>
1021 1022 1023 1024 FCFE 34 (1) 1025 FD06 86 (1) 1025 FD06 86 (1) 1026 FD02 87 (2) 1028 FD07 81 D1 1028 FD07 81 D1 1029 FD08 8D 1 1029 FD08 8D 1 1034 FD06 80 (1) 1033 FD12 39 1034 FD18 30 (2) 1035 FD17 8D (2) 1049 FD13 IF (2) 1044 FD13 1E (2) 1044 FD19 1E (2) 1045 FD18 D3 (2) 1051 FD18 D4 (2) 1052 FD18 D4 (2) 1055 FD25 34 (1) 1055 FD25 C6 (2) (2) 1055 FD25 C6 (2) 1051 FD25 C6 (2) 1051 FD25 C6 (2) 1052 FD25 C6 (2) 1051 FD25 FD25 FD25 FD25 FD25 FD25 FD25 FD25	* 2 6 GETI 996E 5 7 7 GETI 8 2 1 7 7 8 6 5 7 9 6 5 7 9 7 CBII 9 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 5 6 5 7 9 7 10 MCM 8 2 8 2	LDA STA STA BSR DI BSR BNE BSR LDU CHPA RTS V2 BSR EXG ADOD RTS V3 JSR LEAU RTS SOP PSHS SEIF LDB	then set a zero initialises the A MCBIFLG KIER CBIN1 eX+\$80 GETHD1 CBIN1 %0 , S+ get 2 bytes and add offset sinc CBIN1 A, B OFFSet get 1 byte from [CASINA] A, U software asynch as start bit, 8 by sount in del saves all regis CC, D, X %11	<pre>status, else return non-zero status checksum in u to 0 save character turn off keuboard get from tape., .and if not 'x' then., .tru again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while., and save while., get second urong order, swap over move by offset tape, modifying checksum to suit get byte then., add to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cht ters</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1025 FD00 86 11 1026 FD02 87 67 1028 FD07 81 07 1028 FD07 81 07 1029 FD08 80 1 1034 FD09 26 FF 1036 FD08 80 1 1033 FD12 39 1034 FD12 39 1034 1035 1035 1038 FD13 80 67 1037 80 67 1038 FD13 80 67 1041 FD15 1F 67 1042 FD18 D3 87 1044 FD17 90 63 1045 FD24 39 1045 1024 39 1046 103 1047 FD1E AD 97 1048 FD22 33 1051 1055 12 1055 FD25 34 11 1057 FD27 1A 55 1056 FD25 34 11 1055	* 2 6 GEV 96E 5 7 7 GETI 8 2 6 7 7 GETI 8 2 6 9 6 5 9 7 CBII 9 7 CBII 9 7 CBII 9 7 CBII 9 7 CBII 9 7 CBII 9 7 5 5 * * * * * * * * * * * * * * * * * * *	LDA STA STA BSR DI BSR BSR LDU CMPA RTS V2 BSR TFR BSR EXG ADDD RTS V1 JSR LEAU RTS SEIF	then set a zero initialises the A WCBIFLG KIER CBIN1 #X+\$800 GETHD1 CBIN1 #6 ,S+ get 2 bytes and add offset sinc CBIN1 A,B CBIN1 A,B OFFSET get 1 byte from [CASINA] A,U software asynch as start bit, 8 by count in del saves all regis CC,D,X	<pre>status, else return non-zero status checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then. add to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cnt ters</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1026 FD02 87 68 1027 FD00 86 11 1028 FD07 81 07 1028 FD07 81 07 1029 FD09 26 FF 1030 FD06 80 1 1031 FD00 CE 00 1032 FD12 39 1034 FD12 39 1034 FD13 80 68 1035 FD13 80 68 1037 FD13 80 68 1038 FD13 80 68 1039 FD13 1F 88 1041 FD19 1E 68 1042 FD18 D3 88 1042 FD18 D3 88 1044 FD17 90 81 1045 FD22 33 C1 1049 FD22 33 1049 FD22 33 1049 FD22 33 1050 FD25 34 11 1055	* 2 6 GET 9 6 7 7 6 FT 8 2 6 6 7 7 7 6 FT 8 2 6 7 8 3 1 7 9 60 6 6 9 6 5 7 9 7 6 5 7 9 7 6 5 7 9 7 6 5 7 9 7 6 5 7 9 7 7 5 6 5 7 9 7 7 5 6 5 7 9 7 7 5 6 5 7 9 7 7 5 6 7 9 7 7 5 6 7 9 7 7 5 6 7 9 7 7 5 7 10 MCA 8 2 4 6 6 7 2	LDA STA STA BSR BSR LSR CMPA RTS V2 BSR TFR BSR EXG ADOD RTS V1 JSR LEAU RTS V1 JSR LEAU RTS SOP PSHS SEIF SEIF LDB PSHS LDB COIIA	then set a zero initialises the A WCBIFLG KIER CBINI ex+s80 GETHOI CBINI dd offset sinc CBINI A,B OFFSeT get 1 byte from [CASINA] A,U software asynch as start bit, 8 by yount in del saves all regis CC,D,X #11 B	<pre>i status, else return non-zero status i checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. and save of save of save move by offset tape, modifying checksum to suit get byte then. and to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cnt ters keep timing total length save on stack low start bit want data inverted</pre>
1021 1022 1024 1025 1024 FCFE 34 1025 FD00 86 1025 FD00 86 1027 FD00 80 1028 FD07 81 1028 FD07 81 1028 FD07 81 1029 FD08 80 1029 FD08 926 F1 1031 FD00 CE 1032 FD12 39 1033 FD12 39 1034 1035 1036 FD13 90 81 1049 FD15 1F 1041 FD15 1E 1044 FD17 80 81 1044 FD17 80 81 1045 FD18 33 1044 1045 1044 FD12 33 1044 1045 1044 FD12 40 1055 1055 1055 FD27 34 1055 FD27 1A 1055 FD27 1	* 2 6 GET 3 7 7 GETI 8 2 7 7 GETI 8 7 7 GETI 9 6 6 5 7 7 68 9 7 CBII 9 6 6 5 7 7 9 7 CBII 9 6 5 5 7 9 7 CBII 9 6 5 7 10 MCA 3 8 2 4 6 5 2 2 MCA	LDA STA STA BSTA BST BSR LDN CHPA RTS V2 BSR ESR ESR ESR ESR ESR ADDD RTS V3 SOP PSHS SEIF LDB COHA SO1 NOP	then set a zero initialises the A #CBIFLG KIER CBIN1 #Z + #80 GETHD1 CBIN1 #0 , S+ get 2 bytes and add offset sinc CBIN1 A, B OFFSET get 1 byte from [CASINA] A, U software asynch as start bit, 8 by count in del saves all regis CC, D, X #11 B #KPCR-COPBIT	<pre>istatus, else return non-zero status i checksum in u to 0 sue character turn off kupboard get from tape . and if not 'x' then try again get next character setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte and save while get second wrong order, swap over move by offset tape, modifying checksum to suit get byte then add to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cht ters keep timing total length save on stack low start bit want data inverted get timing.</pre>
1021 1022 1023 1024 FCFE 34 67 1025 FD00 86 11 1026 FD02 87 68 1027 FD00 86 11 1028 FD07 81 07 1028 FD07 81 07 1029 FD09 26 FF 1030 FD06 80 1 1031 FD00 CE 00 1032 FD12 39 1034 FD12 39 1034 FD13 80 68 1035 FD13 80 68 1037 FD13 80 68 1038 FD13 80 68 1039 FD13 1F 88 1041 FD19 1E 68 1042 FD18 D3 88 1042 FD18 D3 88 1044 FD17 90 81 1045 FD22 33 C1 1049 FD22 33 1049 FD22 33 1049 FD22 33 1050 FD25 34 11 1055	* 2 6 GET 3 7 7 GETI 8 2 7 7 GETI 8 7 7 GETI 9 6 6 5 7 7 68 9 7 CBII 9 6 6 5 7 7 9 7 CBII 9 6 5 5 7 9 7 CBII 9 6 5 7 10 MCA 3 8 2 4 6 5 2 2 MCA	LDA STA STA BSR BSR LSR CMPA RTS V2 BSR TFR BSR EXG ADOD RTS V1 JSR LEAU RTS V1 JSR LEAU RTS SOP PSHS SEIF SEIF LDB PSHS LDB COIIA	then set a zero initialises the A WCBIFLG KIER CBINI ex+s80 GETHOI CBINI dd offset sinc CBINI A,B OFFSeT get 1 byte from [CASINA] A,U software asynch as start bit, 8 by yount in del saves all regis CC,D,X #11 B	<pre>i status, else return non-zero status i checksum in u to 0 save character turn off. keyboard get from tape. . and if not 'x' then. try again get next character. setup checksum and compare with required then return form a 16 bit value in d e is address get 1 byte. and save while. and save of save of save move by offset tape, modifying checksum to suit get byte then. and to checksum ronous transmitter, outputs value in a data bits, 2 stop bits, at rate controlled cnt ters keep timing total length save on stack low start bit want data inverted</pre>

6 MCASO2 DEC bits counter 1064 FD33 66 E4 . S 1065 FD35 28 ØE MCAS03 all done BMI 1066 FD37 F7 0980 STB PUDA put out bit 5 wait 1 bit time assume next is zero 1067 FD3A 8D ØÐ ~ BSR CWAIT #255-COPBIT 1068 FD3C C4 DE 2 ANDB I CDA get next hit 1069 FD3E 44 23 FF MCASO1 do want zero. BCS 1070 FD3F 25 1071 FD41 CA 20 ž ORB #COPBIT . else set bit 3 MCAS02 and loop round 1072 FD43 20 FF BRA 1073 FD45 32 1074 F047 35 61 5 MCHSO3 LEAS 1, S remove counter and return 1074 FD47 97 12 PULS CC. D. X. PC 1075 * 1076 ÷ cwait waits for 1 bit time, destroys × hwait waits 1/2 bit time, also destroys x 1077 . 1078 * SETTR 0000 1679 a 1080 FD49 80 7 CWAIT HUATT do first half 99 BSP 0363 6 HWA1T LDX DELCNT get count required 1081 FD48 PF 5 HW1 LEAX decrement. . 1882 ED4E 38 1F -1.X 1083 FD50 26 FC ž BNE HWI ... while non-zero 1084 FD52 39 5 RTS 1085 0003 SETDP MONDP 1086 * 1987 * software asynchronous receiver, gets value into a saves all other registers, only gets 1 stop bit 1088 * 1089 1090 ED53 34 8 MCASIN PSHS 8. X 14 1091 FD55 86 #\$80 80 LDA rotating counter 2 1092 F057 F6 0980 5 MCASI1 LDB KIRB wait for start bit 1093 FD5A 2B з BMI MCASI1 FB BSR HWAIT wait 1/2 bit time 1094 FD5C 7 80 ED 1095 FD5E 5 LDB KIRE recheck. F6 0980 1096 FD61 2B F4 37 8MT MCAST 1 . start bit MCAST2 BSR wait whole bit time 1097 FD63 8D E4 CHATT and get input waste time to. 5 KIRB 1098 FD65 F6 8988 LDB 67 CMPX , s 1899 F068 AC F4 ...match loop delaus CMPS ŝŝ 1100 FD6A 11AC E4 2 LSLB move bit to carry. 1101 FD6D 58 1102 FD6E RORA 46 237 F2 D6 MCAS12 1104 FD71 8D BSR CWAIT 1105 FD73 35 94 10 B, X, PC PULS 1106 * 1107 * routine gets name from input stream, up to 6 1108 * characters long, no name leaves memory unaltered. any name is padded to 6 characters with spaces. 1109 * 1110 * 1111 FD75 8E 0390 3 NAMEIN LOX #NAME+6 1112 FD78 17 FB04 9 NAM2 LBSR CONCHR get a character 1113 FD7B 27 23 ž NAM1 BEQ 1114 FD7D 81 ž CMPA #SPACE if space . 20 1115 FD7F 27 F7 ā BEQ NAM2 .. ignore 1116 FD81 81 1117 FD83 27 #COMMA 20 2 CMPA 18 З REO NAM1 null name 1118 FD85 C6 #256-6 FA 2 IDR minus name length 1119 FD87 A7 1120 FD89 50 1119 FD87 85 5 NAM3 STA B, X store a letter INCB 2 and move up 1121 FD8A 27 15 ž NAM6 done 6 chars, RFO evit 1122 FD8C š CONCHR get next letter 17 FAFØ LBSR 1123 FD8F 27 88 ż BEQ NAM5 on cr, pad name 1124 FD91 81 20 2 CMPA #SPACE on space. 1125 FD93 27 3 NAMA ...pad name if not commu **9**6 **BEO** 1126 FD95 81 1127 FD97 26 #COMMA 20 2 CMPA FE 2 BNE NAM3 . then use 1128 FD99 86 2 NAM5 #SPACE 20 I DA padding 1129 FD98 A7 85 5 NAM4 STA B.X pad until. 1130 FD9D 5C 2 INCB .. end of. . FB 1131 FD9E 26 з BNE NAM4 ... name buffer 1132 FDA0 39 5 NAM1 RTS 1133 FDA1 17 FADB ā NAM6 BSR CONCHR get next input 1134 FDA4 27 FA 3 BEQ NAM1 leave if cr 1135 FDA6 81 20 2 CMPA #COMMA else absorb comma 1136 FDA8 27 F6 з BEQ NAM1 1137 FDAA 97 A2 4 STA LASTC else re-use 1138 FDAC 39 5 PTS :139 1140 * save files on cassette, dumps name block, data blocks as required in 256 byte blocks maximum, then end file block 1141 * 1142 * 1143 . can also inhibit end of file block 1144 * 1145 FDAD 9E 90 5 SAVE LOX CSSTRT modify start address.. 1146 FDAF 17 FE31 9 5 LBSR NUMBX .. if. . 1147 FDB2 9F 90 STX CSSTRT ... required 1148 FD84 9E 9F LOX CSEND modify end address.. 5 9 5 1149 FDB6 17 FE2A 1858 NIMRY .. if. . 1150 FD89 9F 9F STX CSEND ... required 1151 FD98 80 88 30 727 BSR NAMEIN and get name 1152 FDBD C6 LDB #'A output name. 1153 FDBF 8D зč BSR XHEAD . header 1154 FDC1 сs 06 2577 LDB #6 name length 1155 FDC3 30 18 LEAX -6, X point to name 1156 FDC5 80 1157 FDC7 80 46 BSR DATOUT output name .. then checksum 4F RSP CHKOUT 1158 FDC9 9E **9D** 5 t DX CSSTRT get start address 1159 FDCB 34 10 7 SAUG PSHS × save start 1160 FDCD DC LDD CSEND 9F 5 and get end address 88

	-	_		.	
1161 FDCF A3 1162 FDD1 25	E1 1D	9 3	subd Blo	, S++ SAU2	form length needed done all output
1163 FDD3 4D		2	TSTA		if <=256 then.
1164 FDD4 27 1165 FDD6 CC	03 00FF	3	BEQ LDD	SAU5 #255	., leave alone ., else set to 256
1166 FDD9 33	88	8 SAV5	LEAU	D, X	form end of block
1167 FDDB 34		11	PSHS	D, X, U	put start/end on stack
1168 FDDD C6 1169 FDDF 8D	31 1C	2 7	LDB BSR	#1 XHEAD	put a data header
1170 FDE1 30	62	5	LEAX	2, S	point to start/end
1171 FDE3 C6	04	2	LDB	#4	two words
:172 FDE5 8D 1173 FDE7 35	26 . 56	7 11	PULS	DATOUT D, X, U	and put start/end out get all back
1174 FDE9 5C		2	INCB	5,,0	modify length
1175 FDEA 8D	21	7 7	BSR	DATOUT	and send data bytes
1176 FDEC 80 1177 FDEE 20	29 08	3	BSR BRA	CHKOUT SAV6	then checksum and repeat
1178		*			
1179 FDF0 17	FA8C 06	9 SAV2 3	LBSR BEQ	CONCHR SAV3	get input on er send eof block
1180 FDF3 27 1181 FDF5 81	2D	2	CMPA	\$/-	if eof inhibit.
1182 FDF7 27	10	3	BEQ	RTSI	., then skip ×9
1183 FDF9 97	A2	4 2 SAU3	STA LDB	LASTC #19	re-use input send cof.
1184 FDFB C6 1185	39	*	LDD	* 2	3214 6017
1186		*			header to block, header type in b.
1187 1188		*		also initialise	s checksum in y,
1189 FDFD 108E	0000	4 XHEAD	LDY	#0	
1190 FE01 31	3F	5 XH1	LEAY	-1, Y	1000.
1191 FE03 26 1192 FE05 86	FC D8	3 2	BNE LDA	XH1 #'X+\$80	, delay send an × to, .
1193 FE07 80	13	7	BSR	CASOPI	,, cassette
1194 FE09 1F	98	6	TFR	8, A	get type
1195 FE0B 20 1196	ØF	3	BRA	CASOPI	and send
1197		*			tine, sends b data bytes starting from $ imes$
1198		*		(b zero means 2	56 bytes), x moves up by b bytes
1199 1200 FE0D A6	80	6 DATOUT	LDA	, X+	get data
1231 FEOF 31	A6	5	LEAY	A, Y	modify checksum
1202 FE11 80	8 9	7	BSR	CASOPI	send data
1203 FE13 5A 1204 FE14 26	F7	2 3	dece Bne	DATOUT	repeat until zero count
1205 FE16 39	••	5 RTS1	RTS		
1206		*			. Aller Color of Taller Sucha and of
1207 1208		*		send checksum t	o tape from y, lower byte only
1209 FE17 1F	20	6 CHKOUT		Υ, D	checksum to d
1210 FE19 1F	98	é	TFR COMA	B, A	then get low byte want result \$FF
1211 FE1B 43 1212 FE1C 6E	9F Ø369	2 8 CASOPI		E CASOPA3	send check byte
1213		*			
1214 1215		*		drive commands	to perform a boot operation
1216				drive parameter	specification
1217		*			
1218 FE20	35040006	DISCIT	FCB	\$35, 4, \$0D, \$14,\$	05, \$AA for Shugart drive
1219	18E5	*			
1220		*		drive bad track	S
1221	350410FF	*	FCB	475 A DOTIS /498	+8+\$10, \$FF, \$FF, \$FF
1222 FE26	FFFF		FCB	400, 4, DKIVE/ 400	
1223		*			
1224 1225		*		mode register s	etup
1226 FE2C	76021701		FCB		
1227				\$3A-DRIVE, 2, \$17	, \$C1
		*			
1228		*	100		,\$Cl disc, starts motor
1228 1229 1230 FE30	78822368	*	FCB		disc, starts motor
1228 1229 1230 FE30 1231		*		load head onto \$3A+DRIVE, 2, \$23	disc, starts motor ,\$28+DRIVE
1228 1229 1230 FE30 1231 1232		*		load head onto	disc, starts motor ,\$28+DRIVE
1228 1229 1230 FE30 1231 1232 1233 1234 FE34		* * *		load head onto \$3A+DRIVE, 2, \$23	disc, starts motor ,\$28+DRIVE
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235	7 n0 22368	* * *	FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0	disc, starts motor ,\$28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237	7 10 22368 60 90	* * *	FCB FCB	load head onto #3A+DRIVE,2,#23 query drive rea #2C+DRIVE,0 seek to track 0	disc, starts motor ,#28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36	7 n0 22368	* * * *	FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0	disc, starts motor ,#28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1234 FE34 1235 1236 1237 1238 FE36 1239	7 10 22368 60 90	* * *	FCB FCB	load head onto #3A+DRIVE,2,#23 query drive rea #2C+DRIVE,0 seek to track 0	disc, starts motor ,\$28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1237 1238 FE36 1239 1240 1241	7 10 22368 6C 00 690100	* * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 seek to track 0 \$29+DRIVE, 1, \$00 query drive rea	disc, starts motor ,\$28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1239 1240 1241 1241 1242 FE39	7 10 22368 60 90	*	FCB FCB	load head onto #3A+DRIVE, 2, #23 guery drive rea #2C+DRIVE, 0 seek to track 0 #29+DRIVE, 1, #00	disc, starts motor ,\$28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1237 1238 FE36 1239 1240 1241	7 10 22368 6C 00 690100	*	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 seek to track 0 \$29+DRIVE, 1, \$00 query drive rea	disc, starts motor ,\$28+DRIVE dy
1228 1229 1230 FE30 1231 1232 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1238 FE36 1238 FE36 1239 1240 1241 1242 FE39 1243 1244 1245	74022368 6C00 690100 6C00	* * * * * * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 \$eek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0 read sector 2	disc, starts motor ,\$28+DRIVE dy dy
1228 1229 1230 FE30 1231 1232 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1242 FE39 1240 1241 1242	74022368 6000 690100 6000 53030002	* * * * * * * * * * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 seek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0	disc, starts motor ,\$28+DRIVE dy dy
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1237 1238 FE36 1239 1240 1241 1242 FE39 1243 1244 1245 1246 FE38	74022368 6C00 690100 6C00	* * * * * * * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 \$eek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0 read sector 2 \$13+DRIVE, 3, \$00	disc, starts motor ,\$28+DRIVE dy dy ,\$82,\$21
1228 1229 1230 FE30 1231 1232 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1242 FE39 1240 1241 1244 1245 1246 FE38 1247 1248	74022368 6000 690100 6000 53030002	* * * * * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 \$eek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0 read sector 2	disc, starts motor ,\$28+DRIVE dy dy ,\$82,\$21
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1242 FE39 1244 1245 1244 1245 1244 1245 1244 1245 1244 1245 1244 1245 1244 1245 1246 1247	74022368 6000 690100 6000 53030002	* * * * * * * * * * * *	FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 \$eek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0 read sector 2 \$13+DRIVE, 3, \$00	disc, starts motor ,\$28+DRIUE dy dy ,\$82,\$21 t sector 3
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1241 1242 FE39 1240 1243 1244 1245 1246 FE38 1247 1249 1249 1249 1249 1249 1249	74022368 6C00 690100 6C00 53630002 21	* * * * * * * * * * * * *	FCB FCB FCB FCB	load head onto \$3A+DRIVE, 2, \$23 query drive rea \$2C+DRIVE, 0 \$eek to track 0 \$29+DRIVE, 1, \$00 query drive rea \$2C+DRIVE, 0 read sector 2 \$13+ORIVE, 3, \$00 read starting a \$13+DRIVE, 2, \$00	disc, starts motor ,\$28+DRIUE dy dy ,\$62,\$21 t sector 3 ,\$63
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1242 FE39 1244 1244 1244 1244 1244 1245 FE38 1244 1246 FE38 1247 1246 FE38	74022368 6C00 690100 6C00 53630002 21	* * * * * * * * * * * *	FCB FCB FCB FCB	load head onto \$3A+DRIUE, 2, \$23 query drive rea \$2C+DRIUE, 0 seek to track 0 \$29+DRIUE, 1, \$00 query drive rea \$2C+DRIUE, 0 read sector 2 \$13+DRIUE, 3, \$00 read starting a \$13+DRIUE, 2, \$00 this routine bo	disc, starts motor ,\$28+DRIUE dy dy ,\$82,\$21 t sector 3
1228 1229 1230 FE30 1231 1232 1233 1234 FE34 1235 1236 1237 1238 FE36 1239 1240 1241 1241 1242 FE39 1240 1243 1244 1245 1246 FE38 1247 1249 1249 1249 1249 1249 1249	74022368 6C00 690100 6C00 53630002 21	* * * * * * * * * * * * *	FCB FCB FCB FCB	load head onto \$3A+DRIUE, 2, \$23 query drive rea \$2C+DRIUE, 0 seek to track 0 \$29+DRIUE, 1, \$00 query drive rea \$2C+DRIUE, 0 read sector 2 \$13+DRIUE, 3, \$00 read starting a \$13+DRIUE, 2, \$00 this routine bo	disc, starts motor ,\$28+DRIUE dy dy ,\$62,\$21 t sector 3 ,\$63 otstraps from a mini-floppy disc

1254					*			
	FE44		FFØ4		800T	LDX	#ANRTI	set dummy, .
	FE47		83	5		STX	INMI #FLOPY/256	interrupt routine
	FE49		0A	2		LDA TFR	HELDETZ206	set direct page to floppy controller
	FE4B	11-	98 000a	Ð		SETDP	FL0PY/256	and tell assembler
1259	FE4D	8F	FE20	з		LDX	#DISCIT	point to tables
	FE50		4D	7		BSR	CHOPAR	drive parameters
	FE52		4B	7		BSR	CMOPAR	bad tracks
	FE54		49	7		BSR	CHOPAR	mode register
	FE56		47	7		BSR	CMOPAR	drive on
	FE38		5C	7		BSR	DRURDY	check ready
	FE5A		43	7		BSR	CMOPAR	seek track 0
	FE5C		58	7		BSR	DRURDY	check ready
	FESE		3F	?		BSR	CHOPAR	read sector 2
	FE60		0000	3		LDU	#0	put at 0
	FE63		32	6		TFR	U, Y TRNSFR	and point to it move disc to memory
	FE65		62	73		BSR	DERR	NON-Zero memory Non-Zero means error
	FE67 FE69		2A FF42	3		LDD	##FF42	error \$FF in case
	FE6C			10		CMPD	, Y++	if not \$FF42.
	FE6F		22	Ĩž		BNE	DERR	, then error, no boot present
	FE71		20	7		BSR	CMDPAR	start read at sector 3
	FE73		61	8		LDU	++	get address to put at
1278	FE75	A6	AØ	6		LDA	, Y+	and number of sectors
	FE77			e		LDY	, Y	start of program
	FE7A		20	2		ADDA	#\$20	add sector length value
	FE7C		90		BOOT 1	LDB	FDCS	get fdc status and.
	FE7E		20	2		BITB	#\$20 BOUT1	if parameter register full then wait
	FE80		FA	3 4		BNE	FOCP	send number of sectors
	FE82 FE84		01 43	7		BSR	TRNSFR	move data to memory
	FE86		43 06	ંટ		BNE	DERR	error if non-zero
	FE88		FCE1	9		LBSR	NUMB	try get number
	FE8B		38	รั		BUC	RTS2	got one, stay in monitor
	FE8D			7		LDS	STACK	reset stack and.
	FE91		A4	3		JMP	,Υ	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
	FE9C	16	FA4A	5		LBRA	OPCRLF	newline and exit
1295					*			
1296					*			nds 1 command followed by a variable
1297					*			eters, possibly none mand, next byte is number of parameters
1298 1299					*			after last parameter, destroys d
1300					*		× reit porner ig	erter last perameter, destroys a
1301	FE9F	EC	81	8	CMOPAR	L.DD	, X++	get command and number
1382	FEA1	ØD	00		CP1	TST	FDCS	test status and
	FEA3		FC	3		BMI	CP1	wait if busy
	FEA5		88	4		STA	FDCC	send command
1305	FEA7	58			CP2	DECB		if no more parameters,
1306	FEA8	2B	1E	3		BMI	RTS2	. then exit
13 06 1307	FEA8 FEAA	2B 96	00	3 4	CP2 CP4	BMI LDA	FDCS	., then exit if parameter,,
13 06 1307 1308	FEA8 FEAA FEAC	28 96 85	00 20	3 4 2		BMI LDA BITA	FDCS #\$20	., then exit if parameter ,.register full
1306 1307 1308 1309	FEA8 FEAA FEAC FEAE	28 96 85 26	00 20 FA	3423		BMI LDA BITA BNE	FDCS #\$20 CP4	,,then exit if parameter,, ,register full., ,,then wait
1306 1307 1308 1309 1310	FEA8 FEAA FEAC FEAE FEB0	28 96 85 26 A6	00 20 FA 80	34236		BMI LDA BITA BNE LDA	FDCS #\$20 CP4 , X+	then exit if parameter. register full. then wait get parameter and
1306 1307 1308 1309 1310 1311	FEA8 FEAA FEAC FEAE FEB0 FEB2	2B 96 85 26 A6 97	00 20 FA 80 01	34N364		BMI LDA BITA BNE LDA STA	FDCS #\$20 CP4 , X+ FDCP	.,then exit if parameter .,register full .,then wait get parameter and .,send it
1306 1307 1308 1309 1310 1311 1312	FEA8 FEAA FEAC FEAE FEB0	2B 96 85 26 A6 97	00 20 FA 80	34236		BMI LDA BITA BNE LDA	FDCS #\$20 CP4 , X+	then exit if parameter. register full. then wait get parameter and
1306 1307 1308 1309 1310 1311 1312 1313	FEA8 FEAA FEAC FEAE FEB0 FEB2	2B 96 85 26 A6 97	00 20 FA 80 01	34N364		BMI LDA BITA BNE LDA STA	FDCS #\$20 CP4 , X+ FDCP CP2	. then exit if parameter. register full. then wait get parameter and. send it then repeat
1306 1307 1308 1309 1310 1311 1312	FEA8 FEAA FEAC FEAE FEB0 FEB2	2B 96 85 26 A6 97	00 20 FA 80 01	34N364	CP4 *	BMI LDA BITA BNE LDA STA	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r	.,then exit if parameter .,register full. .,then wait get parameter and. .,send it
1306 1307 1308 1309 1310 1311 1312 1313 1314	FEA8 FEAA FEAC FEAE FEB0 FEB2	2B 96 85 26 A6 97	00 20 FA 80 01	34N364	CP4 * *	BMI LDA BITA BNE LDA STA	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317	FEA8 FEAC FEAC FEB0 FEB2 FEB4	28 96 85 26 A6 97 20	00 20 FA 80 01 F1	3423643	CP4 * * *	BMI LDA BITA BNE LDA STA BRA	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command	. then exit if parameter. . register full. . then wait get parameter and. . send it t. . send it t. eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318	FEA8 FEAA FEAC FEB0 FEB2 FEB4 FEB6	2B 96 85 26 97 20 1F	00 20 FA 80 01 F1	3423643	CP4 * * * * * DRVRDY	BMI LDA BITA BNE LDA STA BRA	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U	. then exit if parameter. register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319	FEAS FEAA FEAC FEB2 FEB4 FEB4 FEB6 FEB8	2B 96 85 26 97 20 1F 1F	00 20 FA 80 01 F1 31	3423643 66	CP4 * * *	BMI LDA BITA BNE LDA STA BRA TFR	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320	FEA8 FEAA FEAC FEB9 FEB4 FEB4 FEB8 FEB8 FEB8	2B 96 85 26 97 20 1F 1F 8D	00 20 FA 80 01 F1 31 E3	3423643 667	CP4 * * * DRURDY DR2	BMI LDA BITA BNE LDA STA BRA TFR FRR BSR	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command X, U U, X U, X CMDPAR	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321	FEAS FEAC FEAC FEBO FEBO FEBO FEBO FEBO FEBO FEBO	2B 96 85 26 97 20 1F 1F 8D 96	00 20 FA 80 01 F1 31 E3 00	3423643 6674	CP4 * * * * * DRVRDY	BMI LDA BITA BNE LDA STA BRA TFR TFR SSR LDA	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X, U U, X CMDPAR FDCS	. then exit if parameter. . register fill. . then wait get parameter and. . send it then repeat eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until.
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321	FEAS FEAC FEBO FEBO FEBO FEBO FEBO FEBO FEBO FEBO	2B 96 85 26 97 20 1F 1F 8D 96 85	00 20 FA 80 01 F1 E3 E3 00 10	3423643 66742	CP4 * * * DRURDY DR2	BMI LDA BITA BITA BNE LDA BRA TFR TFR BSR LDA BITA	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command to next command X,U U,X CMDPAR FDCS ##10	. then exit if parameter. . registar full. . then wait get parameter and. . send it then repeat eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result.
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323	FEAS FEAA FEAC FEB2 FEB3 FEB4 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 96 856 66 97 20 1F 1F 80 65 27	00 20 FA 80 01 F1 31 E3 00 10 FA	3423643 667423	CP4 * * * DRURDY DR2	BMI LDA BITA BNE LDA STA BRA TFR BSR LDA BITA BEQ	FDCS #\$20 FDCP CP2 tstif drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result. . result.
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323	FEA8 FEAA FEAC FEB2 FEB3 FEB4 FEB8 FEB8 FEB8 FEB8 FEB2 FEC2	2B 96 85 26 97 20 1F 1F 8D 96 57 96 296	00 20 FA 80 01 F1 31 E3 60 10 FA 01	3423643 6674234	CP4 * * * DRURDY DR2	BMI LDA BITA BNE LDA STA BRA TFR FSR LDA BITA BEQ LDA	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command to next command X,U U,X CHDPAR FDCS ##10 DR1 FDCR	. then exit if parameter. . registar full. . then wait get parameter and. . send it then repeat eady. on entry x points to read drive sayuence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . ready get result
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324	FEAS FEAA FEAA FEB2 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 1FFD 857 985 1FFD 857 985	00 20 FA 80 01 F1 31 E3 00 10 FA	34N3643 6674N34N	CP4 * * * DRURDY DR2	BMI LDA BITA BITA BNE LDA STA BRA TFR BSRA BITA BEQ LDA BITA	FDCS #\$20 FDCP CP2 tstif drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result. . result . ready bit mask
1306 1307 1308 1309 1310 1311 1312 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326	FEAA FEAAE FEB0 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * DRURDY DR2 DR1	BMI LDA BITA BNE LDA STA BRA TFR FSR LDA BITA BEQ LDA	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command: to next command: X,U U,X CMDPAR FDCS ##10 DR1 FDCS ##10 DR1 FDCR FDCP FDCS	. then exit if parameter. . registar full. . then wait get parameter and. . send it then repeat eady. on entry x points to read drive sayuence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . ready get result
1306 1307 1308 1309 1310 1311 1312 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326	FEAS FEAA FEAA FEB2 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * * DRURDY DR2	BMI LDA BITA BITA BITA BITA BRA TFR BSR LDA BITA BEQ LDA BITA BEQ	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command: to next command: X,U U,X CMDPAR FDCS ##10 DR1 FDCS ##10 DR1 FDCR FDCP FDCS	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result. . result . ready bit mask
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326	FEAA FEAAE FEB0 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * DRURDY DR1 DR1 RTS2	BMI LDA BITA BITA BITA BITA BRA TFR BSR LDA BITA BEQ LDA BITA BEQ	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command : status command : x,U U,X CMDFAR FDCS ##10 DR1 FDCR #DRIVE/128*60+4 DR2	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result. . result . ready bit mask
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329	FEAA FEAAE FEB0 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * DRURDY DR2 DR1 RTS2 * *	BMI LDA BITA BITA BITA BITA BRA TFR BSR LDA BITA BEQ LDA BITA BEQ	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr sta-ting at add	then exit then exit register fill. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer ask for drive status wait until ready get result. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1315 1316 1317 1318 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330	FEAA FEAAE FEB0 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * DRURDY DR2 DR1 RTS2 * *	BMI LDA BITA BITA BITA BITA BRA TFR BSR LDA BITA BEQ LDA BITA BEQ	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr sta-ting at add	. then exit if parameter. . registar full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . ready get result ready bit mask not ready, wait ansfers data from disc to memory
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1317 1328 1321 1322 1323 1324 1325 1326 1327 1328 1329 1320 1320 1332	FEAA FEAAE FEB0 FEB2 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3 FEB3	2B 965 856 97 20 1FF 896 57 985 795 295 295 295 295 295 295 295 295 295 2	00 20 FA 80 01 F1 13 31 E3 60 10 FA 01 04	3423643 667423423	CP4 * * DRURDY DR2 DR1 RTS2 * * *	BMI LDA BITA BITA BITA BITA BRA TFR BSR LDA BITA BEQ LDA BITA BEQ	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr sta-ting at add	then exit then exit register fill. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer ask for drive status wait until ready get result. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in
1306 1307 1308 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1322 1323 1324 1322 1323 1324 1322 1322 1324 1322 1322 1324 1322 1322 1324 1322 1322 1324 1322 1324 1322 1324 1322 1324 1322 1324 1325 1326 1327 1328 1326 1326 1327 1328 1329 1331 1332	FEAG FEAC FEAC FEBC FEBC FEBC FEBC FEBC FEC0 FEC0 FEC0 FEC0	28 96 85 26 97 20 1F F 8D 96 87 99 85 29 39 39 39 39 39 39 39 39 39 39 39 39 39	00 2FA 80 01 F1 31 31 31 30 10 61 04 F0	3423643 6674234235	CP4 * * * * DRURDY DR2 DR1 RTS2 * * *	BMI LDA BITA BITA BITA BRA STA BRA TFR TFR BRA BITA BEQ LDA BEQ LDA BEQ RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS ##10 DR1 FDCR #DRIUE/128*60+4 DR2 this routine tr sta-ting at add 'a' when transfi	. then exit if parameter. . register full. . then wait get parameter and. . send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. . result. . ready get result ready bit mask not ready, wait ensfers data from disc to memory ress in u. returns completion code in er finished or error occurs
1306 1307 1308 1308 1308 1310 1311 1312 1313 1314 1315 1316 1317 1321 1321 1323 1324 1323 1324 1325 1326 1327 1328 1326 1327 1328 1328 1329 1331 1332 1333 1334	FEAS FEAC FEAC FEBE FEBS FEBS FEBS FEBS FEBS FECS FECS FECS	28 6 85 26 6 97 20 1F F 80 9 85 27 9 34	00 20 20 70 00 F1 F1 F1 F1 F1 F1 F0 F0 F0 F0	3423643 6674234235 6	CP4 * * DRURDY DR2 DR1 RTS2 * * *	BMI LDA BITA BITA BITA BRA STA BRA STA BRA BERA BITA BEDA BITA BECA RTS PSHS	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr sta-ting at add	then exit then exit register full then wait get parameter and. send it then repeat eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while.
1306 1307 1309 1309 1309 1309 1310 1310 1310 1310	FEAS FEACE FEACE FEBE FEBE FEBE FEBE FEBE FEES FEES FECO FECO FECO FECO	28 685 266 997 20 1F F 8D 965 27 965 27 96 52 79 34 16	00 FA 80 F1 F1 F1 F1 F1 F1 F6 01 F6 F0 01 50	7427647 667427427 5 67	CP4 * * * * DRURDY DR2 DR1 RTS2 * * *	DMI LDA BITA BITA BITA BARA STA BRA TFR TFR BSR BITA BEQ LDA BITA BEQ RTS SEIF	FDCS #\$20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command X,U U,X CMDPAR FDCS #\$10 DR1 FDCR #DR10E/128+60+4 DR2 this routine tr sta-ting at add 'a' when transfi	then exit then exit register full then wait get parameter and send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until ready get result. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI
1306 1307 1309 1309 1309 1310 1310 1310 1311 1312 1313 1314 1315 1316 1317 1318 1317 1320 1321 1322 1324 1322 1324 1325 1326 1327 1328 1328 1328 1328 1328 1328 1328 1328	FEAS FEACE FEBE FEBE FEBE FEBE FEBE FECE FECE FE	28 6 85 26 6 97 20 1F F 80 6 95 27 9 85 27 9 34 A 6 6	00 90 FA 90 FA 90 FA 90 FA 50 64 F0 01 04 F0 01 04	7427647 6674274275 672	CP4 * * * * DRURDY DR2 DR1 RTS2 * * *	BMT LDA BITA BITA BITA BITA BRA STA BRA STA BRA BITA BEQ BITA BEQ RTS SEIF LDB	FDCS ##20 CP4 ,X+ FDCP CP2 testif drive r status command x,U U,X U,X U,X CMDPAR FDCS ##10 DR1 FDCR #DR1UE/128*60+4 DR2 this routine tr stating at add 'a' when transfi	then exit then exit register full then wait get parameter and. send it then repeat eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while.
1306 1307 1308 1309 1309 1310 1310 1310 1310 1312 1313 1314 1315 1316 1317 1318 1317 1318 1317 1328 1320 1321 1322 1322 1322 1323 1324 1325 1326 1326 1327 1328 1328 1328 1328 1328 1328 1328 1328	FEA8 FEAACE FEACE FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEC9 FEC9 FEC9 FEC9	28 6 8 5 2 6 6 9 7 2 0 1 F 1 F B 9 6 8 7 7 9 8 7 9 8 5 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7	00 20 FA 80 F1 F1 31 31 31 31 31 31 60 F0 F0 61 64 F0 61 64 F0	3423643 6674234235 6323	CP4 * * DRURDY DR1 DR1 RTS2 * * * * * *	DMT LDA BITA BITA BITA BAL STA BRA TFR TFR BSR BITA BEQ LDA BEQ LDA BEQ LDA BEQ LDA BEQ LDA BEQ LDA RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command : to next command : to next command X,U U,X CMDPAR FDCS ##10 DR1 FDCR #BRIUE/128*60+4 DR2 this routine tr sta^ting at add 'a' when transfi CC ##04 TRN2	then exit then exit register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer ask for drive status wait until ready get result. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask
1306 1307 1308 1309 1309 1309 1309 1307 1307 1307 1316 1316 1317 1316 1317 1316 1317 1318 1317 1318 1320 1321 1322 1324 1323 1324 1323 1324 1323 1324 1323 1324 1323 1324 1323 1324 1325 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1326 1327 1327 1327 1327 1327 1327 1327 1327	FEA8 FEAACE FEAACE FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEC8 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9	28 6 5 2 6 6 9 7 2 0 1 F F 8 9 6 5 2 7 9 6 5 7 9 7 2 0 1 F F 8 9 6 5 7 9 6 5 7 9 8 5 7 9 9 7 4 A 6 6 8 9 6	00 60 FA 80 FA 80 FA 80 FA 80 FA 80 FA 80 60 FA 80 60 FA 80 60 FA 80 60 FA 80 FA 70 F 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	3423643 6674234235 632234	CP4 * * * * DRURDY DR2 DR1 RTS2 * * *	BMI LDA BITA BITA BITA BRE LDA BEA BEA BITA BEA RTS PSHS SEIF LDB BEA RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command x,U U,X CMDPAR FDCS ##10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr stating at add 'a' when transfi CC ##04 TRN2 FDCD	then exit then exit then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data
1306 1307 1208 1309 1310 1310 1310 1312 1314 1312 1313 1314 1315 1316 1317 1318 1317 1318 1317 1328 1320 1321 1322 1323 1326 1327 1328 1326 1327 1328 1332 1334 1335 1336 1337 1338 1339 1339 1339 1339 1339 1339 1339	FEAS FEAS FEACE FEBO FEBO FEBO FEBO FEBO FECO FECO FECO FECO FECO	28655669720 IF IF 89657765579 34 A 66 86 67	00 20 FA 80 F1 F1 31 31 31 31 31 31 50 01 F0 01 50 04 F0 01 04 F0	? 4 2 ? 6 4 ? 6 6 7 4 2 7 4 2 ? 5 6 7 2 7 4 6	CP4 * * DRURDY DR1 DR1 RTS2 * * * TRNSFR TRNSFR	DMT LDA BITA BITA BITA BITA BRA TFR BSRA STA BEQ LDA BEQ LDA BEQ LDA BEQ LDA BEQ LDA STA SEIF LDB STA STA	FDCS #220 CP4 ,X+ FDCP CP2 test if drive r status command: to next command: X,U U,X CMDPAR FDCS #\$10 DR1 FDCS #\$10 DR1 FDCR #\$10 DR1 FDCR #\$10 DR1 FDCR #\$10 CC #\$94 TRN2 FDCD CC U	then exit then exit register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer restore pointer ask for drive status wait until. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data. and store it
1306 1307 1307 1308 1309 1310 1310 1310 1310 1310 1310 1310	FEA6 FEA6 FEA7 FEB0 FEB0 FEB0 FEB0 FEB0 FEB0 FEC0 FEC0 FEC0 FEC0 FEC0 FEC0 FEC0 FEC	28695266970 1FF896527985279 31A66896730 31A66896730	00 FA 80 F1 F1 31 31 31 31 31 31 70 F0 F0 F0 64 F0 C0	? 4 2 ? 6 4 ? 6 6 7 4 2 7 4 2 ? 5 6 7 2 7 4 6	CP4 * * DRURDY DR1 DR1 RTS2 * * * * * *	BMI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA STA BEQ RTS SEIF LDA SEIF LDA STA SEIF LDA STA SEIF LDA STA STA STA SEIF	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command x,U U,X CMDPAR FDCS ##10 DR1 FDCS #DR1UE/128+60+4 DR2 this routine tr stating at add 'a' when transfi CC ##804 TRN2 FDCD UU-	then exit then exit then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. result. ready get result ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data. and store it wait for interrupt
1306 1307 1308 1309 1310 1310 1310 1310 1310 1310 1310	FEA8 FEAACFEAC FEAACFEAC FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9 FEC9	28 9 85 2 66 97 20 F F 8 0 96 5 27 96 5 27 99 34 A 6 0 96 7 30 5 2 7 98 5 27 99 5 27 9	00 FA 80 FA 70 C 70 C 70 C 70 C 70 C 70 C 70 C 70	ਲ਼ 4 2ਲ਼64ਲ਼ 66742ਲ਼42ਲ਼5 20	CP4 * * DRURDY DR1 DR1 RTS2 * * * TRNSFR TRNSFR	DMT LDA BITA BITA BITA BITA BRA TFR BSRA STA BEQ LDA BEQ LDA BEQ LDA BEQ LDA BEQ LDA STA SEIF LDB STA STA	FDCS #220 CP4 ,X+ FDCP CP2 test if drive r status command: to next command: X,U U,X CMDPAR FDCS #\$10 DR1 FDCS #\$10 DR1 FDCS #\$10 DR1 FDCR #\$10 DR1 FDCR #\$10 CC #\$94 TRN2 FDCD CC U	then exit then exit register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer restore pointer ask for drive status wait until. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data. and store it
1306 1307 1308 1307 1308 1310 1311 1312 1312 1314 1315 1314 1315 1314 1315 1314 1315 1327 1328 1321 1322 1323 1324 1335 1326 1339 1334 1334 1341 1341 1341	FEA8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB	28 6 5 5 6 4 9 7 2 6 5 5 7 9 5	00 FA 80 F1 F1 31 31 31 31 31 31 76 60 F6 F0 61 64 64 64 64 F0 F0 F0	3423643 6674234235 63234 69 4 294	CP4 * * DRURDY DR1 DR1 RTS2 * * * TRNSFR TRNSFR	BMI LDA BITA BITA BITA BITA BITA BRA STA BRA STA BRA BITA BEQ BITA BEQ BITA BEQ RTS SEIF CHAS SEIF CHAS STA BITB	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command X,U U,X CHDPAR FDCS ##10 DR1 FDCS ##10 DR1 FDCS ##10 DR1 FDCR eDR1UE/128*60+4 DR2 this routine tr stating at add 'a' when transf CC ##04 TRN2 FDCD ,U+ FDCD ,U+ FDCD ,U+ FDCD S	then exit then matt tprogrammeter then usit get parameter and send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer restore pointer ask for drive status wait until ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data and store it wait for interrupt check data
1306 1307 1308 1307 1310 1310 1310 1310 1310 1310 1310	FEA8 FEAACFFEB0 FEB2 FEB4 FEB2 FEB4 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEC2 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 80 F1 13 13 13 13 13 13 13 13 13 13 13 FA 91 84 F0 61 84 F0 F6 F6 F6	∽ 4 2 3 6 4 3 6 6 6 7 4 2 3 4 2 3 5 6 7 8 2 3 4 6 2 4 3 6 4 2 6 4 3 7 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 6 5 6	CP4 * * DRURDY DR1 DR1 RTS2 * * * TRNSFR TRNSFR	DHI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA BEQ RTS PSHS SEIF LDB BEQ RTS STA DRA LDA BEQ RTS DA DHA BEQ RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command x,U U,X CMDPAR FDCS ##10 DR1 FDCR #DR1VE/128*60+4 DR2 this routine tr stating at add 'a' when transf CC ##64 FDCD trN2 FDCD trN2 FDCD TRN2 FDCD trN5 FDCD TRN2 FDCD TRN1 FDCD TRN1 FDCD TRN1 FDCD TRN1	then exit then wait then wait get parameter and send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until ready get result. ready get result ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data and store it wait for interrupt check data and store it and
1306 1307 1308 1307 1308 1310 1311 1312 1314 1315 1314 1315 1314 1315 1314 1316 1317 1328 1329 1320 1327 1328 1327 1328 1327 1328 1327 1328 1329 1330 1329 1330 1334 1334 1341 1342 1341	FEA8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB8 FEB	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 90 FA 90 F1 71 73 13 13 13 13 13 13 70 FA 90 FA 70 FA 90 FA 90 FA 70 FA 90 FA 90 FA 90 FA 90 FA 70 FA 70 FA 70 FA 70 FA 90 FA 70 FA 90 FA 70 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	∽+N∿6+⊼ 667+N⊁+N∩5 67N2+80 8697+N75875	CP4 * * DRURDY DR1 DR1 RTS2 * * * TRNSFR TRNSFR	BMI LDA BITA BITA BITA BITA BRA STA BRA STA BRA BITA BEQ BITA BEQ RTS SEIF RTS SEIF BEQ RTS SEIF BRA STA BEDA BITA BEQ RTS	FDCS ##20 CP4 ,X+ FDCP CP2 testif drive r status command x,U U,X U,X U,X CMDPAR FDCS ##10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr sta-ting at add 'a' when transf CC ##04 TRN2 FDCS FDCS TRN1 CC	<pre>then exit if parameter register full then wait get parameter and send it then repeat eady. on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until result. result. result. eady bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data. avat store it wait for interrupt check data available get interrupt masks back</pre>
1306 1307 1308 1307 1310 1310 1311 1312 1313 1314 1315 1314 1315 1314 1315 1314 1315 1314 1325 1326 1327 1328 1324 1321 1325 1326 1327 1328 1329 1329 1329 1329 1329 1329 1329 1329	FEA8 FEAACFFEB0 FEB2 FEB4 FEB2 FEB4 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEC2 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 90 FA 90 F1 71 73 13 13 13 13 13 13 70 FA 90 FA 70 FA 90 FA 90 FA 70 FA 90 FA 90 FA 90 FA 90 FA 70 FA 70 FA 70 FA 70 FA 90 FA 70 FA 90 FA 70 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	∽ 4 2 3 6 4 3 6 6 6 7 4 2 3 4 2 3 5 6 7 8 2 3 4 6 2 4 3 6 4 2 6 4 3 7 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 6 5 6	CP4 * * * DRURDY DR1 RTS2 * * TRNSFR TRN1 TRN2 *	DHI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA BEQ RTS PSHS SEIF LDB BEQ RTS STA DRA LDA BEQ RTS DA DHA BEQ RTS	FDCS #20 CP4 ,X+ FDCP CP2 test if drive r status command: to next command: X,U U,X CMDPAR FDCS #\$10 DR1 FDCS #\$10 DR1 FDCR #\$10 DR1 FDCR #\$10 DR1 FDCR #\$10 CC #\$94 TRN2 FDCD ,U+ #\$FF FDCS TRN1 CC FDCR	then exit then exit register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points seve for drive status wait until. ready get result. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs seve CC while. set masks, disc is on NMI data available mask get data. and store it wait for interrupt check data. available get interrupt masks back then get result and return Z bit accordingly
1306 1307 1308 1309 1310 1310 1310 1310 1310 1310 1310	FEA8 FEAACFFEB0 FEB2 FEB4 FEB2 FEB4 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEC2 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 90 FA 90 F1 71 73 13 13 13 13 13 13 70 FA 90 FA 70 FA 90 FA 90 FA 70 FA 90 FA 90 FA 90 FA 90 FA 70 FA 70 FA 70 FA 70 FA 90 FA 70 FA 90 FA 70 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	∽ 4 2 3 6 4 3 6 6 6 7 4 2 3 4 2 3 5 6 7 8 2 3 4 6 2 4 3 6 4 2 6 4 3 7 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 4 6 4 5 6 5 6	CP4 * * * DRURDY DR2 DR1 RTS2 * * TRNSFR TRN1 TRN2 * * *	DHI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA BEQ RTS PSHS SEIF LDB BEQ RTS STA DRA LDA BEQ RTS DA DHA BEQ RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command x,U U,X CMDPAR FDCS ##10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr stating at add 'a' when transf CC ##04 TRN2 FDCD UL UL EDCS ##04 TRN2 FDCD UL UL EDCS TRN1 CC EDCS TRN1 CC EDCS TRN1 CC EDCS EDCR	then exit then exit then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence save pointer restore pointer ask for drive status wait until. ready get result ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. set masks, disc is on NMI data available mask get data awd store it wait on therrupt check data awdiable masks back then get result and return Z bit accordingly st handler, comes here on irg active
1306 1307 1308 1307 1310 1310 1311 1312 1313 1314 1315 1314 1315 1314 1315 1314 1315 1314 1316 1317 1328 1327 1328 1327 1328 1327 1338 1328 1329 1329 1329 1329 1329 1329 1329 1329	FEA8 FEAACFFEB0 FEB2 FEB4 FEB2 FEB4 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEC2 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 90 FA 90 F1 71 73 13 13 13 13 13 13 70 FA 90 FA 70 FA 90 FA 90 FA 70 FA 90 FA 90 FA 90 FA 90 FA 70 FA 70 FA 70 FA 70 FA 90 FA 70 FA 90 FA 70 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	∽ 4 2 3 6 4 3 6 6 6 7 4 2 3 4 2 3 5 6 7 8 2 3 4 6 2 4 3 6 4 2 6 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 4 3 6 4 4 3 6 4 4 3 6 4 4 4 3 6 4 4 4 3 6 4 4 4 3 6 4 4 4 4	CP4 * * * DRURDY DR1 RTS2 * * TRNSFR TRN1 TRN2 * * * * * * * * * * * * * * * * * * *	DHI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA BEQ RTS PSHS SEIF LDB BEQ RTS STA DRA LDA BEQ RTS DA DHA BEQ RTS	FDCS #20 CP4 ,X+ FDCP CP2 test if drive r status command X,U U,X CMDPAR FDCS ##10 DR1 FDCS ##10 DR1 FDCR #10 DR1 FDCR this routine tr stating at add 'a' when transf CC ##04 TRN2 FDCD ,U+ #\$FF FDCS TRN1 CC FDCR interrupt reque: checks for time:	<pre>then exit if parameter register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer ask for drive status wait until. ready get result ready get result ansfers data from disc to memory ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. and store it wait for interrupt check data. and store it wail for interrupt check data. and store jt wailable get interrupt masks back then get result and return Z bit accordingly st handler, comes here on irg active r 1 or keyboard interrupt, if neither</pre>
1306 1307 1308 1309 1310 1310 1310 1310 1310 1310 1310	FEA8 FEAACFFEB0 FEB2 FEB4 FEB2 FEB4 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEB2 FEC2 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3 FEC3	28965266920 IFF809652965279 34 A C 20 96 7 30 52 55 65 7 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 57 59 56 56 56 56 56 56 56 56 56 56 56 56 56	00 FA 90 FA 90 F1 71 73 13 13 13 13 13 13 70 FA 90 FA 70 FA 90 FA 90 FA 70 FA 90 FA 90 FA 90 FA 90 FA 70 FA 70 FA 70 FA 70 FA 90 FA 70 FA 90 FA 70 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	∽ 4 2 3 6 4 3 6 6 6 7 4 2 3 4 2 3 5 6 7 8 2 3 4 6 2 4 3 6 4 2 6 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 3 6 4 4 4 3 6 4 4 3 6 4 4 3 6 4 4 4 3 6 4 4 4 3 6 4 4 4 3 6 4 4 4 4	CP4 * * * DRURDY DR2 DR1 RTS2 * * TRNSFR TRN1 TRN2 * * *	DHI LDA BITA BITA BITA BITA BRA STA BRA STA BRA LDA BEQ RTS PSHS SEIF LDB BEQ RTS STA DRA LDA BEQ RTS DA DHA BEQ RTS	FDCS ##20 CP4 ,X+ FDCP CP2 test if drive r status command x,U U,X CMDPAR FDCS ##10 DR1 FDCR #DR1UE/128+60+4 DR2 this routine tr stating at add 'a' when transf CC ##04 TRN2 FDCD UL UL EDCS ##04 TRN2 FDCD UL UL EDCS TRN1 CC EDCS TRN1 CC EDCS TRN1 CC EDCS EDCR	<pre>then exit if parameter register full. then wait get parameter and. send it then repeat eady, on entry x points to read drive sequence, on exit drive is ready and x points sequence, on exit drive is ready and x points sequence save pointer restore pointer restore pointer ask for drive status wait until. ready get result ready get result ansfers data from disc to memory ready bit mask not ready, wait ansfers data from disc to memory ress in u. returns completion code in er finished or error occurs save CC while. and store it wait for interrupt check data. and store it wail for interrupt check data. and store jt wailable get interrupt masks back then get result and return Z bit accordingly st handler, comes here on irg active r 1 or keyboard interrupt, if neither</pre>

1350	FEEØ	96	03	2		1.00	#MONDP	tak un
	FEE2		88	6	IRQHAN	TFR	A, DP	set up direct page
1353			0003			SETDP	MONDP	tell assembler
	FEE4 FEE7		0980 00	35		ldx Lda	#KUIA KIFR-KUIA, X	point to via address get flag register
	FEE9		16	3		BPL	UNUSED	not the via!
	FEEB		40	2		ANDA	#T1IFLG	try timer 1
	FEED		96	3		BEQ	IROH1	not timer 1
	FEEF FEF1		90 9F 0373	5 8		STA JMP	KIFR-KUIA, X (LIRORTS)	clear the interrupt
1361		UL.	57 0375	Ξ.	*	011	L INGR TOS	
	FEF5		0 D		IRQH1	LDA	KIFR-KVIA, X	get flags again
	FEF7 FEF9		10 0A	23		anda Beq	#CB1FLG UNUSED	try for keyboard query user if not
	FEFB		00	5		STA	KIFR-KUIA, X	clear the interrupt
1366	FEFD	6 6	00	5		LDA	KIRB-KVIA, N	and get a character.
1367	FEFF FF01	84	7F FAFF	29		ANDA LBSR	#\$7F HAUCHR	stripping spare bit put into buffer
	FF04				ANRT1	RTI	TRIVENK	. then leave irg level
1370					•			
1371 1372					*		come here te un	used interrupts are active, so
1373					*			r and stop processor
1374					*		since cannot clo	ear an unknown interrupt
1375	FF05	8F	FFCC	з	* UNUSED	I DX	#IERR	query user
1377	FF08	17	F9E1	9		LBSR	STRING	. on display
	FFØB	20	FE	3		BRA	*	and stop dead!
1379 1380					*			
1381	FFØD		3F283444		CRTCSV	FCB	\$3F, \$28, \$34, \$44	set up table for s version 6845 crt controller
1382			1E02191B			FCB	\$1E, \$02, \$19, \$18	
1383	FF15		03127013		*	FCB	\$03, \$12, \$70, \$1 3	
	FF 19		3F283404		CRTCRU		\$3F, \$28, \$34, \$84	set up table for r version ort controller
1386			1E02191B			FCB	\$1E, \$02, \$19, \$1B	
1387 1388	FF21		00096809		*	FCB	\$00, \$09, \$68, \$ 89	
1389							set up table 1,	copied directly to ram at rtabl
1390			Ø35B		* PTAB1	FD8	ISTACK	initial stack pointer
1391	FF25 FF27		0000		FINDI	FDB	0	trace initialy off
1393	FF29		7F			FCB	BSPACE	character echoet on rubout
	FF2A FF2B		FF 00			FCB FCB	\$FF 0	echo on, line buffer on printer off
	FF2C		0A			FCB	LF	do not send line feeds, use or only to printer
1397	FF2D		00CD		•	FDB	205	initial baud rate for cassette is 300
1398					*			110 baud use 564 (\$0234) 300 baud use 205 (\$00CD)
1399 1400					*			1200 baud use 48 (\$0030)
1401					*	500		
	FF2F FF31		F95B FASE			FDB FDB	DISPLA GETCHR	console output console input
	FF33		FD25			FDB	MCASOP	cassette output
	FF35		FD53			FOB	MCASIN	cassette input
	FF37 FF39		FA9D FF7F			FDB	PRINT FUNCTS	printer routine display function table
	FF3B		FF51			FDB	CHINDS	monitor command table
	FF3D		FF05			FDB	UNUSED	initial timer 1 routine
	FF3F FF41		0000 FF05			FDB FDB	0 UNUSED	no memory interpret reserved vector
	FF43		FF05			FDB	UNUSED	swi3
	FF45		FF05			FDB FDB	UNUSED	swi2 firq
	FF47 FF49		FF05 FEE0			FDB	IRQHHN	1rg
1416	FF4B		FBC3			FDB	SWIHAN	Slu 1
	FF4D		FF05			FDB	UNUSED	nmi initial load offset
1418	FF4F		0000		*	FDB	0	THE VERY LONG OF SEV
1420			FF51		PTAB2	EQU	-	end of table
1421				•	*			
1423					*		this table cont	ains the standard set of commands
1424					*		provided by the	monitor.
1425 1426	FF51		ØE		CMNDS	FCB	CMNDE-CMNDS/3-1	number of entries
1427	FF52		47			FCB	'G	
	FF53 FF55		FC60 40			FDB FCB	GOUSER-CHINDS	go to program
	FF56		FBB0			FDB	MEM-CMNDS	memory examine
	FF58		52			FCB FDB	'R EXREG-CMNDS	examine registers
	FF59 FF5B		F967 50			FCB	P	and and a second se
1434	FF5C		FC68			FDB	RESUME-CMNDS	proceed after break
1435 1436	FF5E		54 FD10			FCB FDB	'T TRACEN-CMNDS	set trace number
	FF61		53			FCB	'S	
	FF62		FE5C 4C			FDB FCB	SAVE-CHNOS	save on cassette
	FF64 FF65		40 FD50			FDB	LOAD-CMNDS	load from cassette
1441	FF67		56			FCB	۰ u	
	FF68 FF6A		FCD2 44			FDB FCB	BRKSET-CMNDS	set break address
			FÉF3			FOB	BOOT-CMMOS	disc bootstrap
	FF6B FF6D		43			FCB	· C	

1446 FF6E	FD29		-DB	PCNTL-CMNDS copy to printer	
1447 FF70	20		CB	SPACE	
1448 FF71	FBØC		DB	ANRTS-CMNDS ignore spaces COMMA	
1449 FF73 1450 FF74	2C FBØC		-CB -DB	ANRTS-CMNDS ignore commas	
1450 FF74 1451 FF76	2E		CB		
1452 FF77	FD16		DB	TRACE-CMNDS do trace operation	
1453 FF79	46		СВ	'F	
1454 FF7A	FD77		DB	LOAD?-CMNDS finish file load	
1455 FF7C	01 FD38		7C8 708	a BADCMD-CMNDS default is query user	
1456 FF7D 1457	FF7F		QŬ	*	
1458		*	-		
1459		*		this table contains the standard functions provi	ded
1460		*		by the vdu control programs.	
1461 1462 FF7F	04	* FUNCTS F	ECB	FUNCTE-FUNCTS/3-1 number of entries	
1463 FF83	00		CB:	CR	
1464 FF81	FA6D		DB .	DOCR-FUNCTS carriage return	
1465 FF83	0A		CB	LF	
1466 FF84	F9F8 7F		08 CB	DOLF-FUNCTS line feed BSPACE	
1467 FF86 1468 FF87	FA06		-DB	DORUB-FUNCTS rubout	
1469 FF89	90		CB	FFEED	
1470 FF8A	F986		DB	CRTCI-FUNCTS form feed	
1471 FF8C	01		CB	1 OTHERE FILIPETE ALCOULT IN ACCENTAL OF	
1472 FF8D 1473	F9EA FF8F	FUNCTE E	08 101	SIMCHR-FUNCTS default is display it *	
1473	FFOF	*			
1475		*		this is the list of strings used by the monitor	
1476		*			
1477 FF8F	526F6D3F		CC CB	ZRom?Z	
1478 FF93 1479 FF96	000A00 43432020	SCRLF F		CR,LF,0 ZCC A B DP X Z	
14/0/11/00	41202042				
	20445020				
	20202058				
1480 FFA7	20 20202059	5	CC 30	Z Y U PC SZ	
1400 FFN/	20202020		00	, , , , , , , , , , , , , , , , , , , ,	
	55202020				
	50432020				
	202053				
1481 FFBA	000A00		CB	CR, LF, 0	
1482 FFBD	000a00 000a	PCMESS F	CB	CR, LF	
1482 FFBD 1483 FFBF	000A00	PCMESS F			
1482 FFBD	000a00 000a 504350 00 57686174	PCMESS FI	CB CC	CR, LF ZPC3Z	
1482 FFBD 1483 FFBF 1484 FFC2 1485 FFC3	000A00 000A 50435D 00 57686174 2069733A	PCMESS FI Fi CQRY FI	CB CC 7CC	CR.LF /PC3/ 6 /What is:/	
1482 FFBD 1483 FFBF 1484 FFC2 1485 FFC3 1486 FFCB	000A00 000A 504350 00 57686174 2069733A 00	PCMESS FI FI CORY FI	св ссв ссв ссв	CR, LF /PCJ/ 0 /What is:/ 0	
1482 FFBD 1483 FF8F 1484 FFC2 1485 FFC3 1486 FFCE 1487 FFCC	000A00 000A 504350 00 57686174 2069733A 00 49	PCMESS FI FI CQRY FI IERR FI	CB CC 7CC	CR.LF /PC3/ 6 /What is:/	
1482 FFBD 1483 FFBF 1484 FFC2 1485 FFC3 1486 FFCB 1487 FFCC 1488 FFCD 1489 FFCE	000A00 000A 504350 00 57686174 2069733A 00	PCMESS FI Fi CQRY FI LERR FI LQRY FI DQRY FI		CR,LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Err /	
1482 FFBD 1483 FFBF 1483 FFC2 1485 FFC3 1486 FFCE 1486 FFCC 1488 FFCD 1489 FFCE 1499 FFCE	000A00 000A 504350 00 57686174 2069733 A 00 49 20	PCMESS FI Fi CQRY FI LQRY FI LQRY FI DQRY FI		CR.LF /PCJ/ 8 /What is:/ 8 /1/ /-/	
1482 FFBD 1483 FF8F 1484 FFC2 1485 FFC3 1486 FFCB 1487 FFCC 1488 FFCD 1489 FFCE 1498 FFD2 1491	000A00 000A 504350 00 57686174 2069733A 00 49 20 45727220	PCMESS FI Fi CQP.Y Fi LCRP.Y Fi LCRP.Y Fi DQRY FI Fi Fi		CR,LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Err / 0	
1482 FFBD 1483 FFBF 1484 FFC2 1485 FFC3 1486 FFCE 1488 FFCD 1488 FFCD 1489 FFCE 1490 FFD2 1491 1492	000A00 000A 504350 00 57686174 2069733A 00 49 20 45727220	PCMESS FI Fi CQRY Fi LCRY Fi LCRY Fi DQRY Fi PQRY Fi *		CR.LF /PC3/ 0 /What is:/ 0 /1/ /-/ /Err / 0 this is the set of indirect jumps to redirect	
1492 FFBD 1483 FF6F 1484 FFC2 1485 FFC3 1486 FFC2 1486 FFC2 1488 FFCC 1488 FFCD 1489 FFCD 1499 FFC2 1491 1492 1493	000A00 000A 504350 00 57686174 2069733A 00 49 20 45727220	PCMESS FI Fi CQP.Y Fi LCRP.Y Fi LCRP.Y Fi DQRY FI Fi Fi		CR,LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Err / 0	
1492 FFB0 1493 FFBF 1494 FFC2 1485 FFC3 1486 FFC2 1487 FFCC 1488 FFC0 1489 FFC0 1499 FFC0 1499 FFC2 1491 1492 1493 1493 1495 FFD3 6E	000400 0004 504350 00 57686174 20697334 00 49 20 45727220 00 96 97 0377 8	PCMESS FI Fi COPY Fi LORY Fi DORY Fi DORY Fi * * * *		CR.LF /PCJ/ 0 /What is:/ 0 /// /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU]	
1492 FFBD 1493 FFBF 1484 FFC2 1485 FFC3 1486 FFC2 1486 FFC2 1486 FFC0 1489 FFC0 1489 FFC0 1499 FFC2 1499 FFC2 1491 1492 1493 1494 1495 FFD3 6E 1496 FFD7 6E	0000408 0004 504350 00 57686174 20697334 00 49 20 45727228 06 95 8377 8 95 8377 8	PCMESS FI Fi CQRY FI LQRY FI LQRY FI DQRY FI * * * * * * * * * * * * *		CR.LF /PCJ/ 0 /What is:/ 0 /1/ /-/ //Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [IRESU] [ISHI3]	
1492 FFB0 1493 FFB7 1494 FFC2 1495 FFC3 1496 FFC2 1496 FFC2 1498 FFC2 1499 FFC2 1499 FFC2 1491 1492 1493 1492 1493 1495 FFD3 6E 1496 FFD7 6E 1497 FFD8 6E	0000408 0004 504350 00 57686174 20697334 00 49 20 4572728 06 9F 0377 8 9F 0377 8 9F 0378 6	PCMESS FI FI CQRY FI LCRR FI LCRR FI LCRR FI FI * * * RESUI JI SWI31 JI SWI31 JI		CR.LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISHT2] [ISHT2]	
1492 FFBD 1493 FF6F 1484 FFC2 1485 FFC3 1485 FFC3 1486 FFC2 1488 FFC2 1488 FFC0 1489 FFC0 1499 FFC2 1499 FFC2 1499 FFC2 1491 1492 1494 1495 FFD3 6E 1496 FFD7 6E 1499 FFD8 6E	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0379 8 9F 0378 8	PCMESS FI FI CORY FI LORY FI DORY FI DORY FI DORY FI X * * * X SWI32 JI SWI32 JI SWI32 JI SWI32 JI	CB CCC CCC CCC CCC CCC CCC CCC CCC CCC	CR.LF /PCJ/ 0 /What is:/ 0 /// /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH13] [ISH13] [ISH12] [IFIR0]	
1492 FFB0 1493 FFB7 1494 FFC2 1495 FFC3 1496 FFC2 1496 FFC2 1498 FFC2 1499 FFC2 1499 FFC2 1491 1492 1493 1492 1493 1495 FFD3 6E 1496 FFD7 6E 1497 FFD8 6E	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS FI FI COPY FI LORY FI DORY FI DORY FI DORY FI SWI32 JI SWI32 JI SWI32 JI SWI32 JI SWI32 JI SWI32 JI SWI31 JI IROI JI SWI1 JI		CR.LF /PCJ/ 0 /What is:/ 0 /I/ /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [IFIR0] [IIR0] [ISH13]	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC2 1485 FFC2 1485 FFC2 1487 FFC2 1489 FFC2 1498 FFC2 1491 1492 1493 1494 1495 FFD3 6E 1496 FFD7 6E 1499 FFD7 6E 1499 FFD7 6E 1499 FFD7 6E 1590 FFE3 6E 1590 FFE3 6E	0000406 0004 504350 00 57686174 20697334 00 49 20 49 20 49 20 49 20 97 0377 8 97 0377 8 97 0377 8 97 0377 8 97 0377 8	PCMESS FI FI CQRY FI LGRY FI LGRY FI DGRY FI DGRY FI * * * * * * * * * * * * * * * * * * *	CB CC CC CC CC CC CC CC CC CC CC CC CC C	CR.LF /PCJ/ 0 /What is:/ 6 /// /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISHI3] [ISHI3] [ISHI2] [IFRQ]	
1492 FFB0 1493 FFB7 1493 FFC2 1495 FFC2 1495 FFC2 1496 FFC2 1496 FFC0 1499 FFC2 1499 FFC2 1491 1492 1492 1493 1494 1495 FFD7 6E 1496 FFD7 6E 1499 FFE7 6E 1499 FFE7 6E 1500 FFE7 6E 1501 FFE8 6E	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS F F CQRY F LGRY F LGRY F LGRY F LGRY F S LGRY F F S S S S S S S S S S S S S S S S S S		CR.LF /PCJ/ 0 /What is:/ 0 /// /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISHT2] [ISHT2] [ISHT2] [ISHT2] [ISHT2] [ISHT3] [ISHT3] [ISHT3]	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC3 1486 FFC2 1485 FFC3 1489 FFC0 1489 FFC0 1499 FF02 1491 1492 1493 1494 1495 FF03 6E 1496 FF07 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1500 FFE3 6E 1501 FFE3 6E 1502 1503	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS FI FI CGRY FI LGRY FI LGRY FI DGRY FI * * * * * * * * * * * * * * * * * * *		CR.LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Zerr / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH13] [ISH13] [ISH13] [ITRG] [ITRG] [ITRG] [INH1] the following hardware vectors reside in the top	
1492 FFB0 1493 FFBF 1494 FFC2 1495 FFC3 1496 FFC2 1495 FFC3 1496 FFC2 1499 FFC2 1499 FFC2 1499 FFC2 1499 FFC2 1492 1493 1495 FFD3 6E 1496 FFD7 6E 1499 FFC8 6E 1500 FFE7 6E 1501 FFEB 6E 1502 1503	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS F F CQRY F LGRY F LGRY F LGRY F LGRY F S LGRY F F S S S S S S S S S S S S S S S S S S		CR.LF /PCJ/ 0 7What is:/ 6 71/ /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [ISH12] [ISH12] [ISH11] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its	
1492 FFB0 1493 FFBF 1494 FFC2 1495 FFC3 1495 FFC3 1495 FFC3 1496 FFC2 1499 FFC2 1499 FFC2 1499 FFC2 1499 1492 1493 1494 1495 FFD3 6E 1496 FFD7 6E 1499 FFD7 6E 1590 FFE7 6E 1590 FFE7 6E 1591 FFEB 6E 1592 1594 1596	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS P PCQRY F CQRY F LQRY F LQRY F LQRY F * * * * * * * * * * * * *		CR.LF /PCJ/ 0 /What is:/ 6 /// /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISHI3] [ISHI3] [ISHI2] [ISHI2] [ISHI3] [ISHI1] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position.	
1482 FFB0 1483 FFB7 1484 FFC2 1485 FFC3 1486 FFC2 1486 FFC2 1489 FFC2 1489 FFC2 1499 FF02 1499 FF02 1491 1492 1493 1494 1495 FF03 GE 1496 FF07 GE 1499 FFE3 GE 1500 FFE3 GE 1501 FFEB GE 1502 1504 1505 FFF0	0000408 0004 504350 200 57686174 20697334 00 49 20 45727220 00 45727220 00 9F 0377 8 9F 0379 8 9F 0379 8 9F 0378 9 9F 0378 8 9F 0378 8 9F 0378 8 9F 0378 8	PCMESS P: PCGRY P: LGRY P: LGRY P: LGRY P: DGRY P: * * * * * * * * * * * * * * * * * * *		CR.LF /PCJ/ 6 /What is:/ 8 /1/ /-/ /Err / 8 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [ISH12] [IFIR0] [ISH12] [ISH13] [ISH13] [ISH13] [SH11] [SH11] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF6	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC2 1485 FFC2 1485 FFC2 1487 FFC2 1489 FFC2 1498 FFC2 1499 FFD2 1491 1492 1493 1494 1495 FFD3 GE 1496 FFD7 GE 1508 FFE7 GE 1508 FFE7 GE 1504 1505 FFF0 1536 1536	000408 0004 504350 00 57686174 20697334 00 49 20 45727220 06 9F 0377 8 9F 0377 8 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378 9F 0378	PCMESS FI FI CQRY FI LGRY FI LGRY FI DQRY FI * * * * * * * * * * * * * * * * * * *		CR.LF /PCJ/ 0 /What is:/ 6 /// /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISHI3] [ISHI3] [ISHI2] [ISHI2] [ISHI3] [ISHI1] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position.	
1492 FFB0 1493 FFBF 1494 FFC2 1485 FFC3 1486 FFC2 1486 FFC2 1498 FFC0 1499 FFC2 1499 FFC2 1499 FFC2 1492 1492 1492 1493 1494 1495 FFD3 6E 1499 FFD7 6E 1499 FFD5 6E 1499 FFD5 6E 1499 FFD5 6E 1500 FFE7 6E 1501 FFE8 6E 1502 1503 1504 1505 1536 1536 1536 1536	0000408 0004 504350 00 57686174 20697334 00 49 20 45727220 00 9F 0377 8 9F 0379 8 9F 0379 8 9F 0378 8	PCMESS P: PCGRY P: IERR P: LGRY P: LGRY P: DGRY P: * * * * * * * * * * * * *		CR.LF /PCJ/ 0 /What is:/ 0 /I/ /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [ISH12] [ISH12] [ISH12] [ISH12] [ISH13] [ISH13] [ISH13] [ISH13] [SH11] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF0 #E7F0	
1492 FFB0 1493 FFBF 1494 FFC2 1495 FFC3 1496 FFC2 1496 FFC2 1499 FFC5 1499 FFC5 1499 FFC5 1499 FFC2 1492 1493 1494 1495 FFD3 6E 1496 FFD7 6E 1499 FFC7 6E 1500 FFC7 6E 1500 FFC7 6E 1502 1504 1505 1506 1507 FFF6 1509 1509 FFF6 1509 FFF6 1509 FFF6	0000408 0004 504350 00 57686174 20697234 00 49 20 45727220 00 9F 0377 8 9F 0377 8 9F 0379 8 9F 0379 8 9F 0379 8 9F 0378 8	PCMESS P PCQRY F CQRY F LQRY F LQRY F LQRY F LQRY F * * * * * * * * * * * * *		CR, LF /PCJ/ 0 /What is:/ 0 /1/ /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [ISH12] [ISH12] [ISH12] [INH1] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF0 #EFF0 #ESUI SWI31	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC2 1485 FFC2 1486 FFC2 1489 FFC2 1489 FFC2 1499 FF02 1491 1492 1493 1494 1495 FF03 6E 1496 FF07 6E 1496 FF07 6E 1499 FF07 6E 1501 FFF0 6E 1502 1503 1504 1505 1506 1509 1509 1510 FFF0 1510 FFF0 1511 FFF2	000408 0004 50435D 00 57686174 20697334 00 49 20 45727220 06 97 0377 8 97 0378 8 97 10 10 10 10 10 10 10 10 10 10	PCMESS FI FI CQRY FI LQRY FI LQRY FI DQRY FI * * * * * * * * * * * * * * * * * * *	TOB TOCC TOCC TOCC TOCC TOCC TOCC TOCC T	CR.LF /PCJ/ 6 /What is:/ 6 /1/ /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH13] [ISH13] [ISH13] [ISH13] [ISH13] [INN1] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF6 #EFF6 PESUI SW131 SW131 SW131	
1492 FFB0 1493 FFBF 1494 FFC2 1495 FFC2 1495 FFC2 1495 FFC2 1496 FFC2 1499 FFC2 1499 FFC2 1499 FFC2 1499 FFD2 1491 1492 1493 1494 1495 FFD3 GE 1496 FFD7 GE 1499 FFD7 GE 1590 FFC7 GE 1590 FFC7 GE 1591 FFC8 1596 FFC7 1597 FFC8 1596 FFC7 1597 FFC8 1599 FFC8 1599 FFC8 1599 FFC8 1599 FFC8 1599 FFC8 1599 FFC8 1599 FFC8 1591 FFC8	0000408 00064 504350 00 57686174 20697334 00 49 20 45727220 00 9F 0377 8 9F 0377 8 9F 0379 8 9F 0379 8 9F 0378 8 9F 038 8 9F	PCMESS PI PCQRY PI CQRY PI LQRY PI LQRY PI * * * * * * * * * * * * * * * * * * *	тсв тсв тсв тсв тсв тсв тсв тсв тсв тсв	CR, LF /PC3/ 0 /What is:/ 0 /// /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. (IRESU) (ISW13) (ISW13) (ISW12) (ISW12) (ISW12) (ISW13) the following handware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF0 #E7F0 #ESUI SW131 SW121 SW121 FIR01 FIR01	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC3 1486 FFC2 1485 FFC3 1489 FFC2 1489 FFC2 1499 FF02 1499 FF02 1491 1492 1493 1494 1495 FF03 6E 1496 FF07 6E 1496 FF07 6E 1499 FFC3 6E 1500 FFC7 6E 1500 FFC7 6E 1500 FFC7 6E 1502 1504 1505 1536 1536 1536 1536 1536 1536 1537 FFF0 1541 FFF4 1513 FFF6	000A00 000A 000A 50435D 00 57686174 2069733A 00 49 20 45727220 00 97 0377 8 97 0379 8 97 0378 8 97 0383 8 77 700 700 700 700 700 700 700	PCMESS PI PCGRY FI LGRY FI LGRY FI LGRY FI DGRY FI * * * * * * * * * * * * *	TOB TOCC TOCC TOCC TOCC TOCC TOCC TOCC T	CR.LF /PCJ/ 6 /What is:/ 6 /1/ /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH13] [ISH13] [ISH13] [ISH13] [IFIRG] [IFIRG] [IFIRG] [INM1] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF6 #EFF6 #EFF6 #EFF6 #EFF6 #EFF6 #EFF6 #EFF6	
1482 FFB0 1483 FFB7 1484 FFC2 1485 FFC3 1486 FFC2 1485 FFC3 1486 FFC2 1489 FFC2 1499 FF02 1499 FF02 1499 FF03 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1500 FF07 6E 1500 FFC9 1501 FFE8 6E 1502 1504 1502 1504 1505 1536 1510 FFF0 1510 FFF0 1511 FFF2 1514 FFF8 1515 FFF6 1514 FFF8 1516 FFF6 1516 FFF6	0000408 00064 504350 00 57686174 20697334 00 49 20 45727220 00 9F 0377 8 9F 0377 8 9F 0379 8 9F 0379 8 9F 0378 8 9F 038 8 9F	PCMESS FI FI CQRY FI LQRY FI LQRY FI LQRY FI W * * * * * * * * * * * * * * * * * *	1768 1766 1766 1766 1766 1766 1766 1766	CR, LF /PC3/ 0 /What is:/ 0 /// /-/ /Err / 0 this is the set of indirect jumps to redirect the interrupt vector addresses. (IRESU) (ISW13) (ISW13) (ISW12) (ISW12) (ISW12) (ISW13) the following handware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF0 #E7F0 #ESUI SW131 SW121 SW121 FIR01 FIR01	
1482 FFB0 1483 FFBF 1484 FFC2 1485 FFC2 1485 FFC2 1485 FFC2 1486 FFC2 1489 FFC2 1490 FFD2 1491 1492 1493 1494 1495 FFD3 GE 1496 FFD7 GE 1598 FFD7 GE 1598 FFD7 GE 1598 FFD7 GE 1598 FFD7 GE 1598 FFF5 1599 FFF6 1599 FFF6 1510 FFF6 1511 FFF6 1512 FFF6 1513 FFF6 1515 FFF6 1515 FFF6	000A08 000A 000A 50435D 00 57686174 2069733A 00 49 20 45727220 00 9F 0377 8 9F 0377 8 9F 0377 8 9F 0378 6 9F 0378 6 9F 0378 6 9F 0378 6 9F 0378 8 9F 0378 8 9F 0388 8 9F 03888 8 9F 03888 8 9F 03888 9F 0388 8	PCMESS PI PCGRY PI IERR PI LGRY PI LGRY FI DGRY PI * * * * * * * * * * * * * * * * * * *	1768 1767 1767 1767 1767 1767 1767 1767	CR.LF /PCJ/ 0 /What is:/ 6 /// /-/ /Err / 6 this is the set of indirect jumps to redirect the interrupt vector addresses. (IRESUJ [ISW12] [ISW12] [ISW12] [ISW12] [ISW12] [ISW13] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF6 #EFF6 #EFF6 RESUI SW131 SW121 FIR01 IR01 SW11	
1482 FFB0 1483 FFB7 1484 FFC2 1485 FFC3 1486 FFC2 1485 FFC3 1486 FFC2 1489 FFC2 1499 FF02 1499 FF02 1499 FF03 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1499 FF07 6E 1500 FF07 6E 1500 FFC9 1501 FFE8 6E 1502 1504 1502 1504 1505 1536 1510 FFF0 1510 FFF0 1511 FFF2 1514 FFF8 1515 FFF6 1514 FFF8 1516 FFF6 1516 FFF6	0000408 00004 504350 00 57686174 20697334 00 49 20 45727220 00 97 0379 8 97 0379 8 97 0378 8 0378 8 03788 8 03788 8 03788 8 0378 8 03788 03788 03788 8	PCMESS PI PCMESS PI PCQPY PI LGRP PI LGRP PI LGRP PI SWI22 J SWI22 J SWI22 J SWI22 J SWI22 J SWI22 J SWI23 J SWI23 J SWI2 J SWI24 J SWI24 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25 SWI25	тсв тсс тсс тсс тсс тсс тсс тсс тсс тсс	CR.LF /PCJ/ 6 /What is:/ 8 /I/ /-/ /Err / 8 this is the set of indirect jumps to redirect the interrupt vector addresses. [IRESU] [ISH12] [ISH12] [ISH12] [ISH12] [ISH12] [ISH13] the following hardware vectors reside in the top 16 bytes of memory when the monitor is in its standard position. #FFF6 #E7F8 PESUI SW121 SW	

. command 9, 13 + in copy command 15 in indexed addressing 52 ++ in indexed addressing 52 \$ in hexadecimal value 49 ; in modify command 11 - in indexed addressing 52 in modify command 11 in save command 14 -- in indexed addressing 52 in indexed addressing 52 Err error message 10, 14 ? in load command 14 A register 38 ABX instruction 42 Accumulator addressing 49 Accumulator-offset indexed addressing 52 Accumulators 38 ACK signal 32 Adding to monitor commands 16CASIN signal 29Address RESET 37 AddressesCASOUT signal 29interrupt vector 19 AddressingCassette calibration 5modes 49Cassette calibration 5 modes 49 modes instruction set and 38 Addressing modes accumulator 49 accumulator-offset indexed 52 auto decrement indexed 52 auto increment indexed 52 constant offset indexed 51 direct 50 extended 49 extended indirect 50 immediate 49 indexed 51 indexed indirect 53 inherent 49 PCrelative 54 program-counter relative 54 register 50 relative 53 ANDCC instruction 47 Audio cassette interface 28 Auto decrement indexed addressing 52 increment indexed addressing 52 B register 38 BA signal 26 Backplane connections 76 Baud rate ,selection 22

Binary-to-decimal program 56 program re-entrant 58 Binomial coefficient program 60 Bit halfcarry 41 BIT instruction 65 BLOCK0 signal 23 BOOT routine 36 Branch instructions 45 Branches long 53 short 53 Breakpoint cancelling 9 command 8, 13 Breakpoints 7 Buffering bus 25 Bus available signal 26 buffering 25 BUSY signal 32 C command 15 flag 40 Calibration cassette Cancelling breakpoint 8 Capacitor values 67 Carriage-return character 35 Carry flag checksum 30 format 30 input 28 interface audio 28 interface card 74 interface circuit for 29 interface commands 13 interface connections 74 output 28 verify from 14 CBIN1 routine 36 CBIN2 routine 36 CC register 39, 40 CCLOCN routine 37 Character carriage-return 35 delete 35 form-feed 35 line-feed 35 Checksum 10 cassette 30 Circuit for cassette interface 29 Clock system 26 CMDPAR routine 36 Colour code resistor 70 Command passing 2, 20 passing program illustrating 21 Commands . 9, 13 breakpoint 8, 13

```
(1)
```

C 15 cassette interface 13 copy 15 D 15 disk 15 disk interface 15 do trace 9, 13 F 10, 14 G 5, 12 go 5, 12 L 6, 14 load 6, 14 M 4, 11 . MG 12 modify 4, 11 MP 12 MR 7, 12 MV 8, 12 P 7, 12 printer interface 15 proceed 7, 12 R 7, 13 registers 7, 13 S 6, 13 save 13 store 6 T 9, 13 trace 9, 13 ✓ 8, 13 Compatability with 6502 65 with 6800 62 Component layout 6809 card 69 CONCHR routine 33 Condition code register 40 Condition-code flags CONIN routine 33 CONOUT routine 35 Constant offset indexed addressing 51 Copy command 15 Crystal mounting 73 D command 15 register 38 Debugging program 13 Decoding memory 23 Delayed BA signal 27 Delete character 35 Direct addressing 50 Disk command 15 interface commands 15 routines 36 DISPCH routine 37 DISPLA routine 35 Display 2 DMA 26 Do trace command 9, 13 DP register 38 Driver routines 20

DRVRDY routine 36 Dump program 17 E flag 41 signal 29 Entering a program 4 Entire flag 41 Entry point reset 37 Err error message 10 Error messages 10 Error messages -Err 10, 14 Err 10 I-Err 10, 19 Rom? 10 What is; 4, 10 Exchange registers 47 EXG instruction 47 Expansion monitor 16 Extended addressing 49 indirect addressing 50 Extra ROM 2732 as 25 ROM 16, 25 F command 10, 14 Finish command 10, 14 FIRO mask 41 service routine program 19 Flag C 40 carry 40 E 41 entire 41 N 41 negative 41 ONLINE 33 overflow 40 ✓ 40 Z 41 zero 41 Form-feed character 35 Format cassette 30 VDU 34 G command 5, 12 GETCHR routine 34 GETHEX routine 34 GETHXS routine 34 Go command 5, 12 Half-carry bit 41 Hardware description 23 Hexadecimal value S in 49 HEXOUT routine 35 I-Err error message 10, 19 I/O driver routines 20 routine serial 28 Immediate addressing 49 Index register instructions 44 (2)

registers 38 Indexed addressing 51 indirect addressing 53 jumps 54 Input cassette 28 routines 33 Instruction set 42 set and addressing modes 38 Instructions ABX 42 ANDCC 47 BIT 65 branch 45 EXG 47 index register 44 LEA 42 load effective address 42 miscellaneous 45 MUL 40, 47 multiply 47 ORCC 47 pull 46 push 46 SEX 47 sign extend 47 stack pointer 44 SWI 7, 48 SWI2 48 SWI3 48 SYNC 48 synchronize with interrupt 48 TFR 47 16-bit 44 8-bit 43 Interface audio cassette 28 circuit for cassette 29 keyboard 30 printer 32 Interrupt instruction synchronize with 48 keyboard 31 vector addresses 19 vectoring 19 Interrupts 3 software 48 Introduction 1 IRQ mask 41 Jumps indexed 54 Keyboard 1 connections 74 interface 30 interrupt 31 Kit assembly instructions 67 L command 6, 14 LEA instruction 42 Line-feed character 35

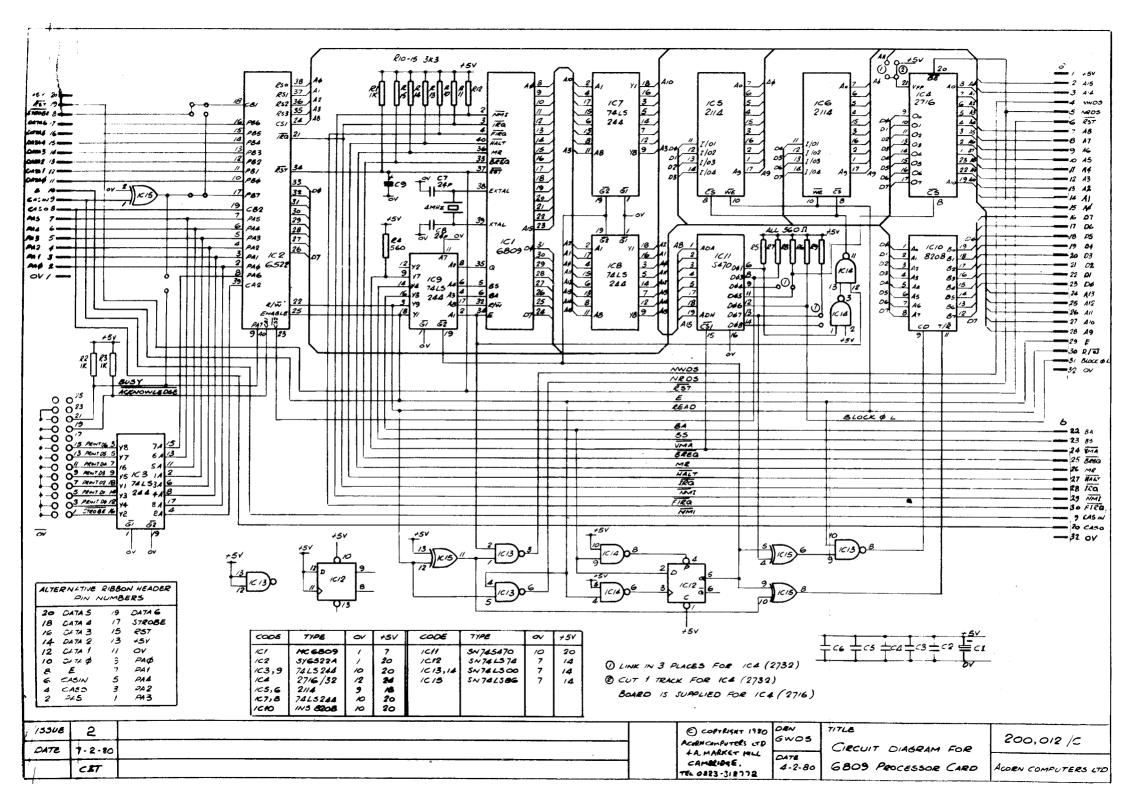
Listing of monitor program 77 Load command 6, 14 effective address instruction 42 with offset 14 Long branches 53 M command 4, 11 Map memory 24 Mask FIRO 41 IRQ 41 MCASIN routine 36 MCASOP routine 36 Memory decoding 23 map 24 organization 23 MEMUSE routine 20, 37 MG command 12 Miscellaneous instructions 45 routines 37 Modes addressing 49 Modify command 4, 11 Monitor expansion 16 operation 4 program 77 return to 12 Monitor commands 1 adding to 16 summary 11 MP command 12 MR command 7, 12 MUL instruction 40, 47 Multiply instruction 47 MV command 8, 12 N flag 41 NAMEIN routine 34 Negative flag 41 NUMB routine 34 Offset load with 14 ONCARD signal 23 ONLINE flag 33 OPAREG routine 35 OPARSP routine 35 OPCRLF routine 35 OPXREG routine 35 ORCC instruction 47 Output cassette 28 device terminal as 21 routines 34 Overflow flag 40 P command 7, 12 Parts list 6809 card 68 Passing command 20 PC register 40 PC-relative addressing 54 Position-independent code 55 Postbyte for EXG/TFR 47 (3)

Power supply 76 PRINT routine 35 Printer interface 32 interface commands 15 Program counter 40 debugging 13 entering a 4 Program-counter relative addressing 54 Programming model of 6809 39 re-entrant 56 recursive 60 techniques 55 Programs binary-to-decimal 56 binomial coefficient 60 dump 17 FIRQ service routine 19 illustrating command passing 21 monitor 77 re-entrant binary-to-decimal 58 Pull instruction 46 order 46 Push instruction 46 order 46 R command 7, 13 Re-entrant binary-to-decimal program 58 programming 56 Recursive programming 60 Register addressing 50 Registers Ă 38 B 38 CC 39, 40 command 7, 13 condition code 40 D 38 DP 38 exchange 47 PC 40 S 38 transfer 47 U 38 user's 7 X 38 Y 38 Relative addressing 53 addressing program-counter 54 Relocatable code 55 RESET address 37 Reset entry point 37 signal 26 Resistor colour code 70 Return to monitor 12 ROM extra 16, 25 Rom? error message 10 Routines

BOOT 36 CBIN1 36 CBIN2 36 CCLOCN.37 CMDPAR 36 CONCHR 33 CONIN 33 CONOUT 35 disk 36 DISPCH 37 DISPLA 35 20 driver 36 DRVRDY GETCHR 34 GETHEX 34 34 GETHXS 35 HEXOUT 33 input MCASIN 36 36 MCASOP MEMUSE 20, 37 miscellaneous 37 NAMEIN 34 NUMB 34 OPAREG 35 OPARSP 35 OPCRLF 35 OPXREG 35 output 34 PRINT 35 serial I/O 28 STRING 35 tape 36 TRNSFR 36 using SYNC 48 S command 6, 13 register 38 Save command 13 non-contiguous blocks 18 Scrolling VDU 35 Serial I/O routine 28 terminal 21 Service routine program FIRQ 19 SETDP directive 50 SEX instruction 47 Short branches 53 Sign extend instruction 47 Signals ACK 32 BA 26 BLOCK0 23 bus available 26 BUSY 32 CASIN 29 CASOUT 29 delayed BA 27 E 29 ONCARD 23 (4)

reset 26 VMA 2.6 Software description 33 interrupts 48 Stack depth 23 pointer instructions 44 pointers 38 Store command 6 STRING routine 35 SWI instruction 7, 48 SWI2 instruction 48 SWI3 instruction 48 SYNC instruction 48 routine using 48 Synchronize with interrupt instruction 48 System clock 26 T command 9, 13 Tape routines 36 Techniques programming 55 Terminal as output device 21 serial 21 TFR instruction 47 Trace command 9, 13 facility 9 mode 31 Transfer registers 47 TRNSFR routine 36 U register 38 User stack 60 User's registers 7

V command 8, 13 flag 40 VDU card 74 format 34 scrolling 35 Vector addresses interrupt 19 Vectoring interrupt 19 Verify from cassette 14 VMA signal 26 What is; error message 4, 10 X register 38 Y register 38 Z flag 41 Zero flag 41 16-bit instructions 44 2732 as extra ROM 25 6502 compatability with 65 6800 compatability with 62 equivalents to 6809 instructions 62 6809 card component layout 69 card parts list 68 instructions 6800 equivalents to 62 programming model of 39 8-bit instructions 43





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