WDC

THE WESTERN DESIGN CENTER, INC.
2166 E. Brown Rd. Mesa, AZ 85213
Ph 480-962-4545   Fx 480-835-6442
www.westerndesigncenter.com

W65C134S Monitor ROM REFERENCE MANUAL

Release 2.0

October 18, 2000

Information in this document is subject to change without notice and does not represent a commitment on the part of The Com Log Company, Inc. or The Western Design Center, Inc.

W65C134 Monitor ROM ($F000)
Reference Manual

Thank you for choosing the state-of-the-art features of the W65C134 microprocessor from Western Design Center. This manual describes the internal ROM monitor developed by the Com Log Company, Inc. for the W65C134 microprocessor.

For best results, we recommend that you please carefully read this manual completely before you attempt to use the W65C134 monitor. This manual contains important information on the proper operation of the monitor and it's library of subroutines.

MONITOR PRE-REQUISITES

1. A W65C134 microprocessor from Western Design Center. Configuration:
   - ❍ The default clock (CLK) for the W65C134 must operate at 32,768 Hz.
   - ❍ No external RAM or EPROM or I/O is necessary.

2. This monitor expects a terminal (or computer emulator) to be connected to the serial UART port. The terminal/interface must be configured as follows:
   - ❍ All signals must be converted to TTL levels.
   - ❍ Hardware handshaking.
   - ❍ 8 bit data.
   - ❍ No parity.
   - ❍ 9600 Baud (Unless otherwise noted).

3. External elements must not conflict with the W65C134 monitor memory mapping:
   - ❍ Address range $0100-$01FF is reserved for stack space in external RAM.
   - ❍ Address range $0200-$7FFF is available for user memory. (Monitor assumes RAM.)
   - ❍ Address range $8000-$9FFF is available for user memory. (Monitor assumes EPROM.)
   - ❍ Address range $A000-$EFFF has been used by Com Log BASIC and other products in external EPROM. It is available for user programs in non-conflicting applications.

NOTICE

The Western Design Center, Inc. has made every attempt to ensure that the information in this manual is complete and accurate. However, WDC assumes no liability for errors, or for any damages that result from the use of this document or the W65C134 microprocessor.

Users of the W65C134 Monitor firmware should note that it is both possible and relatively simple to connect *any* microcomputer equipment to external devices, and then to harm or destroy those devices (or anything that they may control). WDC assumes no liability for any connections or use of the W65C134 microprocessor and associated firmware.

Nothing herein shall be construed as a recommendation to use the W65C134 in violation of existing patents or other rights of third parties.

Information in this document is subject to change without notice and does not represent a commitment on the part of The Western Design Center, Inc. or The Com Log Company, Inc. for future products.

Table Of Contents

Introduction

The W65C134 chip has firmware to handle interrupts, serial buffering, the real time clock, and the power down mode. It also has a serial port user interface program, called the *monitor*. The monitor acts as a simple operating system, allowing the user to examine registers and memory, load and save programs, and debug applications in RAM.

The monitor is screened into the mask ROM onboard the W65C134 microprocessor chip. It will run on the W65C134 alone, and needs no ROM or RAM outside the W65C134 chip. It may be used with a variety of different circuits which use the W65C134.

W65C134 Internal Hardware

The W65C134 microprocessor chip comprises a W65C02 CPU surrounded by RAM, ROM, several I/O ports and timers, and a serial UART. There are also built-in special controls for external elements such as chip selects and clocks. All hardware features of the W65C134 chip are described in detail by Western Design Center.[1] This section will merely review the main components.

Central Processor Unit (CPU

Development Tools

The Com Log Company, Inc. offers a variety of hardware and software development tools for WDC products. These include: **THE PROJECT COMPUTER**, a low-cost prototyping platform for the W65C134. Several "ROMable" software products are also available.

The CPU on the W65C134 chip is essentially the W65C02 microprocessor from Western Design Center.[2] It's instruction set is compatible with most "65C02" microprocessors available from other sources. Numerous programming instruction books have been published on this subject. Many software tools (i.e. compilers, assemblers, operating systems, etc.) are available to developers from third party sources.

Random Access Memory (RAM)

The W65C134 has 196 bytes of internal RAM. This RAM can appear in several places. The W65C02 CPU uses the area from $0100 to $01FF as a stack, and the area from 0000-$00FF as special page zero memory. (Page 0 memory can be used for short addressing and for indirect addressing modes.)

If CS6 is active, then the internal RAM appears from $0040 to $00FF. If CS6 is not active, then the internal RAM appears in two places - $0040 to $00FF, and $0140-$01FF. The selection allows systems with external RAM to separate the stack from page 0, and users without external RAM some memory in both the stack area and page 0.

[1] Refer to WDC literature: W65C134 INFORMATION, SPECIFICATION AND DATA SHEET for details

[2] A detailed description may be found in: W65C02 INFORMATION, SPECIFICATION AND DATA SHEET from WDC.

Read Only Memory (ROM)

The W65C134 has a 4K byte mask ROM from $F000 to $FFFF. This may be enabled or disabled by hardware. The mask ROM is added to the chip as part of the manufacturing process. It cannot be changed by the user. On power-up or reset, the internal mask ROM may be enabled by bringing up the BE line before the RESET line goes TRUE, and disabled by bringing up the BE line after RESET line goes TRUE. If the internal ROM is enabled, I/O ports 0,1, and 2 initialize as usable I/O ports. If the internal mask ROM is disabled, I/O ports 0, 1 and 2 initialize to control the address and data bus.

Input/Output (I/O) Ports

The W65C134 has a large number of bi-directional I/O ports. Each port has a data register and a data direction register. The data direction register allows the program to select any bit as an input or an output. The data register allows the program to read inputs or to write outputs. Both the data direction register and the data register can be read or written; the data direction register will read back the last data written to it, and the data register will read the input lines or the data output, depending on the associated data direction register bits.

If an I/O port is selected as an input, the physical input will act as a light pull up or pull down latch. If the input is high, a 5 microamp pull-up is applied to the input. If the input is low, a 5 microamp pull-down is applied to the input. As the input changes from high to low or low to high, the pull up/down will abruptly change, thus 'pulling' the line quickly through the middle, undefined logic state.

Another feature of these devices is that they latch input data. If a device connected to the inputs and set to some value is then tri-stated, the original value will stay on the inputs due to the pull up/down latching.

I/O ports 0, 1 and 2 can be selected as either general I/O ports, or as the address and data bus for the W65C02 processor. Typical applications use these ports as the microprocessor bus. The only way they may be used as I/O ports is when a program is masked into the ROM, and no external EPROM or RAM is used.

Clocks

The W65C134 has two clock inputs, called CLK (the default clock) and FCLK (a separate fast clock). On reset, the W65C134 uses the CLK oscillator to run the W65C02 core processor.

The second oscillator, FCLK, is under program control. Software has the option to turn it on or off, and to select it as the input clock for the W65C02. On reset, this oscillator is turned off.

Programmable Timers

The W65C134 has four timers - timer 1, timer 2, timer A and timer M. Timers 1 and 2 are general purpose, and can select either the FCLK or the CLK input as their reference.
Timers 1 and 2 are usually used to provide a source of regular interrupts, and have several modes available.

Timer A can select either the W65C02's clock (whichever it happens to be) or a separate input from pin 2 of the chip. Timer A is used for the baud rate generator, and therefore can't be used for other purposes if the serial UART is in use. When not in use as a baud rate generator, timer A can be used to output an exact frequency square wave from pin 3 of the chip.

Timer M is the 'Watchdog' timer. If this timer is never started, nothing happens. Once started, it cannot be stopped by software. This timer counts down, and pulls the system RESET line when it hits zero. It may be reloaded under program control to prevent it from reaching zero. The purpose of Timer M is to provide a way to insure that execution won't get caught in an infinite loop or other error path. A macro or subroutine may be used to reload the timer from a variety of places in the normal program flow. If an error prevents the software from operating properly, then the timer will eventually reach 0. This will reset the W65C134, hopefully allowing the application to regain control.

Serial Universal Asynchronous Receiver-Transmitter (UART)

The W65C134 has one serial port built into the chip. Enabling the serial port changes pins 2 and 3 on the chip from general purpose I/O to receive and transmit data. The UART uses Timer A as a baud rate generator. The hardware provides one byte buffering in both directions, parity generation and checking, and interrupts on RX full and TX empty. The nature of the operation of the UART requires that the serial routines use an interrupt for service. The W65C134's UART cannot easily be used in a 'Polled' mode of operation.

Chip Selects

The W65C134 has eight outputs which may be selected as simple outputs or as chip selects. The chip select outputs reduce outside logic and provide selects to external devices when particular address ranges are in use. Enabling particular chip selects may change the way internal RAM is accessed. The chip selects are discussed in detail in the W65C134 literature from Western Design Center.

W65C134 Monitor Configuration

W65C134 MONITOR ROM DESIGN GOALS

- ❍ The monitor must be able to be 'shut off' that is it must exit to another program immediately after reset if necessary.

- ❍ The monitor must handle the serial port on the W65C134, and must provide routines such that another program can easily use the serial port via the monitor.

- ❍ The monitor must be able to load other programs into RAM, and provide some debugging capabilities.

- ❍ The monitor must maintain a time of day clock, and be capable of maintaining that clock on minimum power.

- ❍ The monitor must fit in the $F000 to $FFFF memory space.

The firmware installed from $F000 to $FFFF provides several functions and utilities consistent with the prior design goals. These include:

- ❍ Power-up & Reset Initialization
- ❍ User Console Interface Logic
- ❍ Real-time Clock Support
- ❍ Library of Utility Subroutines

The W65C134 is a very versatile chip, but this versatility comes at the price of a complex initialization. The monitor firmware will initialize the chip on reset to provide the proper memory mapping of internal and external RAM and registers. The initialization will also set up the interrupts used by the rest of the firmware. The power up sequence is shown in the software listings.

The W65C134 monitor uses the serial port to allow the user/developer to access the hardware and applications software. It allows uploading and downloading of programs, memory and register inspection and modifications, and general debugging. Complete descriptions of monitor functions appear elsewhere in this manual.

The firmware interrupt routine on timer 2 provides a complete real-time clock (RTC) implementation in RAM. The real-time clock updates a set of memory locations which can be read or written by any software. The real-time clock implementation also includes a configurable alarm function. The RTC interrupt handler will set a flag if the alarm time matches the current time.

The W65C134 firmware also contains many useful subroutines. A vector table is used to provide easy access from application software. These library routines include buffered serial functions, real-time clock support, and I/O control. Each of these functions and its operation is discussed later in this manual.

External Clocks

There are two clock inputs on the W65C134 chip. The default clock is called CLK. It must be connected to a 32,768 Hz watch crystal (or comparable oscillator) if the W65C134 monitor ROM is to be used. This frequency makes the CLK oscillator a excellent time base reference (32,768 is 2^{15}, which easily divides to 1 second). The crystal is a very low power oscillator, generally consuming less than 20 microamps. On reset, the W65C134 defaults to this oscillator to run the 65C02 core processor.

Helpful Hint #1

The Project Computer uses an FCLK oscillator of 2.4576 MHz. This particular frequency was chosen because this clock source (FCLK) is also used for the baud rate generator. The frequency, 2.4576 MHz, just happens to divide down to provide very accurate serial baud rates from 50 baud to 38,400 baud. **- COM LOG**

The second oscillator, FCLK (fast clock), is under program control. This oscillator is turned off upon hardware reset of the W65C134. The FCLK frequency may be selected by the application designer. The W65C134 firmware will check the FCLK speed against the CLK frequency (which is ALWAYS 32 KHz) during initialization. This test allows the firmware to select values for the baud rate and system tick timers.

Acceptable crystal values[3] for FCLK include: 1.8432 MHz, 2 MHz, 2.4576 MHz, 3.6864 MHz, and 4.00 MHz. The values 2 MHz and 4 MHZ should be avoided if the serial port is used at high baud rates; the baud rate error becomes significant above 2400 baud with those crystals. It should be noted that this processor does not have a clock divider built in. This means that the processor and memory cycle times are the same as a single clock time. Therefore, 2.45 MHz on the Project Computer is roughly as fast as an 8 MHz IBM PC.

The W65C134 monitor firmware turns on the FCLK shortly after reset initialization. The monitor then waits for the FCLK oscillator to stabilize, before selecting the FCLK as the operating frequency for the core 65C02. In most cases, this selection is not changed - the processor always runs from the FCLK. The one exception is power down. In this case, the FCLK is turned off and the 65C02 core is run from the low frequency, low power oscillator.

Terminal/User Console

This monitor expects a terminal (or computer emulator) to be connected to the serial port. The terminal must be configured as follows:

- ❍ Hardware handshaking.
- ❍ 8 bit data.
- ❍ No parity.
- ❍ 9600 Baud (Unless otherwise noted).

Signal conditioning and conversion to/from TTL levels to/from RS-232 (or other links) must be performed by external circuitry.

The W65C134's firmware provides full interrupt control of the serial port, along with software buffering of transmit and receive data. The user can select the size and placement of the software serial buffers. The W65C134 monitor designates two more I/O pins for hardware handshaking of the serial port. These are pins 4 and 61. Pin 4 is used as an output, indicating the serial receive buffer is ready to accept data. Pin 61 is an interrupt input, and indicates to the monitor that the outside device can or cannot accept data.

Restrictions On Application Software

The W65C134 has four timers - timer 1, timer 2, timer A and timer M. Timer 1 is used by the monitor for a system .001 second interrupt, and timer 2 is used for a 1 second, time of day interrupt. Timer A is used for the baud rate generator, and therefore can't be used for other purposes. Timer M is the 'Watchdog' timer. It is not used by the W65C134 monitor, and is never enabled. Application software may use this timer. (Be careful in power down modes - it's not reset.)

The W65C134 monitor firmware assumes that I/O ports 0, 1 and 2 will be used as the address and data bus for the 65C02 processor. They are used to access external EPROM, RAM and I/O.

[3] Initial versions of the W65C134 did not support clock rates higher than the 2.4576 MHz crystal. This limitation is not a problem with the chips manufactured today.

W65C134 Monitor Memory Map

Address Range	Function
$0000-$002F	W65C134 internal registers.
$0040-$00FF	W65C134 internal RAM.
$0100-$01FF	W65C134 Stack Space in External RAM.
$0200-$7FFF	External Static RAM.
$8000-$9FFF	EPROM for user programs.
$A000-$DFFF	Com Log BASIC interpreter, if installed.
$E000-$EFFF	Com Log external firmware, if installed.
$F000-$FFFF	W65C134 Internal ROM, monitor firmware.

Memory Map Assumptions

The W65C134 monitor resides in the 4K byte mask ROM from $F000 to $FFFF. This may be disabled if desired. The monitor executes an initialization sequence after reset occurs. It turns on the external bus, checks locations $8000-$8002 and jumps to $8004 if a 'WDC' is found. Using this signaling, a system can be designed to start on the internal ROM, then switch under software control to external ROM or RAM.

An identical set of semaphores ('WDC') can exist in RAM at location $0200 (with a JMP at $0204). This is checked immediately *after* the semaphore at $8000 is checked. Using the $0200 semaphore, a program in RAM can restart after a reset (but NOT after a power down - external RAM data is lost).

The W65C134 uses $0140-$01FF as a stack, and the area from 0000-$00FF as special page zero memory. (Page 0 memory can be used for short addressing and for indirect addressing modes.) If there is no external RAM in the system, these blocks both actually reference the same page of internal RAM.

Reset Initialization Sequence

The monitor was designed for the internal ROM of the W65C134. It assumes that the RESET vector is entered from an internal ROM reset.[4]

Reset may be either a triggered reset or a power-up reset. There is no simple way for the firmware to differentiate which reset occurred. However, some semaphores (flags) in memory tell the monitor that certain aspects of the system have already been initialized. These should not be changed by the reset initialization sequence.

[4] This means that the code is started with the BCR=00.

There is a checksum associated with the time-of-day clock and baud rate. If the checksum is correct, then the time-of-day clock has been running. If this is the case, the clock value and baud rate will not be re-initialized.

Helpful Hint #2

There is only way to get the time-of-day (ToD) clock or the baud rate to completely reset under the W65C134 monitor. Completely remove power from the processor long enough for the memory to scramble. (One minute is usually sufficient.)

- Com Log

The second reset semaphore is a three byte sequence in RAM indicate that the system is operating in low-power mode. If this is indicated, the reset initialization sequence will jump to the "power down" vector (NE46) immediately.

The following pages describe the steps taken by the W65C134 monitor from power-up (or triggered RESET) to a command prompt. This is intended as an overview only; not a specific, line by line analysis of the code.

STATE #0 - POWER OFF or Any previous powered state, including LOW-POWER MODE.
- RESET OCCURS! (This may result from POWER ON or a triggered RESET.)

STATE #1 - "Essential initialization"
- Disable interrupts
- Reset stack
- Clear decimal mode
- Check for low-power semaphore;
  - **If** semaphore is set, go to LOW-POWER MODE.
  - **Else** proceed to STATE #2.

STATE #2 - "Enable External Memory"
- Set BCR to 01 (This turns on external address and data lines.)
- Set PCS3 to $C0 (This turns on RAM and ROM chip selects from the W65C134.)
- Check location $8000-$8002 for the string 'WDC'.
  - **If** the string is there, transfer control to USER program in EPROM. (JMP $8004)
  - **Else** proceed to STATE #3.

STATE #3 - "Check External RAM"
- Check locations $0200-$0202 for the string 'WDC'.
  - **If** the string is there, transfer control to USER program in RAM. (JMP $0204)
  - **Else** proceed to STATE #4.

STATE #4 - "Miscellaneous Initialization"
- Start the fast clock. (Don't use it, just start it.)
- Initialize the RAM interrupt vectors.
- Delay while fast clock becomes stable. (256 * 5 cycles)
- Switch to fast clock.
- Enable the NE46 interrupt, but not the overall I bit. (This interrupt is used to detect power going down.)
- Set Timer 2 for a 1 second interrupt (ToD timer).
- Enable T2 interrupt (but not the I bit yet).
- Set up pointers to the serial buffers in P RAM.
- Calculate the fast crystal frequency by comparing it to the 32 KHz clock crystal.
- Finally, check the Time -of-Day clock checksum.
  - **If** the clock checksum is valid, proceed to STATE #6.
  - **Else** perform additional initialization in STATE #5.

STATE #5 - "Initialize Clock/UART"
- Reset the Time-of-Day clock and reset the baud rate counters for the serial port to the default values.
- Set up control port of serial port for Xmit and RX.
- Proceed to STATE #6.

STATE #6 - "Check for $E000 EPROM"

- Check location $E000 for $4C.

  If a $4C is present, transfer control to the Project Computer[5] program EPROM for additional initialization. (JSR $E000)

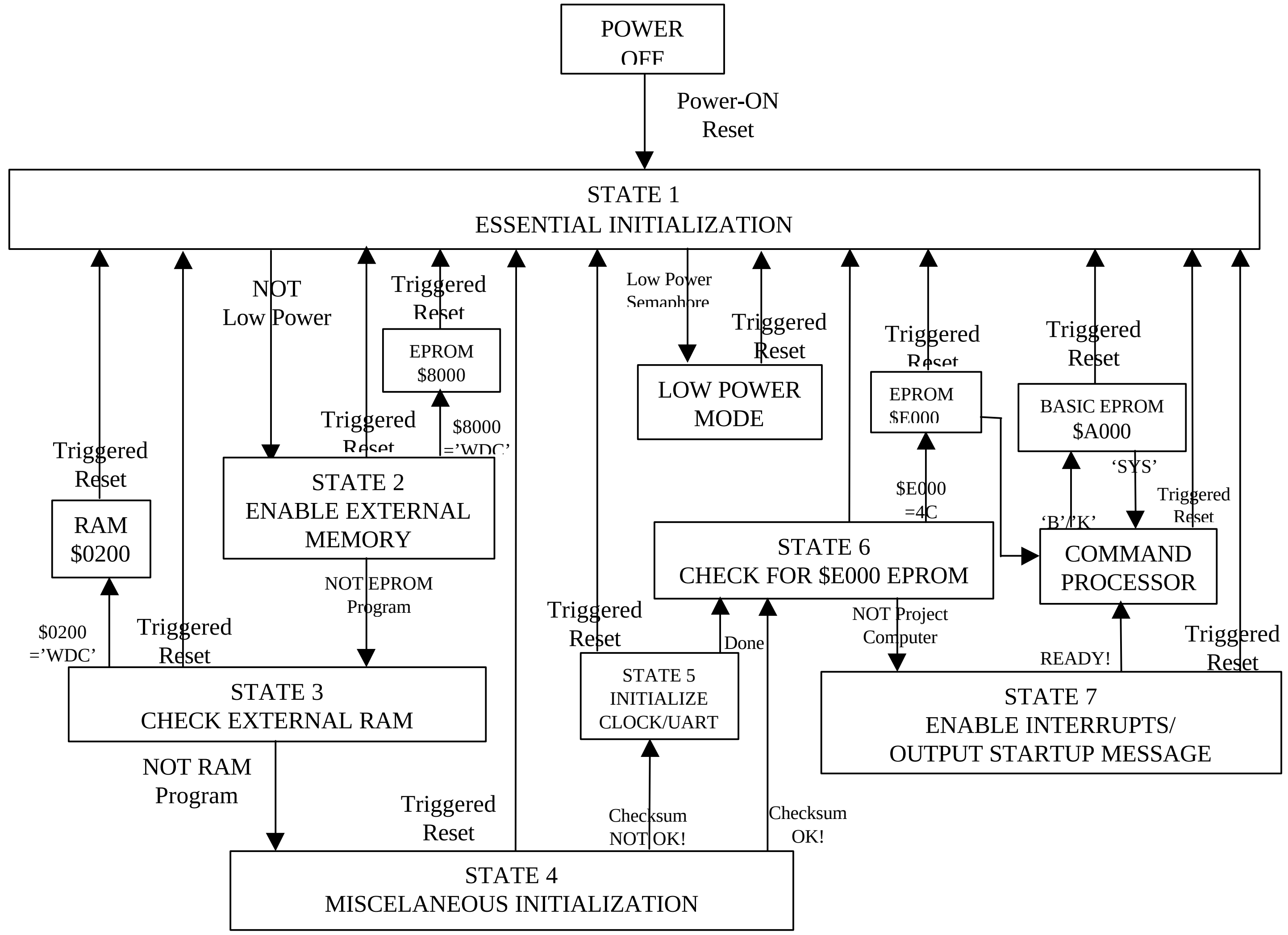
  Else proceed to STATE #7.

STATE #7 - "Enable interrupts/Output startup message"

- Read the serial port to clear any initial trash data.
- Enable interrupts. (CLEAR THE I BIT)
- Output the initial message to the serial port.[6]
- Execute a BRK instruction, which transfers control to the command interpreter.

[5] Com Log uses the EPROM addresses $E000-$EFFF in the Project Computer configuration for code which is specific to this product. Other developers, with different configurations, may use this memory for other purposes. The W65C134 monitor will still test location $E000 for $4C, and vector accordingly.

[6] Small initial buffers and a handshake line held in the FALSE state may cause the monitor to hang here waiting for buffer space that will not be available until the handshake line goes TRUE. Verify cables and handshake signals are correct.


POWER OFF
Power-ON Reset
STATE 1 ESSENTIAL INITIALIZATION
NOT Low Power
Triggered Reset
EPROM $8000
$8000 ='WDC'
Low Power Semaphore
Triggered Reset
LOW POWER MODE
Triggered Reset
EPROM $E000
Triggered Reset
BASIC EPROM $A000
'SYS'
Triggered Reset
'B'/'K'
Triggered Reset
RAM $0200
Triggered Reset
STATE 2 ENABLE EXTERNAL MEMORY
$E000 =4C
STATE 6 CHECK FOR $E000 EPROM
COMMAND PROCESSOR
NOT EPROM Program
$0200 ='WDC'
Triggered Reset
Triggered Reset
Done
NOT Project Computer
READY!
Triggered Reset
STATE 3 CHECK EXTERNAL RAM
STATE 5 INITIALIZE CLOCK/UART
STATE 7 ENABLE INTERRUPTS/ OUTPUT STARTUP MESSAGE
NOT RAM Program
Triggered Reset
Checksum NOT OK!
Checksum OK!
STATE 4 MISCELANEOUS INITIALIZATION

Low-Power Mode

The W65C134 monitor enters low-power mode when a "power-down" interrupt occurs. It accomplishes this by performing the following sequence:

- Shut down all interrupts (except TOD clock).
- Clear any pending interrupts.
- Reset the stack to FF.
- Write a semaphore series into RAM to indicate that the system is powered down.[7]
- Enable the power down routine.
- Switch to the slow (default) clock and then shut off the fast clock.
- Configure all I/O ports to inputs.

The power down routine will service the time-of-day interrupt. It will then check RAM for a semaphore series at location: $7E. If there is a valid power down routine in RAM, then locations: $7E = $55, and $7F = $AA. The RAM routine must begin at location: $0088. If the semaphore indicates that a low power service routine exists in RAM, the monitor will transfer control via a JSR to location: $0088 once per second.

The RAM routine may do anything appropriate to the low-power mode. Bus operations are not allowed. RAM locations: $40 - $87 retain meaningful data during low-power mode, and RAM locations: $F8-$FF are used for the stack when servicing the time-of-day interrupt. The RAM service routine should not modify these memory locations. Finally, it must return to the monitor, when finished. It must execute an RTS instruction.

After handling the RAM routine, if present, the monitor's power down interrupt service routine will determine whether or not power has been restored. If power has returned to the system, it will initialize/restore a few key registers and then jump to RESET.

If a physical reset occurs while the system is in low-power mode, this must be detected by the reset initialization sequence. It must return to the power down code without restarting anything on the bus. The semaphore series to flag this condition is in locations $7B = $55, $7C = $AA, $7D = $88. These locations are cleared to zero when not in low-power mode.

[7] The semaphore series for low power mode: locations $7B = $55, $7C = $AA, $7D = $88. These locations are cleared to zero when not in low-power mode.

User Console Operation

The monitor is entered upon power-up, reset, and when a BRK instruction is encountered. When the monitor is ready for a command, a '.' (period) will appear as a prompt. On reset, the monitor sends copyright and version notices, as well as a register display to the terminal. (There are semaphores in memory to enable/disable specific features.)

Command Entry

Commands are entered after the period prompt. The W65C134 monitor parses a command as it is entered. Spaces and other separators are automatically generated by the monitor as parsing dictates their appearance. No backspaces are allowed. Buffer limitations allow no provision for editing commands. If an error is made in entry, a carriage return (CR) or ENTER will usually cancel the command and return the monitor prompt. If a command has been started, a control-C character will usually cancel the command.

The first command to learn on the W65C134 monitor is the HELP command. Enter a 'H' at the command prompt (no return is needed) and a help menu will be displayed. This menu simply lists all the monitor commands.

Debugging

WARNING!

Generally, if code goes wild, execution will eventually encounter a BRK instruction and return to the monitor. Interrupts may however, still be running. A bad interrupt can disable the monitor functions.

Code debugging may be accomplished by placing BRK (00) instructions in the code. When the BRK instruction is executed, the CPU registers will be displayed and the monitor prompt will appear. The user may examine or change memory and registers. BRK instructions should be used with care. When a BRK instruction has occurred, the program counter (PC) in the monitor will be pointing at the location *AFTER* the BRK instruction.

If a BRK instruction was placed in the code by an assembler, and the next instruction follows the BRK, then the user may continue execution simply by hitting 'G' followed by a carriage return (CR).

If a normal instruction has been replaced with a BRK, the program may not properly resume until the original instruction has been restored and the PC register or the 'G' command adjusted accordingly.

Command Descriptions

The following pages describe all of the monitor commands in detail. They are listed here, in summary, along with their functions.

Command Summary

Command	Usage
A	ALTER registers
B	BASIC interpreter
C	Generate CHECKSUM of memory block
D	DISPLAY memory block
E	Display serial load ERRORS
F	FILL memory block with constant
G	GO to address (JMP execution)
H or ?	HELP
J	JUMP to subroutine (JSR execution)
K	Re-enter BASIC interpreter
M	Examine/Change MEMORY
R	Display REGISTERS
S	Download 'S' record format
T	Display current date and TIME
U	USER command prefix
V	Move a block of memory
W	WRITE block of memory as 'S19' records
X	Toggle X-On/X-Off handshake mode
/	Display memory
>	Display NEXT memory location
<	Display PREVIOUS memory location
(space)	Display current memory location
^C	Cancel current operation

(NOTE: The W65C134 monitor outputs are shown on the following pages in normal font, and user inputs are shown in *italics*.)

A, ALTER

This command will display the registers and allow the user to change them.

An executing program will re-enter the W65C134 monitor when it encounters a BRK instruction. The monitor will copy all the registers inside the processor into RAM. These copies of the registers may be viewed using the 'A' or 'R' commands. The 'R' command only allows viewing. The 'A' command permits the user to enter new values.

When a program is started via the 'J' or 'G' commands, the RAM versions of the registers are re-copied into the microprocessor before control is transferred. Changing the registers via the 'A' command only changes the RAM copy. This will have no effect until a program resumes.

Example:

```
.A
 PC    F   A   X  Y   SP
0000 00 00 00 00 00
1234 30 55 33 22 44
.
```

The W65C134 monitor will output the current values of the registers to the console: **PC**=Program Counter, **F**=Flag register, **A**=Accumulator, **X**=X register, **Y**=Y register and **SP**=Stack Pointer. The cursor must be placed under each entry, in order, for the user to change the contents. The monitor will insert spaces automatically between fields. A space entered from the console will cause the current field to be skipped, leaving it unchanged.

Spaces were entered at the A, Y and SP locations. These values were not altered.

Example:

```
.A
 PC    F   A   X  Y   SP
1234 30 55 33 22 44
3433 10    88
.
```

B, BASIC

The 'B' command initializes and starts BASIC. If BASIC is not present, the monitor prompt is returned. (The monitor checks for BASIC by looking for a $4C at address $A000, if present it does a JMP $A000. If the $4C is not present, the command returns to the monitor prompt.

Example:

```
.B

   (c) COPYRIGHT 1988,89
   COM LOG CO. INC.
BASIC VERSION 01.02 03-30-89 15:37
  20478 BYTES

READY
SYS   (This returns control to the monitor.)


ADDR F   A   X  Y   SP
A2B3 B1 AB 10 08 FC

.
```

C, Checksum

The 'C' command generates a checksum of the memory from the *starting address* to the *ending address*.

Example:

```
.C 1000 1FFF
12FD
.
```

The user enters a '*C*', followed by '*1000*', followed by '*1FFF*'. No spaces are entered by the user. The monitor will generate a 16 bit checksum by simply adding all bytes in the specified range together. Checksum = ($1000)+($1001)...+($1FFF). Only the lower sixteen bits of the checksum are used. If the checksum exceeds sixteen bits (i.e. Checksum is greater than $FFFF.), the overflow is ignored.

This command is useful to insure that a program loaded into RAM has not been accidentally altered.

D, Display Memory

This monitor command displays a range of memory, in hex and ASCII. Parsing is performed as the command is entered. The user types no spaces and no return or enter key.

Example:

```
.D 1000 1FFF
ADDR 00 01 02 ...  0D 0E 0F       ASCII
1000  55 AA 88 ... 41 42 43    U*. ... ABC
.
.
.
1FF0  22 33 44 ...  64 65 66    "3D ... def
.
```

A shorter form involves entering the '*D*', the *starting address*, and a *space*. The space means 'Skip this operand', as it does on the other commands, and the display command will display 16 bytes starting at the given starting address. A *space* is unacceptable as the *starting address*.

E, Errors

This command is only valid after a serial load. The command displays the number of errors, the address of the last error, and then resets the number of errors to 0. It is often a good idea to use the E command before a serial load to clear the errors, then after the load to insure that no errors rolled off the screen while the user wasn't looking.

Example:

```
.E 005A  00    (Clear the errors.)

.E 0000  00    (Prove they're clear.)

.S             (Load some 'S' records.)
.S
.S
.S

.E 0000  00    (Load errors = None.)

.
```

F, Fill

Example:

```
.F 4000 5FFF 22
.D 4000 400F      (Confirm filled.)
ADDR 00 01 02 03 04 05 06 ... 0D 0E 0F
4000 22 22 22 22 22 22 22 ... 22 22 22
.
```

This command is used to fill a block of memory with a particular *value*.

No spaces are entered by the user. This command will fill memory from $4000 through $5FFF, inclusive, with the value hexadecimal 22.

G, Go

This command restarts a program stopped by a BRK instruction. The program resumes one byte after the BRK instruction itself. The registers used to restart the processor are the RAM copies made when the program was stopped. (See the **A, Alter** command for more details.)

Example:

```
.G<return>
.

or

.G 2000
.
```

In this case, the *<return>* is necessary to differentiate the forms of the command. The first form uses all the register values stored in RAM. The second form uses the supplied address for the PC, effectively restarting from the address given. The second form does not need the *<return>*.

H or ?, Help

The HELP command lists all the commands available via the monitor. It is a single character, entered at the prompt. No <return> or <space> is necessary.

Examples:

```
.H

D       Display memory
SPACE   Display current memory address
<,>     Decrement, Increment memory address
M       Alter memory
/       Host memory access
R,A     Display, Alter registers
G,J     JMP, JSR to PC [Location]
F,V,C   Block Fill, Move, Checksum
S,W,E   S28 Input, Output, Errors
?,H     Help
B,K     BASIC start, Continue
T       Display time
X       Toggle handshake mode
U       User installed commands

.
```

or

```
.?

D       Display memory
SPACE   Display current memory address
<,>     Decrement, Increment memory address
M       Alter memory
/       Host memory access
R,A     Display, Alter registers
G,J     JMP, JSR to PC [Location]
F,V,C   Block Fill, Move, Checksum
S,W,E   S28 Input, Output, Errors
?,H     Help
B,K     BASIC start, Continue
T       Display time
X       Toggle handshake mode
U       User installed commands

.
```

J, Jump to subroutine

This command is used like the 'G' command, excepting that the address of the BRK handler is put on the stack before starting the program. If the program ends with an RTS, it will return cleanly to the monitor.

Examples:

```
.J<return>
.

or

.J 5000
```

The first example uses the old PC address stored in the RAM copies of the registers, the second executes from the given address. Excepting the **SP** (which is bumped for the return address) and the PC if the long form is used, the registers are copied from the RAM into the processor before the jump takes place. (See *A*, *Alter* for more details.)

K, Restart BASIC

This command would normally be used only if Com Log BASIC has been installed in the system. The user may choose to exit BASIC and return to the W65C134 monitor after a program has been entered[8]. Eventually, the user may desire to restart Com Log BASIC and continue with the BASIC program previously entered.

The 'B' command initializes BASIC, clearing anything currently in RAM. The 'K' command allows users to return to BASIC at the point they left, without erasing the BASIC program currently in RAM, but it does not make a full integrity check of the BASIC environment. If the user has been making changes in BASIC RAM locations from the monitor, the 'K' command may be inoperable or produce unpredictable results.

Example:

```
.K

    (c) COPYRIGHT 1988,89
     COM LOG CO. INC.
BASIC VERSION 01.02 03-30-89 15:37
 20478 BYTES

READY
```

[8] Com Log BASIC users may enter the monitor via the SYS command.

M, Change Memory

This command is used to change the contents of memory locations, beginning at the specified address.

Example:

```
.M 1000

ADDR   00 01 02 03 04 ... 0D 0E 0F
1000   0A 4C 88 19 33 ... 32 54 A7
1000   AA CC 22 55 44 ... 88 6F 3D
.
```

The user must enter: '*M*' and an *address*. The monitor will respond by printing a legend and the current contents of the first 16 locations. The cursor is then positioned one line below the first memory location. The user may enter a new value for that location. If the user continues typing new values, and they modify consecutive locations. Parsing is performed as the command is typed. The monitor will automatically output spaces between fields. A space may be entered by the user to skip a location. That byte will be left at its original value.

R, Registers

This command displays the current RAM copy of the microprocessor registers.[9]

Example:

```
.A
 PC  F  A  X  Y  SP
0000 00 00 00 00 00
1234 30 55 33 22 44

.R
 PC  F  A  X  Y  SP
1234 30 55 33 22 44

.
```

The W65C134 monitor will output the current values of the registers to the console: **PC**=Program Counter, **F**=Flag register, **A**=Accumulator, **X**=X register, **Y**=Y register and **SP**=Stack Pointer.

[9] An executing program will re-enter the W65C134 monitor when it encounters a BRK instruction. The monitor will copy all the registers inside the processor into RAM. These copies of the registers may be viewed using the ‘A’ or ‘R’ commands. The ‘R’ command only allows viewing. The ‘A’ command permits the user to enter new values.

S28 Format:

S2LLHHMMLWDDDDDDDD...CC

where:

S2 are literally the ASCII characters:"S2",
LL is the length of the data+4,
HH is the high byte of the address,
MM is the middle byte of the address,
LW is the low byte of the address,
DD is one byte of data,
DD is the next byte, etc
CC is the checksum (1's complement of the sum of the length, address, and data bytes.)

NOTE: Convert the bytes from ASCII to hex before performing the addition.

A valid S28 record is:

```
S2140090004C28BA4C629D0102031589E6F3D002E6AD
```

The end record, starting with S8... is ignored by the W65C134 monitor.

S19 format:

S1LLHHLWDDDDDDDD...CC

where:

S1 are literally the ASCII characters:"S1",
LL is the length of the data+3,
HH is the high byte of the address,
LW is the low byte of the address,
DD is one byte of data,
DD is the next byte, etc
CC is the 1's complement of the sum of the length, address, and data bytes.

NOTE: Convert the bytes from ASCII to hex before performing the addition.

A valid S19 record is:

```
S113800057444300A981851B4C57F0000000000031
```

The end record, starting with S9 ... is ignored by the W65C134 monitor.

S, S record input

The W65C134 monitor uses Motorola style 'S' records to upload or download programs. Acceptable formats are: S19 and S28.[10] These formats comprise lines of data, each starting with an 'S' and ending with a <return>.

When the monitor receives an 'S' as the first character of a command, it assumes that the remainder of the line is an 'S' record. It reads the remainder of the line data into memory as it is received. The user typically will not type 'S' records as commands. They are usually sent automatically by the host communications program as a "file transfer/interchange" function. When the S is detected, the W65C134 monitor turns off the serial port echo. This means the S record is not echoed back to the sending device. When the CR is found, the echo is turned back on. If 'S' records are being loaded, the user will see only a single 'S' on each line as it is loaded.

Parsing is dynamic. If the line contains a non-hex character, or if the ending checksum is wrong, the error count will be incremented. (See the **E, Error** command for more details.) When this occurs, any previous memory modifications have already been made. If a program download has errors, the memory block has been corrupted. Execution of the program may produce unpredictable results.

[10] The Motorola S19 format is almost identical to the S28 format, excepting the address field is 16, not 24 bits, and the length of each record is one byte less.

T, Time

The T command prints the current time.

It is in the format:

W MM/DD/19YY HH:MM:SS

where:

W = the day of the week (1 = Sunday, 7 = Saturday).
MM is the month,
DD is the day,
YY is the year,
HH is the hour (24 hour format)
MM is the minute,
SS is the second.

Setting the time may be accomplished through the 'T' command or by setting the memory locations directly.

Example:

```
.T

3 03/09/1993 13:05:03

.
```

U, User Command

This is a mechanism by which the user can easily add his own commands. When the monitor prompt is present, characters are parsed by the W65C134 monitor as they are entered. If the first character of a command is a 'U' or 'u', the monitor will perform a JMP (UCMDPTR). Additional characters entered after the 'U' must be handled by the user routine. When the user's routine is finished, an RTS instruction will return control to the monitor.

The user must supply a routine to handle the characters after the 'U', and then store the address of that routine at location UCMDPTR ($0050). The user's command processor should return to the monitor via an RTS instruction.

V, Block Move

This command is used to move a block of memory.

Example:

```
.V 1000 5000 22
.
```

The first operand is the source address, the second address is the destination, and the third byte is the number of bytes to move. The maximum number of bytes is 256, set by making the third operand a 0.

W, S19 output

This command is used to output S19 records via the serial port.

Example:

```
.W 1000 2FFF<return>

.
```

The first address is the starting address, the second address is the address of the last byte to be sent. This command specifically requires a return, allowing the user to turn on a recording device to record the serial data.

S19 format:

S1LLHHLWDDDDDDDDD...CC

where:

S1 is literally the ASCII characters S then 1,
LL is the length of the data+3,
HH is the high byte of the address,
LW is the low byte of the address,
DD is one byte of data,
DD is the next byte, etc
CC is the 1's complement of the sum of the length, address, and data bytes.

NOTE: The checksum addition is performed before the data bytes are converted to ASCII characters.

A valid S19 record is:

```
S113800057444300A981851B4C57F0000000000031
```

The end record, starting with S9 is automatically generated by the W65C134 monitor.

X, Set Handshake Mode

This mode toggles between hardware handshake and XON-XOFF handshaking. The RESET default is hardware.

The command responds with the new value of the handshake.
(00 = Hardware, and 01 = XON-XOFF.)

Example:

```
.X
01
.
```

The above example changed the handshake from hardware to XON-XOFF.

Example:

```
.X
00
.
```

This example changed the handshake from XON-XOFF to hardware. There is no command to view the current state without changing it.

Special Memory Commands: /,<,>,SPACE

The SPACE, <, >, allow the user to view the current memory address, one lower, or one higher, respectively. The current address is set via the M or the D command.

The SLASH (/) command is intended to allow a program on a host system quick access to W65C134 memory locations. Its operation is documented in the software listings.

Programming

The Com Log Company, Inc. offers an extended monitor for the W65C134 mapped into external EPROM. It uses memory addresses: $E000 through $EFFF. This portion of the monitor provides display and button control for the **Project Computer** configuration. It also provides a number of software timers and clocks for general purpose use. It has a JMP table at $E000 to provide consistent access to routines even if they are relocated in later revisions.

An EPROM version of BASIC is also available from Com Log. It is sold as an option for the Project Computer. If installed, it occupies EPROM addresses from $A000 to $DFFF.

The $E000 monitor software comes standard with the **Project Computer**. Since the **Project Computer** has sockets for only one ROM, it is shipped with a 27C256 EPROM. This is usually pre-programmed from $E000 to $FFFF. If the BASIC option has been included, the EPROM will also be pre-programmed from $A000 to $DFFF. This leaves the memory in range: $8000 through $9FFF available for user programs.

The monitor is masked into the W65C134's internal ROM, and resides between $F000 and $FFFF. This portion of the monitor handles the serial port and provides a simple command line interface. The command line functions include program upload and download, memory display, alter, fill, move, and checksum. There are additional functions for dealing with the W65C134's internal registers, the real time clock, and the serial handshaking. The command line interface runs on the serial port.

This firmware also handles all interrupts (since the vectors are in its screened memory). In most cases, interrupts are re-vectored through RAM to the user's routine. In some cases, the RAM vectors are initialized to point back into the core ROM. The internal ROM has routines for buffering the serial ports, for updating the time of day clock, and so forth. The user may handle these tasks with custom software. The only change necessary involves re-writing the vectors in RAM. All routines in the masked ROM are accessed via a JMP table at $F000.

A flag in the external EPROM is used to indicate that the user wants complete control of the PCB and processor. It is checked immediately after reset. If the flag is present, the internal ROM jumps into the EPROM before setting any interrupt vectors or internal registers. The user's code can then turn off the internal ROM. All accesses, including accesses to interrupt vectors, are now pulled from the external EPROM. In this case, none of the monitor's routines are accessible to the user. There some semaphores from $8000 to $8010 which are used to indicate the presence of the ROM to the core monitor. The remainder of the EPROM ($8010 to $DFFF) is available to the user.

Monitor Features/Functions

Example:

```
            CHIP  6502
            LONGA OFF
            LONGI OFF
            SPACES ON
;
; This program sends the phrase 'Hello, world!' to the serial port.
;
            INCLUDE FIRMWARE.H    ;Firmware equates

            .ORG        $0800                    ;A good test location
                                                 ; to start.

            JSR         CRLF                     ;Get to the next line
                                                 ; (send CR/LF characters)

            LDA         #>HSTRING  ;High order address
                                                 ;to accumulator.

            LDX         #<HSTRING  ;Low order address to X

            LDY         #ESTRING-HSTRING ;Number of characters.

            JSR         PRTSTR                   ;Print it.

            BRK                                  ;Return to monitor.

HSTRING  .BYTE  'Hello, world!'       ;Define the text.

       .BYTE  $0D                     ;A 'return' character.

ESTRING  EQU    *         ;Define the end of text.

S records for this program are:
            S2140008002012F0A908A20DA00E201EF00048656C6C
S20F0008106C6F2C20776F726C64210D5B
S804000000FB
```

The W65C134's hardware and firmware can be used for a variety of applications. There are numerous routines in the firmware designed to handle the hardware and several common interrupt tasks. Most applications could use some of these routines.[11]

The simplest way to explain how these routines work is by example.[12] The examples shown in this manual are intended to illustrate. They are not necessarily complete. (Error returns are often ignored.) This code was written for clarity rather than speed or size.

'Hello, world!'

This (right) is one of the most universal and simplest examples of how to start programming on a new system. The purpose of this example is to try to get the program edited, assembled, linked, into the W65C134, and run. The exact operation of the program is secondary.

This example was written, assembled, and linked on an IBM AT using 2500AD's W65C816 cross assembler and linker. The result of the assembly and link is a file consisting of the absolute object code in Motorola S28 format.[13]

[11] The subroutines which are described in detail in the manual have ROM vectors at fixed addresses. These vectors will not move as new revisions of the monitor become available. Other subroutines may be found in Appendix C – W65C134 Monitor Assembly Listing. They may be accessed directly. The user should be aware that such subroutines may be modified, relocated, or removed from later versions of the W65C134 ROM Monitor. Refer to the description of the VERSION subroutine ($F000) as a means of checking for compatibility.

[12] The W65C134 ROM Monitor source code is included in this manual. There is a cross reference at the end of each portion of the source which can be used to find specific sections.

[13] Details of the S19 and S28 formats may be found in Commands: S, S record input and under the routine descriptions for ploading and downloading ‘S’ records.

Interrupts

The W65C134 supports many interrupts, each with its own vector. Definitions of the interrupts are found in the Western Design Center W65C134 Data Specification.

The W65C134 ROM monitor firmware uses some of the interrupts. Some vectors point directly into the W65C134's internal ROM. Others point to their own RAM locations which may be changed by the user. The majority, mostly edge interrupts, point to a common RAM vector which may also be changed by the user.[14] (See Table for more details)

Interrupts on the W65C134 must be enabled in either the interrupt enable registers, the serial control port, or the timer control register. Enabling, disabling, and resetting interrupts is covered in the WDC W65C134 data specification.

Many interrupts go to a location in ROM which is a JMP(XXXX). This just gets us to a RAM vector for the interrupt. The NE46 (used for power down) interrupt vector points into the internal ROM. It points to the following instruction: **JMP ($0046)**

This approach adds a few cycles to the interrupt servicing overhead time, but allows simple byte vectors in the *limited* internal RAM. The location $0046 contains a RAM pointer (initialized after RESET) to the monitor's power down routine.

The monitor handles all interrupts. This is often done by having a second vector in RAM. When the monitor is started, it writes the RAM locations with pointers to its own interrupt routines. Unused interrupts simply jump to RESET. The user can re-vector these interrupts by changing the RAM vector.

[14] Com Log choose this approach as a design decision based upon resource limitations (i.e. available RAM vectors) in the minimum configuration to be supported. The effect of this is that interrupts sharing the common vector require an interrupt service routine that must be able to identify which interrupt was received. See the code for more details.

Interrupt Vectoring

Interrupt	Interrupt	Monitor	Status after
Source	Address	ROM Vector	Initialization
PE44	$FFD0	JMP($004E)	DISABLED
PE45	$FFD2	JMP($004E)	DISABLED
NE46	$FFD4	JMP($004C)	ENABLED[1]
NE47	$FFD6	NE47 (in ROM)	ENABLED[2,3]
PE50	$FFD8	JMP($004E)	DISABLED
PE51	$FFDA	JMP($004E)	DISABLED
NE52	$FFDC	JMP($004E)	DISABLED
NE53	$FFDE	JMP($004E)	DISABLED
IRQAT	$FFE4	Serial TX (ROM)	ENABLED[2,4]
IRQAR	$FFE6	Serial RX (ROM)	ENABLED[2]
PE54	$FFEA	JMP($004E)	DISABLED
PE55	$FFEC	JMP($004E)	DISABLED
PE56	$FFEE	JMP($004E)	DISABLED
NE57	$FFF0	JMP($004E)	DISABLED
IRQT1	$FFF2	JMP($004A)	ENABLED[5]
IRQT2	$FFF4	JMP($0048)	ENABLED[6]
IRQ1	$FFF6	JMP($0046)	DISABLED
IRQ2	$FFF8	JMP($0044)	DISABLED
NMI	$FFFA	JMP($0042)	DISABLED
RESET	$FFFC	RESET	ENABLED[2,7]
BREAK	$FFFE	JMP($0040)	ENABLED

Initialization after RESET Sequence:

$004E, $0046, $0044, and $0042 are initialized to RESET
$004C is initialized to PDOWN
$004A is initialized to CNTRIRQ
$0048 is initialized to TODIRQE[8]

[1] This interrupt is used for power down detection.

[2] These interrupt vectors point directly into the ROM and cannot be intercepted.

[3] This interrupt is used for serial handshaking. (The interrupt occurs when the handshake input to the W65C134 goes low.)

[4] The transmit interrupt is enabled when there is data to send and disabled otherwise.

[5] This interrupt is used to provide a 10 mSec system heartbeat. It is used only in the $E000-$EFFF portion of the firmware, which is in turn used by BASIC.

[6] This interrupt is used for the 1 interrupt per second time of day clock.

[7] Reset is ALWAYS enabled.

[8] This is a different Time-of-Day routine than the one used in the $F000 code.

Low-Power Mode

The W65C134 uses CMOS circuitry. Power consumption is therefore a function of how many devices are switching and how often they switch. Applications may reduce power consumption by switching to the slower default clock (32 KHz). Further power reductions may be achieved by using instructions such as STP (stop) and WAI (wait) which halt the processor until an interrupt occurs.

Dramatic reductions in power may involve shutting off all circuitry other than the microprocessor. This automatically occurs when incoming power is lost. All devices excepting the W65C134 chip lose power.[15] An interrupt will tell the monitor when main power is going down. This interrupt invokes a routine in the internal ROM of the microprocessor which shuts off the fast clock, the address and data bus, and sets all external lines low. The only external sign of operation will be that the slow clock is still running. All interrupts will be disabled, excepting the time-of-day interrupt. When the time-of-day interrupt occurs, the clock will be updated and internal RAM checked for a flag which indicates that a user's program has been loaded into the on-board RAM. If present, the user's program is run after every time-of-day interrupt (once per second). The on-board RAM limits the user's low-power program size to approximately 100 bytes.

Special Note

The W65C134 ROM monitor depends upon an interrupt to notify it of power going down. This assumes that the application configuration has capacitors on the power supply which will provide enough power to execute the power down routine. If a user program disables the interrupt, or has overly long interrupts, then the power down code may not run, and the application system will not correctly power down. This could result in a very rapid discharge of the battery intended for low-power mode.

Low-power mode essentially allows the user to write a small program which runs when main power is off. This program could check the I/O ports (looking for special conditions), or check the alarm flag. (The time-of-day clock includes an alarm function which sets a flag when the current time equals the alarm time previously set.) This user's routine could then decide it is time to re-power the entire system, doing so by raising an I/O bit to turn on external power.

Low-power mode will only work if the W65C134 is running an *internal* program. During low-power mode, the monitor shuts off the external bus and therefore does not have access to the external EPROM or the external RAM. The monitor and it's low-power support routines reside in the internal mask ROM of the W65C134 chip.

Refer to **Appendix C - W65C134 Monitor Assembly Listing** for details of the monitor's support code for low-power mode.

[15] This assumes that in the application, the W65C134 chip has some form of battery backup. Otherwise, this section on low-power mode is irrelevant.

Serial I/O Support

The W65C134 ROM monitor provides a user console via the serial UART as a fundamental feature. This allows the user to examine and change memory or registers, load, debug and save programs, and even interact with the Time-of-Day clock. The firmware involved in the implementation of these functions includes many useful subroutines.

There are several serial I/O support subroutines listed in the table (right). They allow the user to take advantage of the console features in an application program. The developer may use some of these features may be used merely for debugging. Others could become a significant part of the target system.

Serial I/O Support

Routine	Vector	Description
ACI_INIT	$F003	Initialize ACIA.
CK_CONTC	$F009	Check for ^C pressed.
CRLF	$F012	Print a Carriage Return/LineFeed sequence.
GETCH	$F00C	Get a character w/wait.
MS19OUT	$F042	Output 'S19' records.
MS28IN	$F045	Input 'S' records.
OUTCH	$F00F	Write a character.
PRTSTR	$F01E	Print a String.
RD_CHAR	$F006	Read a character (if present).
RDOA	$F021	Read an address value in ASCII HEX format.
RDOB	$F024	Read a Byte value in ASCII HEX format.
SPAC	$F015	Print a space.
WR_ADDR	$F027	Write an address in ASCII HEX format.
WROB	$F02A	Write a Byte in ASCII HEX format.

Complete, detailed descriptions of the vectored serial I/O support subroutines are provided elsewhere in this manual. Refer to: **Monitor Library Subroutines** for more information.

Time-of-Day Clock Support

The Time-of-Day clock is probably the most versatile and useful feature of the W65C134 monitor. It provides a "real-world" reference to programs, allows the programmable alarm function to operate, and continues to run even in "low-power" mode. Several monitor subroutines support the Time-of-Day clock and alarm access by user programs. Most of these subroutines expect or return day, date and time data as a packet in the format shown (right).

Descriptions of the specific support subroutines for the Time-of-Day clock are provided in: **Monitor Library Subroutines**. Refer to that section of this manual for more information.

Time-of-Day Clock Format

Byte	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

NOTE: All values must be in 2's complement format.

Programmable Alarm

The Time-of-Day clock functions include a programmable alarm feature. The alarm is initialized with day, date and time values. These settings are compared each second to the current value of the Time-of-Day clock. The monitor will set a flag (bit #4 of location: $0077) when an acceptable match occurs. This alarm flag will be cleared to zero by the reset initialization sequence. It may set, reset, or checked by user application software.[16]

Time-of-Day Clock / Alarm Support

Routine	Vector	Description
RD_CLOCK	$F04B	Read Time-of-Day Clock.
RTC_MODE	$F051	Set/Clear Daylight Savings Mode.
WR_CLOCK	$F04E	Write Time-of-Day Clock.
WR_ACLOCK	$F054	Program the alarm.

[16] The W65C134 ROM Monitor does not use the programmable alarm feature. This is entirely available to the user application software.

The application software should use the WR_ACLOCK subroutine to initialize the alarm. The subroutine will load values into the seven parameters of the alarm data. All seven of the parameters must be initialized together. This involves a packet of data identical in format to those used by the Time-of-Day clock. The primary difference involves "don't care" fields. If any parameter is set to $FF, it will be ignored.

Programmable Alarm Clock Format

Byte	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

NOTE: All values must be in 2's complement format.

Programming the alarm criteria does not enable the alarm. The alarm function must be enabled by setting a flag bit in memory (bit #3 of location: $0077). Disabling the alarm will reduce the overhead time of the Time-of-Day interrupt handler. The alarm function is disabled (bit #3 = 0) by the reset initialization sequence.

The alarm clock function may be programmed to trigger at 1:00 PM on Saturdays, using the following sequence:

```
00 00 0D FF FF FF 07
```

This specifies zero seconds, zero minutes, 13 hours, any day, any month, any year, and the 7th day of the week (Sun = 1, Sat = 7).

The user should exercise care when programming the alarm function. The "wild card" patterns ($FF) will match anything. The alarm may be programmed for Jan 3, 1995 by using the sequence:

```
FF FF FF 03 01 5F FF
```

This pattern allows: any time on the 3rd of January, 1995 (any day). An alarm will occur every second all day long. Note also that if you want a specific time and date, you should set the day of the week to $FF to avoid a mis-match.

When an acceptable match occurs, the clock interrupt will set the alarm flag. The flag is bit #4 of location $0077. A '1' in that bit indicates that the alarm has triggered. The flag stays set until the user program resets it.

Data Manipulation

Several common data checking and manipulation functions are used by the W65C134 monitor internally. All are available to the application software. Some of these functions convert data from one format to another. Others merely check data using pre-defined criteria. The MV_DATA subroutine will move a block of bytes from one area of memory to another (RAM) area. Those routines listed in the table (right) are described in detail elsewhere. These routines all have vectors at the base of the W65C134 internal ROM.

Data Checking/Manipulation Functions

Routine	Vector	Description
ASCBIN	$F018	ASCII to Binary conversion (2:1).
BINASC	$F01B	Binary to ASCII conversion.
BIN2DEC	$F03F	Binary to Decimal conversion.
CHK_SUM	$F048	Calculate Checksum.
HEXIN	$F03C	Converts ASCII HEX to binary (1:1).
IFASC	$F030	Check for ASCII.
ISUM	$F02D	Check for ASCII decimal digit.
MV_DATA	$F036	Move Data block.
UPPER_CASE	$F033	Convert character to Upper-Case ASCII.

There are additional subroutines which may be useful to the applications developer, but do not have vectors. These subroutines may be found in **Appendix C - W65C134 Monitor Assembly Listing**. They may be accessed directly, but the user should take special care. Some are used by the other functions listed, and could possibly cause conflicts.

Non-vectored subroutines may be modified, relocated, or even deleted in later versions of the W65C134 Monitor ROM.[17] Users should refer to the description of the **VERSION** subroutine ($F000) as a means of checking for compatibility.

[17] The W65C134 Monitor ROM has remained unchanged since 03-07-89. Future changes are expected to maintain the integrity of the vectored functions. Simple, useful internal routines may be relocated as a result of additions or deletions but will probably not be removed. Application software, which uses direct access, may require re-assembly.

Special Functions

The W65C134 monitor includes some special functions which are also available to the application software via vectors. These functions are listed in the table (below), and are explained elsewhere.

Special Functions

Routine	Vector	Description
EXTRESET	$F057	External Reset vector.
START	$F039	Return to monitor command prompt.
VERSION	$F000	Get monitor version.
WAIT	$F05A	WAIT instruction w/RTS.

START and **EXTRESET** are available for initialization/critical error situations wherein the user software cannot recover itself. The **WAIT** subroutine is simply a WAIT instruction followed by an RTS instruction. User software may use **WAIT** to implement delays or to suspend until the next interrupt occurs.[18] The **VERSION** subroutine returns information identifying the current version of the W65C134 monitor. This may allow the user to determine whether or not to access monitor subroutines directly.

Custom Commands

The system developer may choose to add or replace monitor commands. Commands may be added via the 'U' command or through an alternate parser.

When the user enters: 'U' from the console, the W65C134 monitor will JSR to a JMP indirect through a pointer. The "user command pointer", UCMDPTR (locations: $0050/51) must be initialized by the application program. The custom command processor/parser should return to the monitor via an RTS instruction. The monitor will start again and prompt for the next command input.

The alternate parser method allows the custom commands to begin with any letter, including those already used by the W65C134 monitor. Existing monitor commands may be disabled or replaced by this technique. The alternate parser must reside in external EPROM. The W65C134 monitor will prompt, accept a character, and check location: $EFFD. If a JMP ($4C) is present, the monitor will "JSR $EFFD", to execute the alternate parser first. (Refer to: Appendix C – W65C134 Monitor Assembly Listing for details.) Again the alternate parser should return to the monitor via an RTS instruction. The W65C134 monitor will resume, and parse the character in register-A[19]

[18] Application developers should remember that the Time-of-Day clock is operating. This will generate an interrupt once every second.

[19] If the character in register-A is LINEFEED ($0A), the monitor will ignore it and loop back to accept another character without issuing another prompt. If the character is simply not a monitor command, then the monitor will type: '?' and restart with a fresh prompt. Of course monitor commands will be processed as though they were received from the console.

Monitor Library Subroutines

The W65C134 has a large number of standard subroutines, with fixed JMP vectors, available to the user. The JMP table is in the mask ROM, and starts at address: $F000.

The easiest way to use these subroutines is to set up an equate table, as described in Appendix B – W65C134 Monitor Subroutines, to define their names. The application may then simply JSR to the appropriate subroutine name as needed. These firmware subroutines reduce the overhead of user applications, and also standardize the way common operations are performed.

W65C134 Monitor Library Routines

Routine	Vector	Description
VERSION	$F000	Get monitor version.
ACI_INIT	$F003	Initialize UART.
RD_CHAR	$F006	Read a character from UART (if present).
CK_CONTC	$F009	Check for ^C pressed.
GETCH	$F00C	Get a character from UART (wait as needed).
OUTCH	$F00F	Write a character to UART.
CRLF	$F012	Print a Carriage Return/LineFeed sequence to UART.
SPAC	$F015	Print a space to UART.
ASCBIN	$F018	ASCII to Binary conversion (2:1).
BINASC	$F01B	Binary to ASCII conversion.
PRTSTR	$F01E	Print a String to UART.
RDOA	$F021	Read an Address value from UART in ASCII HEX format.
RDOB	$F024	Read a Byte value from UART in ASCII HEX format.
WR_ADDR	$F027	Write an address to UART in ASCII HEX.
WROB	$F02A	Write a Byte value to UART in ASCII HEX.
ISUM	$F02D	Check for ASCII decimal digit character.
IFASC	$F030	Check for ASCII.
UPPER_CASE	$F033	Convert character to Upper-Case ASCII.
MVDATA	$F036	Move Data.
START	$F039	Return to monitor command prompt.
HEXIN	$F03C	Converts ASCII HEX to binary (1:1).
BIN2DEC	$F03F	Binary to Decimal conversion.
MS19OUT	$F042	Output 'S19' records.
MS28IN	$F045	Input 'S' records.
CHK_SUM	$F048	Calculate Checksum.
RD_CLOCK	$F04B	Read Time-of-Day Clock.
WR_CLOCK	$F04E	Write Time-of-Day Clock.
RTC_MODE	$F051	Set/Clear Daylight Savings Mode.
WR_ACLOCK	$F054	Program the alarm.
EXTRESET	$F057	External Reset vector.

ACI_INIT ($F003) Registers Expected: A,Y,X
Registers Modified: A,Y

Baud Rate Selection

A	Baud rate	A	Baud rate
0	75	6	1800
1	110	7	2400
2	150	8	4800
3	300	9	9600
4	600	A	19200
5	1200	B	38400

Register-X has data length: 7 = 7 bits
8 = 8 bits
Other = 7 bits

Register-Y enables and sets parity.
Bit 0 = 0 disables
Bit 0 = 1 enables
Bit 1 = 1 sets even parity.
Bit 1 = 0 sets odd parity.

This subroutine initializes the internal serial port and serial buffers. Set the baud rate number in register-A, data length number in register-X, and parity in register-Y. This subroutine uses the clock speed test, executed at reset,[20] to determine the correct baud rate. It works with a variety of CPU clocks: 2.000000 MHZ, 4.000000 MHZ, 2.457600 MHZ, 3.686400 MHZ, or 1.843200 MHZ). The **ACI_INIT** subroutine will flush any data currently in the serial communication buffers. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **ACI_INIT** subroutine.

Errors:

If the baud rate selection parameter (passed in register-A) is invalid, the **ACI_INIT** subroutine will return with: $FF in register-A. The previous baud rate will remain unchanged.

[20] The clock speed test is performed during the reset initialization sequence. It compares the fast clock to the default (32 KHz) clock. This test allows the W65C134 monitor to select appropriate values when using the baud rate generator and other timers.

Example:

```
LDY #$03         ;Parity on, even
LDX #$08         ;8 bit data
LDA #$0B         ;38.4K baud
JSR ACI_INIT
CMP #$FF
BEQ BAD_INIT ;Good baud rate, no branch
...
LDY #$00         ;Parity off
LDX #$07         ;7 bit data
LDA #$27         ;Invalid baud parameter
JSR ACI_INIT
CMP #$FF
BEQ BAD_INIT ;Bad baud - Branch!
...
```

ASCBIN ($F018) **Registers Expected:** A,X

Registers Modified: A

This subroutine converts two ASCII HEX[21] characters into a single binary byte. The ASCII hexadecimal character in register-A corresponds to the least significant nibble of the result. The most significant nibble is defined by the ASCII hexadecimal character in register-X. The resulting binary value will be returned in register-A. The carry-bit will be **CLEAR** upon a normal return from this subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **ASCBIN** subroutine.

Example:

```
LDX #'0'
LDA #'D'
JSR ASCBIN
CMP #$0D
BEQ OK           ;This should always branch
...
```

Errors:

The **ASCBIN** subroutine does not detect any errors or return error codes. If the calling program passes a parameter which is not an ASCII HEX digit, the resulting conversion value will be meaningless.

[21] A valid ASCII hexadecimal digit is a numeric character: “0123456789” ($2F < char < $3) or one of the first six letters: “ABCDEF” ($40 < char < $47) in uppercase or lowercase: “abcdef” ($61 < char < $7A).

BINASC **($F01B)** **Registers Expected:** A
Registers Modified: A,X

This subroutine converts an 8-bit binary value in register-A to two ASCII HEX characters. Register-A returns the least significant character in ASCII. Register-X returns the most significant character in ASCII. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **BINASC** subroutine.

Example:

```
LDA #$0D
JSR BINASC
CMP #'D'
BNE FAILED     ;Should never branch
...
CPX #'0'
BNE FAILED     ;Should never branch
...
```

Errors:

The **BINASC** subroutine does not detect any errors or return error codes

BIN2DEC **($F03F)** **Registers Expected:** A
Registers Modified: A

This subroutine takes an binary value ($00-$63) in register-A and converts it to packed decimal format ($00-$99) also in register-A. Values larger than $63 will not be properly converted. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **BIN2DEC** subroutine.

Example:

```
LDA HOURS      ;current HOURS (00-$17)
JSR BIN2DEC    ;make it decimal (00-$23)
JSR WROB       ;output it.
...
```

Errors:

The **BIN2DEC** subroutine does not detect any errors or return error codes. If the calling program passes a binary value larger than $63 to this subroutine, the resulting conversion value will be meaningless.

CHK_SUM ($F048) Registers Expected: NONE

Registers Modified: A,Y

This subroutine calculates the 16-bit checksum of a block of memory. TMP0 (RAM location: $007B) must contain the low byte of the starting address. TMP0+1 ($007C) must contain the high byte of the starting address. The ending address must be stored in the same format beginning at TMP2 ($007E/7F). The computed checksum will reside in two bytes beginning at TMP4 ($0081/82). The carry-bit will be **CLEAR** upon a normal return from this subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **CHK_SUM** subroutine.

Example:

```
LDA #<FIRST
STA TMP0       ;RAM location: $007B
LDA #>FIRST
STA TMP0+1     ;RAM location: $007C
LDA #<LAST
STA TMP2       ;RAM location: $007E
LDA #>LAST
STA TMP2+1     ;RAM location: $007F
JSR CHK_SUM
LDA TMP4       ;Checksum in RAM
LDY TMP4+1     ;locations: $0081/82
...
```

Errors:

The **CHK_SUM** subroutine does not detect any errors or return error codes. If the ending address is lower than the starting address, this subroutine will still return normally. The computed checksum in TMP4 ($0081/82) however, will be meaningless.

CK_CONTC ($F009) Registers Expected: NONE

Registers Modified: A

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. If so, it means that the operator has pressed the control-C (^C) key combination and the **CK_CONTC** subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**. The W65C134 ROM monitor's command processor interprets the ^C key combination as a "cancel current operation" command. Application programs may do the same. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **CK_CONTC** subroutine.

Example:

```
JSR CK_CONTC
BCS MYABORT;Branch if Control-C        ...
```

Errors:

The **CK_CONTC** subroutine does not detect any errors or return error codes.

CRLF ($F012) Registers Expected: NONE

Registers Modified: A

This subroutine sends out a CARRIAGE RETURN[22] ($0D) character to the serial port buffer. The carry-bit will be **CLEAR** upon a normal return from the **CRLF** subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **CRLF** subroutine.

Example:

```
JSR CRLF
BCS MYABORT;Branch if Control-C
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and the **CRLF** subroutine will return with the carry-bit **SET** in the status register.

Errors:

The **CRLF** subroutine does not detect any errors or return error codes.

EXTRESET ($F057) Registers Expected: NONE

Registers Modified: NO RETURN

This is not a subroutine, it is an entry point into the external ROM's reset routine. It is occasionally used for vectoring uninitialized interrupts, resetting the machine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of this monitor function.

[22] The CARRIAGE RETURN ($0D) character may appear on the user's keyboard under another designation (ENTER, NEWLINE, CR, RETURN, etc.).

GETCH **($F00C)** **Registers Expected:** NONE
Registers Modified: A

This subroutine reads the serial port buffer and waits until a character is received. If register-A returns a null ($00) character, and the carry-bit is **SET**, it means that the operator has typed the control-C (^C) combination. The **GETCH** subroutine also echoes the character to the output when it is read. The echo feature may be turned ON/OFF in the SFLAG byte (RAM location: $0072). The carry-bit will be **CLEAR** upon a normal return from the **GETCH** subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **GETCH** subroutine.

Example:

```
JSR GETCH
CMP #$0D        ;IS IT A CR?
BEQ ENDLINE1
...

LDA #$20
TSB $72         ;DISABLE ECHO
JSR GETCH
CMP #$0D        ;IS IT A CR
BEQ ENDLINE2
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. If so, it means that the operator has pressed the control-C (^C) key combination and the **GETCH** subroutine will return with $00 in register-A and the carry-bit **SET** in the status register.

Errors:

The **GETCH** subroutine does not detect any errors or return error codes.

HEXIN ($F03C)

Registers Expected: A
Registers Modified: A

This subroutine converts an ASCII HEX character in register-A to its equivalent binary value. The binary value is returned in the lower nibble of register-A. The carry-bit will be **CLEAR** upon a normal return from this subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **HEXIN** subroutine.

Example:

```
LDA #'d'
JSR HEXIN
BCS NOTHEX3 ;Should never branch
CMP #$0D
BNE FAILED     ;Should never branch
...
LDA #'q'
JSR HEXIN
BCS NOTHEX3 ;Should always branch
...
```

Errors:

If the character passed in register-A is not a valid ASCII HEX character, the **HEXIN** subroutine will return with the carry-bit **SET** in the status register. (Note: The contents of register-A may be modified even if the conversion is not performed.)

IFASC ($F030) Registers Expected: A

Registers Modified: NONE

This subroutine checks the byte in register-A. If the byte is a valid ASCII character[23] it will return with the carry-bit **CLEAR**. The carry-bit will be **SET** upon return if the character was not ASCII. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **IFASC** subroutine.

Example:

```
LDA #$F1
JSR IFASC
BCC ASCII_YES ;Should never branch
...
LDA #$41
JSR IFASC
BCS NOT_ASCII ;Should never branch
...
```

Errors:

If the character passed in register-A is not a valid ASCII character, the **IFASC** subroutine will return with the carry-bit **SET** in the status register. This is not necessarily an error.

[23] This manual uses the term: “ASCII” when referring to visable characters ($1F < char < $7F) within the American Standard Code for Information Interchange. Special characters and control characters are of course also part of the ASCII character set.

ISUM ($F02D)

Registers Expected: A
Registers Modified: NONE

This subroutine checks the byte in register-A. If the byte is a valid ASCII decimal digit, it will return with the carry-bit **CLEAR** in the status register. If the carry-bit is **SET** upon return, the character was not an ASCII decimal digit. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **ISUM** subroutine.

Example:

```
LDA #$FA
JSR ISUM
BCC DEC_YES ;Should never branch
...
LDA #'7'
JSR ISUM
BCS NOT_DEC ;Should never branch
...
```

Errors:

If the character passed in register-A is not a valid ASCII decimal digit, the **ISUM** subroutine will return with the carry-bit **SET** in the status register. This is not necessarily an error.

MS19OUT ($F042) Registers Expected: NONE

Registers Modified: NONE

This subroutine outputs Motorola S19 format to the serial port buffer. TMP0 (RAM location: $007B) must contain the low byte of the starting address and TMP0+1 ($007C) must contain the high byte of the starting address. The address of the last byte of memory to be transferred must be stored in the same format in TMP2 ($007E/7F). The offset address must reside in TMP6 ($0083/84). It will be added to the address field as each record is sent. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **MS19OUT** subroutine.

Example:

```
        STZ TMP6        ;Offset=0 in RAM
        STZ TMP6+1      ;locations: $0083/84
        LDA #<FIRST
        STA TMP0        ;Starting address in
        LDA #>FIRST     ;locations: $007B/7C
        STA TMP0+1
        LDA #<LAST      ;Ending address in
        STA TMP2        ;locations: $007E/7F
        LDA #>LAST
        STA TMP2+1
        JSR MS19OUT
        ...
```

Errors:

The **MS19OUT** library subroutine does not detect any errors or return error codes. If the ending address (TMP2: $007E/7F) is lower than the starting address (TMP0:$007B/7C), this subroutine will return normally.

MS28IN ($F045) Registers Expected: NONE

Registers Modified: NONE

This subroutine inputs Motorola S28 or S19 directly into memory through the serial port buffer. This subroutine returns to the caller at the end of each input line. This subroutine also increments the ERRORS counter (location: $0085) if a checksum error is found. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **MS28IN** subroutine.

Example:

```
JSR MS28IN
LDA ERRORLOC ;RAM location: $0085
BNE BADLOAD   ;Branch if load errors
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and this subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

The **OUTCH** subroutine does not detect any errors, other than checksum.

MVDATA **($F036)** **Registers Expected:** Y
Registers Modified: A, Y

This subroutine moves data from memory to memory. TMP0 (RAM location: $007B) must contain the low byte of the starting address. The high byte of the starting address must reside in TMP0+1 ($007C). The destination address must be stored in the same format in TMP2 ($007E/7F). Register-Y must contain the number of bytes to be transferred. If register-Y is zero then the block size to be moved is 256 bytes.

The **MVDATA** subroutine actually moves the contents of the highest memory location in the source block first. The contents of the lowest memory location in the source block are transferred last. Therefore, if the source address is less than the destination address, then the blocks may overlap. If the source address is greater than the destination address, then the data will be corrupted. Obviously, if both addresses are the same, no meaningful change occurs. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **MVDATA** subroutine.

Example:

```
LDA #<HERE     ;Source address in RAM
STA TMP0       ;locations: $007B/7C
LDA #>HERE
STA TMP0+1
LDA #<THERE ;Destination address in
STA TMP2       ;locations: $007E/7F
LDA #>THERE
STA TMP2+1
LDY #202       ;Move 202 bytes from
JSR MVDATA     ;HERE to THERE.
...
```

Errors:

If there is NO RAM at the destination address, the **MVDATA** subroutine will return with the carry-bit **SET** in the status register. If no errors are detected during execution of this library subroutine, it will return with the carry-bit **CLEAR**.

OUTCH ($F00F) **Registers Expected:** A

Registers Modified: NONE

This subroutine writes a character into the serial port buffer. Register-A must contain the character (7-bit ASCII) to be sent. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **OUTCH** subroutine.

Example:

```
LDA #'A'
JSR OUTCH
BCS ABORT07 ;Branch if ^C received
LDA #'B'
JSR OUTCH
BCS ABORT07 ;Branch if ^C received
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and the **OUTCH** subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

The **OUTCH** subroutine does not detect any errors or return error codes.

PRTSTR **($F01E)** **Registers Expected:** A, X, Y
Registers Modified: A, Y

This subroutine prints a string of characters to the serial port buffer. Register-Y must contain the number of bytes to send. If the specified block size is zero, 256 bytes will be sent. Register-X must contain the low byte of the text string address. The high byte of the string address must reside in register-A. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **PRTSTR** subroutine.

Example:

```
        LDY #STRINGEND-STRING
        LDX #<STRING
        LDA #>STRING
        JSR PRTSTR
        BCS MYABORT ;Branch if ^C
        ...
STRING .BYTE 'Hello There!'
STRINGEND
        ...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and this subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

The **PRTSTR** subroutine does not detect any errors or return error codes.

RD_CHAR ($F006) **Registers Expected:** NONE

Registers Modified: A

This subroutine gets a received character from the serial port if one is already present. Otherwise it will return a NULL ($00). The data is returned in register-A. If an ETX ($03) character has been received by the serial input port, it means that the operator has pressed the control-C (^C) key combination. This subroutine will then return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of this monitor library subroutine.

Example:

```
MYLOOP JSR MYALARM  ;Check/handle alarm
       ...
       JSR RD_CHAR
       CMP #00
       BCS MYABORT ;Branch if ^C received
       BEQ MYLOOP  ;Loop until character
       ...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. If so, it means that the operator has pressed the control-C (^C) key combination and the **RD_CHAR** subroutine will return with the carry-bit **SET** in the status register. The carry-bit will be **CLEAR** for a normal return from this subroutine.

Errors:

The **RD_CHAR** subroutine does not detect any errors or return error codes.

RD_CLOCK ($F04B)

Registers Expected: A, X
Registers Modified: A

This subroutine reads the Time-of-Day clock into a memory buffer. It expects a pointer to a buffer which will receive the data. Register-A must contain the low byte of the buffer address. The high byte of the address must reside in register-X.

Example:

```
LDX #>STRING ;Set pointer to the
LDA #<STRING ;destination buffer.
JSR RD_CLOCK
...
```

The **RD_CLOCK** subroutine puts the seven parameters of the Time-of-Day clock in the format described in the table (below). The advantage of using this subroutine (instead of directly reading memory[24]) is that the clock is prevented from rolling while it is being copied. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **RD_CLOCK** subroutine.

Time-of-Day Clock Format

Byte	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

NOTE: All values must be in 2's complement format.

Errors:

The **RD_CLOCK** subroutine does not detect any errors or return error codes. Obviously, the destination buffer must be located in RAM, or the result will not be meaningful.

[24] The stored values for the Time-of-Day clock use locations: $0063 through $0069 in RAM. Refer to Appendix C – W65C134 Monitor Assembly Listing for more details.

RDOA **($F021)** **Registers Expected:** NONE
Registers Modified: A

This subroutine accepts four ASCII HEX characters from the serial port buffer, and converts them into a 16-bit value. It stores the resulting 2 bytes in TMP0. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **RDOA** subroutine (It calls **RDOB** twice.).

Example:

```
        JSR RDOA
        LDA TMP0
        LDY TMP0+1
        ...
```

Errors:

If errors are detected during the execution of the **RDOA** subroutine, it will return with the carry-bit **SET** in the status register.

RDOB **($F024)** **Registers Expected:** A & X
Registers Modified: A

This subroutine accepts two ASCII HEX characters from the serial port buffer and converts them into a single 8-bit value. The resulting hexadecimal byte is returned in register-A. This subroutine also echoes the character to the output. The echo may be shut off via the serial I/O control flag byte.[25] If a SPACE is entered in either ASCII position, the carry is set on return. If a return is entered, this subroutine jumps back to the start of the monitor, clearing the stack and effectively shutting down the calling subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **RDOB** subroutine.

Example:

```
JSR RDOB
CMP #$FF
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and this subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

If errors are detected during the execution of this library subroutine, it will return with the carry-bit **SET** in the status register.

[25] The serial I/O control flag byte is SFLAG (location: $0072). Refer to Appendix C – W65C134 Monitor Assembly Listing for specific details regarding the flag definitioons in SFLAG.

RTC_MODE **($F051)** **Registers Expected:** A
Registers Modified: A

This subroutine sets or clears the DAYLIGHT SAVINGS flag. It expects one parameter in register-A. If the parameter is: $01, the Time-of-Day clock will make daylight savings shifts when required. If register-A = $00, no shifts[26] will be made. This flag is cleared by the reset initialization sequence after power-up, disabling changes for daylight savings time. The carry-bit will be **CLEAR** upon a normal return from this subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **RTC_MODE** subroutine.

Example:

```
LDA #0              ;Turn the daylight
JSR RTC_MODE        ;savings mode: OFF
...
LDA #1              ;Turn the daylight
JSR RTC_MODE        ;savings mode: ON
...
```

Errors:

If a parameter error is detected during the execution of the **RTC_MODE** subroutine, it will return with the carry-bit **SET** in the status register.

[26] Daylight savings changes occur at midnight on the last Sunday in October and the first Sunday in April

SPAC **($F015)** **Registers Expected:** None
Registers Modified: A

This subroutine sends out a SPACE ($20) character to the serial port buffer. The carry-bit will be **CLEAR** upon a normal return from this subroutine. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **SPACE** subroutine.

Example:

```
JSR SPAC
BCS ABORT003 ;^C received
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. This means that the operator has pressed the control-C (^C) key combination and this subroutine will return with the carry-bit **SET** in the status register.

Errors:

This subroutine does not detect any errors or return error codes.

START **($F039)** **Registers Expected:** NONE
Registers Modified: NO RETURN

This is not a subroutine; it is an entry point into the monitor. Invoking this function will return control to the serial monitor, but the processor registers will not be copied into RAM. This function will reset the stack pointer to $FF. Application programs may call (JSR) or jump (JMP) to this function from anywhere in memory.

Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of this monitor function.

UPPER_CASE **($F033)** **Registers Expected:** A
Registers Modified: A

This subroutine converts a lower-case ASCII (a-z) character into an upper-case ASCII (A-Z) character. The character to be converted must be passed in register-A. The converted result will be returned in register-A. If the byte is not a lower case ASCII character, it will return unchanged. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **UPPER_CASE** subroutine.

Example:

```
LDA #'b'
JSR UPPER_CASE
CMP #'B'
BNE FAILED      ;Should never branch
...
```

Errors:

The **UPPER_CASE** subroutine does not detect any errors or return error codes.

VERSION **($F000)** **Registers Expected:** NONE
Registers Modified: A,X,Y

This subroutine returns the monitor version codes (3 bytes). The first byte is returned in register-A. The second byte is returned in register-X. The third byte is returned in register-Y. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **VERSION** subroutine.

Example:

```
JSR VERSION     ;Get the version data.
JSR WROB        ;Display MS byte.
TXA
JSR WROB        ;Display middle byte.
TYA
JSR WROB        ;Display LS byte.
...
```

Errors:

The **VERSION** subroutine does not detect any errors or return error codes.

WAIT **($F05A)** **Registers Expected:** NONE
Registers Modified: NONE

This subroutine comprises a instruction followed by a **NOP** and an **RTS**. The **WAI** will suspend all activity until an interrupt occurs. The interrupt will usually be for the Time-of-Day clock. This subroutine is in the masked ROM. It might be useful in low power applications wherein some hardware is on, and some off.

Example:

```
LOOP    JSR WAIT          ;Suspend until interrupt.
        JSR RD_CHAR       ;Get character if present.
        BCS MYABORT       ;^C - Cancel
        CMP #0
        BEQ LOOP          ;Go wait for character.
        ...
```

Errors:

The **WAIT** subroutine does not detect any errors or return error codes.

WR_ACLOCK ($F054) Registers Expected: A, X

Registers Modified: A

Time-of-Day Clock Format

Byte	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

NOTE: All values must be in 2's complement format.

This subroutine may be used to set the alarm function. It expects a pointer to a buffer containing the seven parameters associated with the alarm and Time-of-Day clocks. Register-A must contain the low byte of the buffer address. The high byte of the address must reside in register-X. The buffer contents will be interpreted according to the format in the table (right).

The **WR_ACLOCK** subroutine will only change the settings of the alarm function. (Refer to the **WR_CLOCK** subroutine description regarding the Time-of-Day clock.) If any of the alarm time parameters is set to $FF, that particular field will be ignored when comparing the alarm time to the real time.

Example:

```
            LDA #<BIG_EVENT
            LDX #>BIG_EVENT
            JSR WR_ACLOCK
            ...
BIG_EVENT   .BYTE 00,00,13,01,03,89,FF
            ...
```

Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **WR_ACLOCK** subroutine.

Errors:

The **WR_ACLOCK** subroutine does not detect any errors or return error codes.

WR_ADDR ($F027) Registers Expected: A & X
Registers Modified: A

This subroutine prints a 16-bit value in ASCII HEX format to the UART. Register-A must contain the least significant byte of the data. The most significant byte must reside in register-X. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **WR_ADDR** subroutine.

Example:

```
LDX #$F0       ;MS Byte
LDA #$02       ;LS Byte
JSR WR_ADDR ;Prints: "FOO2"
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. If so, it means that the operator has pressed the control-C (^C) key combination, and the **WR_ADDR** subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

The **WR_ADDR** subroutine does not detect any errors or return error codes.

WR_CLOCK ($F04E) Registers Expected: A, X

Registers Modified: A

Time-of-Day Clock Format

Byte	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

NOTE: All values must be in 2's complement format.

This subroutine may be used to set the Time-of-Day clock function. It expects a pointer to a buffer containing the seven parameters associated with the alarm and Time-of-Day clocks. Register-A must contain the low byte of the buffer address. The high byte of the address must reside in register-X. The buffer contents will be interpreted according to the format in the table (right).

The real advantage of using this subroutine over directly accessing memory[27] is that the Time-of-Day clock will not roll while it is being written. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **WR_CLOCK** subroutine.

Example:

```
        LDX #>NEWTIME ;Set TOD Clock
        LDA #<NEWTIME
        JSR WR_CLOCK
        ...
NEWTIME .BYTE 00,00,13 ;1PM, Friday, June 19, 1993
        .BYTE 19,06,93,6
        ...
```

Errors:

The **WR_CLOCK** subroutine does not detect any errors or return error codes.

[27] The stored values for the Time-of-Day clock use locations: $0063 through $0069 in RAM. Refer to Appendix C – W65C134 Monitor Assembly Listing for more details.

WROB ($F02A) Registers Expected: A

Registers Modified: A

This subroutine writes a byte from register-A as two ASCII HEX characters to the serial port buffer. Refer to **Appendix C - W65C134 Monitor Assembly Listing** for specific details regarding the internal operation of the **WROB** subroutine.

Example:

```
LDA #$AB
JSR WR0B        ;Prints: "AB"
...
```

Cancel:

This subroutine checks to determine if an ETX ($03) character has been received by the serial input port. If so, it means that the operator has pressed the control-C (^C) key combination and the **WROB** subroutine will return with the carry-bit **SET** in the status register. If not, the carry-bit will be **CLEAR**.

Errors:

The **WROB** subroutine does not detect any errors or return error codes.

Development Tools

The W65C134 monitor was developed by The Com Log Company, Inc. to support the Project Computer.[28] It therefore makes some basic assumptions about the hardware configuration, and also about how some features may be used by the software. These assumptions are not unreasonably restrictive.

The Project Computer was designed to be a prototyping platform for products using the W65C134. It's features were selected after considering just "How" the W65C134 would be used in products. Com Log has used the Project Computer as the basis for over a dozen projects and products, including: telephones, controllers, and test equipment.

A fundamental assumption of the W65C134 monitor is that most software development will occur in a separate, "host" system which may communicate with the target system via the serial port. Compilations, assemblies, and linking will be performed on the host system. Executable object may be placed in EPROM or downloaded serially to RAM as S19 or S28 records. The monitor makes no specific assumptions about the host system. It may be an Apple II, MacIntosh, IBM compatible, or even a mainframe computer system.

Most of the debugging capabilities of the W65C134 monitor only work if the program resides in RAM. Some features are useful even in the minimum configuration (ie. no external RAM or EPROM).

The code examples in this manual were written, assembled, and linked on an IBM AT using 2500AD's 65C816 cross assembler [29]and linker. It is the preferred cross assembler for IBM compatibles. The tools from 2500AD are simple to use, very powerful, with good macro capability, and the company has been very responsive to questions and problems. Other cross-assemblers are available and may better suit the user's preferences.

The MANX 'C' compiler runs only on the Apple II family and produces code which can be used on the W65C134.[30]

The Com Log Company, Inc[31]. offers an optional BASIC interpreter for the Project Computer. It resides in EPROM at locations: $A000-$DFFF. The interpreter allows BASIC programs to be entered, edited, saved and downloaded via the serial port. While it can utilize features specific to the Project Computer, the interpreter only requires the W65C134 internal monitor.

Com Log BASIC has two important features: (1) BASIC programs may be placed in EPROM for stand-alone operation, and (2) It supports user interrupts for real-time control applications.

[28] The Project Computer actually contains two monitors. The W65C134 internal ROM monitor supports the most generic features. The Extended Monitor resides in EPROM locations: $E000 through $EFFF. It supports specific features which are unique to the Project Computer configuration (ie. pushbuttons, LCD display, etc.).

[29] 2500AD Software Inc
109 Brokdale Ave, Box 480
Buena Vista, Colorado 81211
Telephone: (719) 395-8683

[30] Compilers for the Apple IIGS may not produce correct code for the W65C134.

[31] The Com Log Company, Inc.
7825 East Redfield Road, Suite A
Scottsdale, Arizona 85260

Appendices

Appendix A - W65C134 Monitor Commands

W65C134 Monitor Commands

? or H Lists the commands available

T Reads current time of day clock and displays the result

R Display processor registers (PC, F, A, X, Y, SP)

A Alter registers in the order PC F A X Y SP. A space skips to the next register. A carriage-return ends the command.

M Alter Memory address and locations. The first 4 characters entered after the ‘M’ set the current address pointer. Each pair of characters entered after the address changes the byte at the current address. A space entered after a byte change increments the current address pointer. A CR will end the command, and can be entered after the initial address or any number of byte changes.

D Displays memory from start address to end address

The format is: D SSSS EEEE
wherein: SSSS is the start address
EEEE is the end address

> Increment the current address and display the contents. (The M command above, can set the current address pointer.)

< Decrement the current address and display the contents. (The M command above, can set the current address pointer.)

SPACE Uses the current address pointer and displays the contents.

F Fill memory from start address to end address with value.
The format is: F SSSS EEEE W.
wherein: SSSS is the start address
EEEE is the end address
W is the fill character/bytes.

The format is: V SSSS EEEE XX
wherein: SSSS is the start address
EEEE is the end address
XX is the number of bytes
If XX – 0, the 256 bytes are moved.

G(CR) Begin execution from the current address in the PC.

G ADDR Set PC to ADDR and begin execution.

J(CR) Do a JSR to the current PC address

J ADDR Do a JSR to ADDR

U Jump through the USR command vector. (This is a hook to allow additional command to be added to the monitor.)

B Jumps to BASIC cold start

K Jumps to BASIC warm start

I/O Commands

X Switches the system from hardware handshake to XON-XOFF. This is a toggle command, each time X is entered, the system switches. The flag is zero for hardware handshake.

S Start of a data record in Motorola S28 or S19 format. When this command is received, data is not echoed until a CR is received. This command is used to load programs, etc and the Error (E) command should be used after loading a number of S28 records to check for date errors.

W Output data in Motorola S28 format. This command outputs 18 byte records (the last record may be less) from start address to end address.
The format is: W SSSS EEEE
wherein: SSSS is the start address
EEEE is the end address

E Display the number of S28 receive errors noticed. While ‘S’ records are received, an accumulation of checksum errors is kept. This command displays that accumulation. Once displayed, the error number will be cleared to 0.

C Displays a checksum of memory from start address to end address.
The format is: C SSSS EEEE
wherein: SSSS is start address
EEEE is end address

Appendix B - W65C134 Monitor Subroutines

```
          'F000.ASM – Entry points to the monitor ROM'

VERSION         EQU       $F000     ; GET FIRMWARE MONITOR VERSION DATA.
ACI_INIT        EQU       $F003     ; INITIALIZES SERIAL COMMUNICATION.
RD_CHAR         EQU       $F006     ; RETURNS NEXT CHARACTER FROMSERIAL.
CK_CONTC        EQU       $F009     ; RETURNS W/ CARRY SET IF CONTROL-C INPUT.
GETCH           EQU       $F00C     ; READ SERIAL CHARATER, IF PRESENT.
OUTCH           EQU       $F00F     ; WRITE CHARACTER/BYTE TOSERIAL PORT.
CRLF            EQU       $F012     ; PRINT CARRIAGE RETURN & LINE FEED.
SPAC            EQU       $F015     ; PRINTS A SPACE CHARACTER.
ASCBIN          EQU       $F018     ; ASCII TO BINARY CONVERSION.
BINASC          EQU       $F01B     ; BINARY TO ASCII HEX CONVVERSION.
PRTSTR          EQU       $F01E     ; PRINTS A STRING OF CHARACTERS.
RDOA            EQU       $F021     ; READS AN ADDRESS (FOUR ASCII HEX CHARS.)
RDOB            EQU       $F024     ; READS A BYTE (TWO ASCII HEX CHARACTERS.)
WR_ADDR         EQU       $F027     ; WRITE AN ADDRESS AS FOUR ASCII HEX CHARS.
WROB            EQU       $F02A     ; WRITE A BYTE AS TWO ASCII HEX CHARACTERS.
ISUM            EQU       $F02D     ; CHECKS FOR ASCII DECIMAL DIGIT.
IFASC           EQU       $F030     ; CHECKS FOR DISPLAYABLE ASCII HEX CHARACTERS.
UPPERCASE       EQU       $F033     ; CONVERT LOWER CASE ASCII TO UPPERCASE.
MVDATA          EQU       $F036     ; MOVE A BLOCK OF MEMORY DATA.
START           EQU       $F039     ; JUMPS TOCOMMAND PROMPT & RESETS STACK.
HEXIN           EQU       $F03C     ; CONVERTS ASCII HEX INPUT TO BINARY.
BIN2DEC         EQU       $F03F     ; CONVERTS BINARY TO PACKED DECIMAL.
MIS19OUT        EQU       $F042     ; OUTPUT MEMORY IN MOTOROLA S19 FORMAT.
MS28IN          EQU       $F045     ; INPUTMOTOROLA S19 OR S28 RECORDS.
CHK_SUM         EQU       $F048     ; CALCULATE CHECK SUM ON MEMORY BLOCK.
RD_CLOCK        EQU       $F04B     ; READ TIME-OF-DAY CLOCK INTO BUFFER.
WR_CLOCK        EQU       $F04E     ; WRITE BUFFER INTO TIME-OF-DAY CLOCK.
RTC_MODE        EQU       $F051     ; SET/CLEAR DAYLIGHT SAVINGS MODE.
WR_ACLOCK       EQU       $F054     ; WRITE BUFFER INTO PROGRAMMABLE ALARM.
EXTRESET        EQU       $F057     ; VECTOR W/TESTING, TO EXTERNA; RESET.
WAIT            EQU       $F05A     ; WAIT UNTIL INTERRUPT OCCURS & RETURN.

                .END
```

```
        ‘Important Internal RAM Locations Used By Monitor Subroutines’

UCMDPTR         EQU     $0050   ; Pointer to user command processor

; Time-of-Day Clock
SEC             EQU     $0063   ; Seconds   0-59
MIN             EQU     $0064   ; Minutes   0-59
HR              EQU     $0065   ; Hours   0-23
DAY             EQU     $0066   ; Day of month   1-28, 28, 30, 31
MONTH           EQU     $0067   ; Month   1=JAN, 12=DEC
YR              EQU     $0068   ; Year
DAYWK           EQU     $0069   ; Day of week   1=Sunday, 7=Saterday

; Daylight Savings Time Mode Flag
DAYLIT          EQU     $006A   ; Bit #0: MODE 1=ON, 0=OFF
                                ; Bits $1-6: Not used.

; Programmable Alarm
ASEC            EQU     $006B   ; Seconds   0-59
AMIN            EQU     $006C   ; Minutes   0-59
AHR             EQU     $006D   ; Hours   0-23
ADAY            EQU     $006E   ; Day of month  1-28, 28, 30, 31
AMONTH          EQU     $006F   ; Month 1=JAN, 12=DEC
AYR             EQU     $0070   ; Year
ADAYWK          EQU     $0071   ; Day of week 1=Sunday, 7=Saterday

; Serial Communication Control Flag
SFLAG           EQU     $0072   ; Bit #1: ˆC received
                                ;      #5: Echo control  1=OFF, 0=ON

; Alarm Flags
MONFLGS         EQU     $0072   ; Bit #3:  Alarm enable
                                ;      #4:  Alarm interrupt occurred

; Working Variables
TMP0            EQU     $007B   ;
TMP2            EQU     $007E   ;
TMP4            EQU     $0081   ;
TMP6            EQU     $0083   ;

; Errors Flag From Loading ‘S’ Records
ERRORS          EQU     $0085   ; This must be cleared by user, but is incremented by loader.

; Locations w/ special meaning. (Refer to text for details.)
;           $0200-0202 = ‘WDC’ for user startup routine in RAM
;           $8000-8002 = ‘WDC’ for user startup routine in EPROM
;           $E000 = $4C (JMP) for additional initialization
;           $EFFD = $4C for alternate parser

                .END
```

W65C134 Monitor Subroutine

F000 **VERSION** GET MONITOR VERSION
Returns monitor version in register-A, register-X, and register-Y.

F003 **ACI_INIT** INITIALIZES ACIA
Initializes ACI to baud rate in register-A, data length in register-X, and parity in register-Y. Anything in the current buffers is flushed. Baud rate values are:

A	Baud Rate	A	Baud rate
0	75	6	1800
1	110	7	2400
2	150	8	4800
3	300	9	9600
4	600	A	19200
5	1200	B	38400

Register-X has data length:
7 – 7 bits
8 – 8 bits
Register-Y has parity:
B0 – 1 enables
B1 – 1: even parity
B1 – 0: odd parity

F006 **RD_CHAR** RETURNS A CHARACTER
from ACIA (if present). Otherwise it will return a 00 (null). The carry bit = 1 if ˆC is encountered, and control-C flag will be reset. (See **CK_CONTC**).

F009 **CK_CONTC** RETURNS WITH CARRY SET IF ˆC
Checks for a control-C and returns w/carry = 1 if detected, else returns with carry = 0

F00C **GETCH** READ ACIA
Waits until a character is received. Returns w/carry = 1 if ˆC and register-A = null ($00). The control-C flag will be reset (see **CK_CONTC**). This subroutine will also echo the character. The echo feature may be switched off via bit-5 of SFLAG (addr 0072).

F00F **OUTCH** WRITE ACIA
Register-A must contain the character (7-bit ASCII) to be sent. This will return w/carry = 1 if ˆC detected, or carry = 0 for normal.

F012 **CRLF** PRINT A CARRIAGE RETURN
(Modifies register-A.) This will return w/carry = 1 if ˆC detected, or carry = 0 for normal.

F015 **SPAC** PRINTS A SPACE
(Modifies register-A.) This will return w/carry =1 if ˆC detected, or carry = 0 for normal.

F018 **ASCBIN** ASCII TO BINARY CONVERSION
Expects: register-A = ASCII LSB and register-X = ASCII MSB. Returns w/binary result in register-A.

F01B **BINASC** CONVERTS BINARY TO ASCII HEX
Expects: binary in register-A and returns w/register-A = LS nibble in ASCII HEX. Register-X will contain the MS nibble in ASCII HEX.

F01E **PRTSTR** PRINTS A STRING
Expects: Register-A = MSB of string address, register-X = LSB, and number of characters (0 means 256) in register-Y. (Modifies registers: A & Y.) This will return w/carry = 1 if ˆC detected, or carry = 0 for normal.

F021 **RD0A** READS ADDRESS (4 ASCII HEX CHRS)
Accepts 16-bit value as four ASCII HEX digits from ACIA. Returns: (No Errors) TMPO = LSB and TMP0 = 1 = MSB. (Locations: $007B and $007C) This subroutine will also echo the character.

F024 **RD0B** READS BYTE (2 ASCII HEX CHRS)
Accepts8-bit value as two ASCII HEX digits from ACIA. Returns: register-A (No Errors) This subroutine will also echo the character.

F027 **WR_ADDR** WRITE AN ADDRESS AS ASCII HEX
Returns: carry = 1 if ˆC detected.

F02A **WROB** WRITE A BYTE AS ASCII HEX
Returns: carry = 1 if ˆC detected.

W65C134 Monitor Subroutine

F02D **ISUM** CHECKS IF ASCII DECIMAL DIGIT
Returns: carry = 1 if register-A is not ASCII decimal digit. Carry = 0 when it is ASCII decimal digit.

F030 **IFASC** CHECKS IF ITS ASCII
Returns: carry = 1 if register-A is not a displayable ASCII character. If $1F < char < $7F, then Carry = 0.

F033 **UPPER_CASE** LC TO UC TRANSLATION
The ASCII character in register-A is converted to upper case ASCII.

F036 **MVDATA** MOVE BLOCK OF MEMORY DATA
Expects: Source address in word at TMP0/$007B (Format: LSB, MSB) and the word at TMP2/$007E holds the destination address. Register-Y = # of bytes to be moved. If Y = 0, it will move 256 bytes. OVERLAP WARNING: Memory blocks are moved from high end first. Returns: Carry = 1 if no RAM at destination location.

F039 **START** NOT A SUBROUTINE
This is used to get back to the command prompt for the monitor. It resets the stack from the memory location that gets updated on the BRK command.

F03C **HEXIN** CONVERT ASCII HEX TO BINARY
Expects: ASCII HEX in register-A, and returns binary also in register-A. Normal: carry =0; if not ASCII HEX, then carry = 1.

F03F **BIN2DEC** Converts value in register-A from hex to packed decimal. Do not exceed 99.

F042 **MS19OUT** MOTOROLA S19 OUTPUT
Writes a block of memory to the UART using the Motorola ‘S19’ record format. Expects: TMP0/$007B = starting address and TMP2/$007E = ending address. TMP6/$0083 = Offset address. Address field of S-Record is data address + offset)

F045 **MS28IN** MOTOROLA S28 INPUT (Also S19)
Reads Motorola ‘S’ records from UART and update ERRORS ($0085) counter.

F048 **CHK_SUM**CALCULATED CHECK SUM
Calculates checksum for a block of memory. Expects: TMP0 = starting address and TMP2 = ending address. Returns: computes checksum word inTMP4/$0081.

F04B **RD_CLOCK** READ TIME-OF-DAY CLOCK
Copies Time-of-Day Clock data to a buffer wo/wrap risk. Expects: Pointer to buffer, wherein register-A = LSB and register-X = MSB. The time buffer format:

Bytes	Contents
0	Seconds
1	Minutes
2	Hours (0-23)
3	Day of Month
4	Month (1-12)
5	Year
6	Weekday (1-7)

F04E **WR_CLOCK** LOAD TIME-OF-DAY CLOCK
Copies a time buffer into the Time-of-Day Clock wo/wrap risk. Expects: Pointer to buffer, wherein register-A = LSB & register-X = MSB.

F051 **RTC_MODE** DAYLIGHT SAVINGS MODE
This subroutine sets/clears the flag which instructs the Time-of-Day clock to make daylight saving shifts when appropriate. Expects: register-A = 1, Daylight Savings mode ON. If register-A = 0, no shifts will be made. Returns: carry = 1 on error.

F054 **WR_ACLOCK** SET PROGRAMMABLE ALARM
Copies a time buffer into the Programmable Alarm. Expects: Pointer wherein register-A = LSB and register-X = MSB. ($FF = Wild Card)

F057 **EXTRESET** NOT A SUBROUTINE
Entry point to external ROM reset logic.

F05A **WAIT** WAIT FOR INTERRUPT
This subroutine executes a WAI instruction which puts the CPU to sleep until an