Mensch Computer™

Developer Guide

Thank you for choosing the state-of-the-art features of the W65C265S microprocessor from Western Design Center. This manual describes the operating system, library subroutines, connector pinouts, and other useful information about the Mensch Computer development platform.

The Mensch ROM Monitor and Mensch Operating System were developed by the Com Log Company, Inc. for the Mensch Computer.

For best results, we recommend that you please carefully read this manual completely before you attempt to develop applications on the Mensch Computer. This manual contains important information on the proper use of the Mensch Operating System and its library of subroutines.

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 related products.

Users of the Mensch Computer, especially developers, 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 W65C265S microprocessor, the Mensch Computer, and associated firmware or software.

Nothing herein shall be construed as a recommendation to use the W65C265S or Mensch Computer 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

TABLE OF FIGURES

Introduction

The Mensch computer is a product idea which has finally achieved reality. Named after William D. Mensch, Jr., founder, chairman and CEO of The Western Design Center, Inc. (WDC), it is the product of many years of discussion, review and planning. Bill is well-known as the designer and patent holder of the W65C02 and W65C816 microprocessors which were used in early Apple computers and Super Nintendos and are being used in millions of products around the world today. He was honored in 1991 at the Microprocessor Forum as one of the pioneers of the microprocessor industry. It has been Bill's vision to create a true *solid state computer*. The **Mensch** is not a game computer or a "business" computer. It is not a PC nor does it compete in the PC marketplace. It is not a calculator, home controller or personal organizer although it *could* be any of these. This new class of *solid state computer* can include a multitude of user-specific applications supported by a single platform. It is based upon a philosophy which believes: **user empowerment** does not require complexity but does require simplicity and usefulness without intimidation.

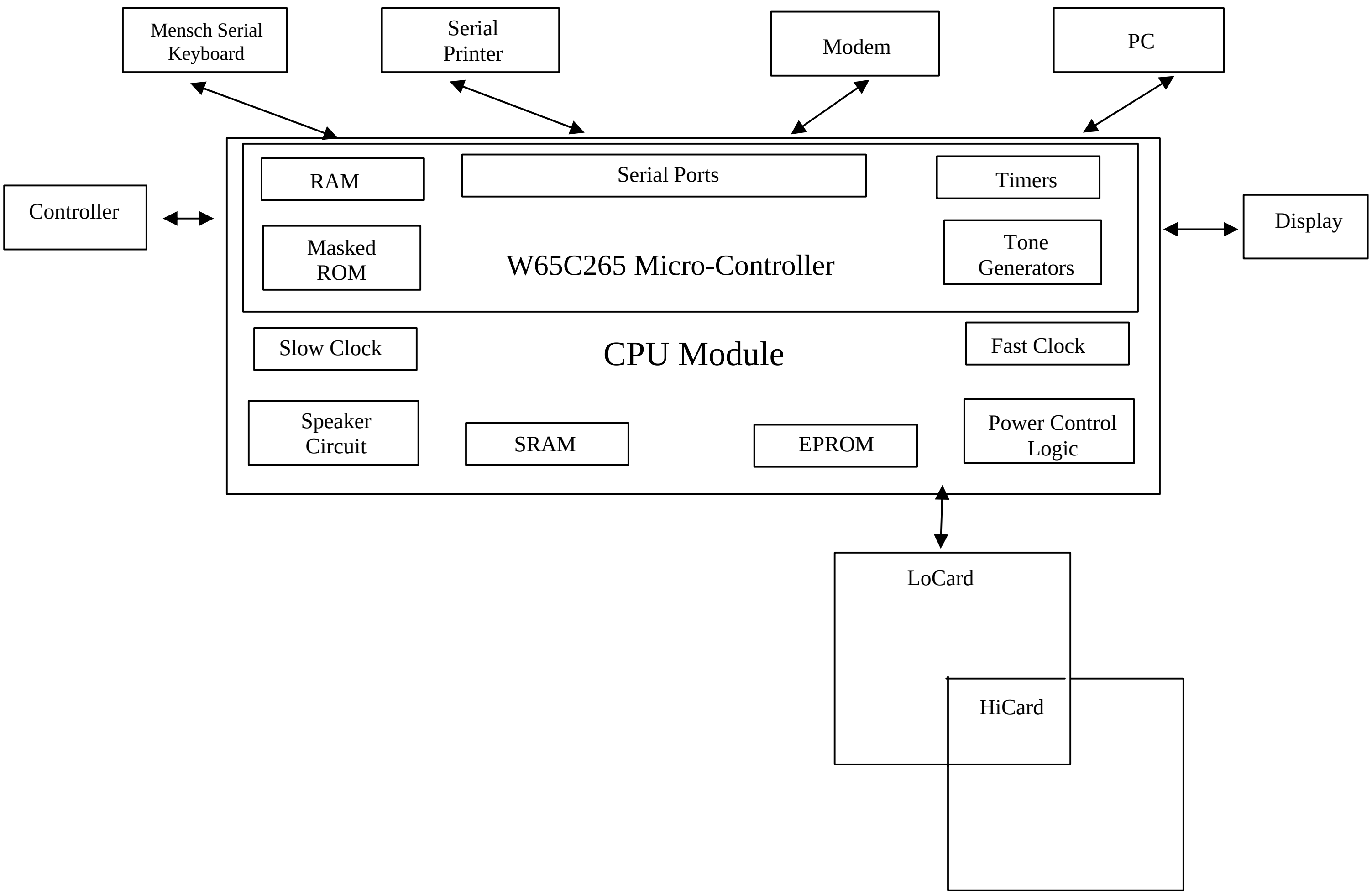

Figure 1
Mensch Computer
Primary Components

Bill Mensch has designed the W65C265 microcomputer for his *solid state computer* and is the core of the CPU module. This powerful chip has a CPU which is instruction-compatible with the W65C816 microprocessor and has four serial ports and eight 16-bit timers on-board. It also has twin tone generators and internal RAM and ROM. The CPU module also has (2) of WDC's VIAs (Versatile Interface Adapters), the W65C22S, for external display and controller interfaces and power management control. The Keytronics "space-saver" keyboard utilizes WDC's 8-bit microcomputer, the W65C134S. The W65C134S provides low power management features and the ability to program the keyboard via a provided on-board 32K EPROM.

Figure 2
Mensch Computer
Assembled

The internal CPU instruction set is 100% compatible with the W65C816 CPU used in the Apple IIgs and Super Nintendo computers as well as the 6502s used in Atari and Commodore computers. It is reasonable to assume that some of the first 3rd-party applications on this solid state computer will be ported from these computer platforms.

IC Memory Cards

The inclusion of IC Memory Cards into the design makes the **Mensch** hardware very versatile. A separate "slow" clock, "low-power" mode and many power management features on the W65C265S allow it to be used effectively in portable, battery-powered configurations. Though the IC cards appear to be memory blocks to the **Mensch**, it can read and manipulate "files" on them. This allows the exchange of data with DOS-compatible computers on a common physical media.

Communications

Four serial ports provide the **Mensch** with adaptability to many types of communication applications. The low power, programmable keyboard uses one of the serial ports. The initial configuration additionally allocates one port each for a serial printer, a modem and a direct link to another computer. These are all generic serial ports which may be used for other purposes by specific application.

The telephone in it's various forms, is rapidly becoming the most important appliance in our lives. The **Mensch** offers a platform which supports telephone –related application.

MenschWorks is an application IC Card for:

1) *Terminal Emulation*
2) *E-Mail Terminal*
3) *MenschCall* (textual preamble to a voice communication)
4) *MenschMail* (Mensch Computer –to-Mensch Computer screen-to-screen communication)

Mensch Computer Availability

The **Mensch** is manufactured and distributed to order and a limited number are products for qualified developers.

Configuration

The Mensch CPU module, Keyboard, and Display are packed in individual boxes for shipping. The Keyboard and Display are easily connected to the CPU. The charger and peripherals attach to the Mensch CPU module via connectors on the rear panel. (See Figure 3)

Figure 3
Mensch Computer
Rear Panel

CPU Module

The CPU module is the core of the Mensch. It contains the main board with the W65C265 microcontroller[1]. This also includes: both fast and slow clock circuits, appropriate power control logic, a speaker circuit, RAM, EPROM, and connectors for external peripherals.

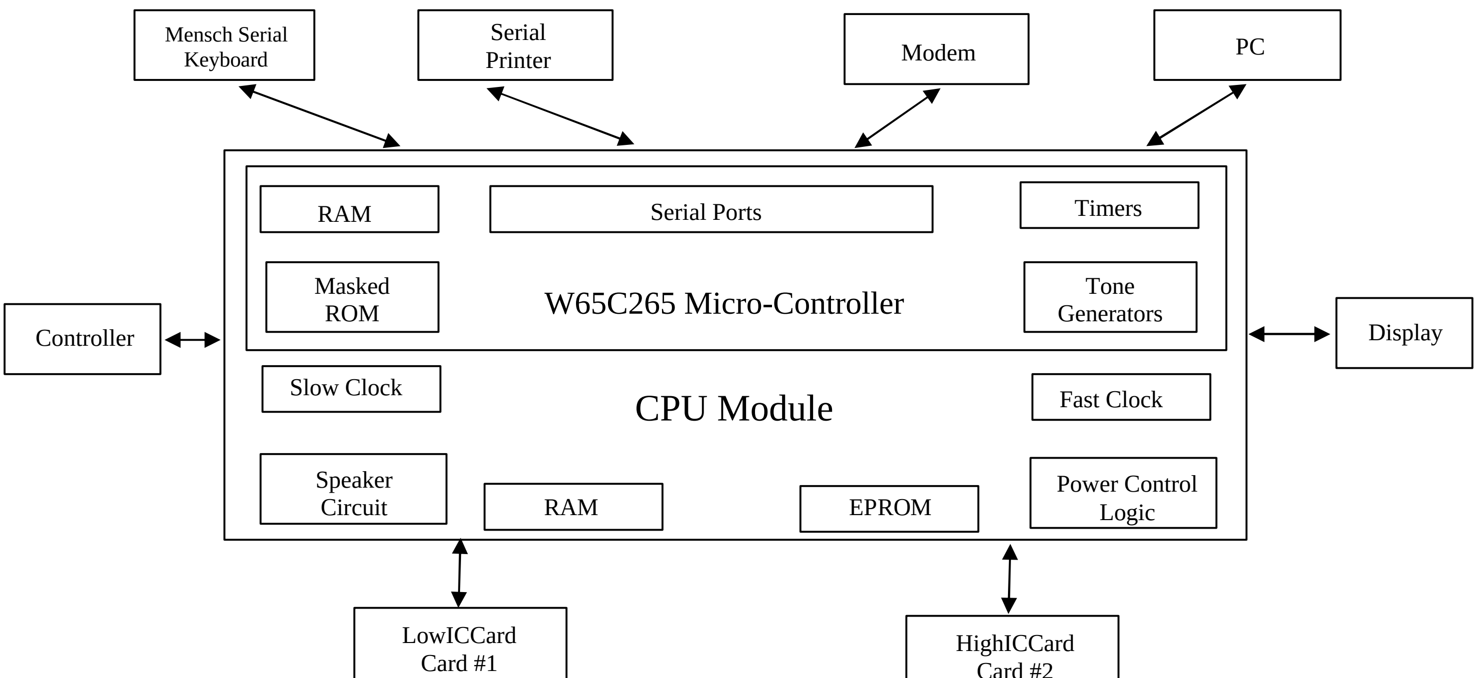

Figure 4
Mensch Computer
CPU Module

1 The Western Design Center, Inc. offers a variety of additional information on the W65C265 micro-controller. (Refer to **Appendix F – Bibliography** for specific titles.)

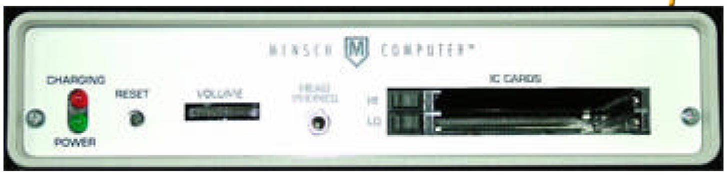

Figure 5
Mensch Computer
Front Panel

Power Control

There is no power ON/OFF switch on the Mensch. A special *low-power mode* feature reduces power consumption when the system is inactive. Power is never removed from the W65C265 chip itself [2]. Background operations in the Firmware maintain the time-of-day clock. The support software for this resides in the masked ROM of the W65C265 chip. It can continue to operate even in low-power mode.

CHARGER Jack

The CHARGER jack is located on the rear panel of the CPU module. The external Charger/Power module plugs into a standard wall outlet and provides a DC voltage to recharge the internal battery pack. (Refer to: **Power Subsystem** section for more explanation.)

CHARGING Indicator

Whenever the system is charging the battery pack, the CHARGING indicator LED (red) will glow. (Refer to: **Power Subsystem** section for more explanation.)

POWER Indicator

This green LED will glow whenever power (+5 volts) is available to the LCD display connector[3]. Generally, when this LED is dark, there is not sufficient power available to operate the Mensch.

RESET Button

The user may force a system reset at any time by pressing the RESET button on the front panel of the CPU module. (Refer to: **RESET Initialization Sequence** for more information.)

Internal Speaker & HEAD PHONES Jack

The Mensch allows application programs to generate sound via the speaker port. The internal amplifier is connected to the internal speaker through a *normally closed* switch in the HEAD PHONES jack. External headphones, or an external amplifier and speaker may be plugged into the jack on the front panel of the CPU module, replacing the internal speaker in the circuit.

VOLUME Control

The output of the speaker port may be adjusted using the built-in VOLUME Control. This is just a potentiometer feeding the internal amplifier. It is located on the rear panel of the CPU module.

Expansion Connectors

All relevant signals from the CPU module are available on the rear expansion connectors. These may be used by developers to monitor internal activity, or interface new peripheral circuits to the Mensch. (Refer to: **Appendix D.6** for details of the expansion connector pinouts.)

---

[2] Power is available to the W65C265 chip and "slow" clock circuitry whenever a battery is attached to the internal battery connector. (Refer to: **Power Subsystem** for more explanation.)

[3] The indicator does not require that the LCD display module be attached.

Serial Port Connectors

A block of four (4) serial port connectors is located in the center of the rear panel of the CPU module. (Refer to: **Appendix D.3** for details of the serial port connector pinouts.)

Ports

S0	Keyboard
S1	Printer
S2	Modem
S3	PC Link

Pinouts

1	GND
2	TXD
3	+5 volts
4	RXD
5	DSR
6	DTR

Memory Map

The Mensch Monitor resides in the 8K byte mask ROM from $00:E000 to $00:FFFF. The Mensch ROM Monitor executes an initialization sequence after reset occurs[4].

It turns on the external bus, checks locations $00:8000-$00:8002 and jumps to $00:8004 if a 'WDC' is found. The Mensch ROM Monitor uses this signaling to begin execution in then internal ROM of the W65C265 chip, and the switch under software control to external EPROM in the Mensch. (Refer to: **RESET Initialization Sequence** for more information.)

The CPU module has a socket for a 256K EPROM (32K bytes). This has been mapped to address range: $00:8000-$00:DEFF. The external portion: the Mensch Operating System resides in the EPROM[5]. Refer to: **Reset Initialization Sequence** for a description of how this is used.

There is also a socket on the board for 32K bytes of RAM. Initially, this RAM is reserved for use by system firmware. This RAM chip has been mapped to address range: $00:0200-$00:7FFF in the Mensch configuration.

Mensch Computer Memory Map

Address Range	Function
$00:0000-$00:00FF	W65C265S internal RAM, (Page #0)
$00:0100-$00:0138	RAM IRQ Vectors
$00:0139-$00:01FF	W65C265S internal RAM
$00:0140-$00:02FF	Mensch Computer Stack.
$00:0300-$00:7FFF	Variables & buffers used by the Mensch OS in External RAM Memory
$00:8000-$00:DEFF	"WDC" semaphore and Mensch Operating System in external EPROM
$00:DF00-$00:DFFF	Reserved addresses and Mapped I/O
$00:E000-$00:FFFF	W65C265S Internal ROM, monitor firmware.
$01:0000-$3F:FFFF	Low IC Card Memory
$40:0000-$BF:FFFF	High IC Card Memory
$C0:0000-$FF:FFFF	Accessible via the Expansion Connector

4 This feature may be used or disabled when the W65C265 is used in other configurations. It is internal to the W65C265 chip, and does not require the Mensch Computer or the external EPROM-based Mensch Operating System. (Refer to **W65C265S INFORMATION, SPECIFICATION, AND DATA SHEET** or the **Mensch Monitor ROM REFERENCE MANUAL** for details.)

5 Developers may choose to use this hardware configuration, but replace the EPROM with their own custom firmware. While such an approach may be viable for specialized applications, it restricts the use of other independently developed software.

Plug-In IC Memory Cards

The Mensch has 32K bytes of internal RAM to be used by the firmware, operating system and specific applications. Most memory in the system is assumed to reside on removable IC memory cards. The Mensch is equipped with two slots of the PCMCIA form factor, supporting a subset of the Type II standard. Additional memory for the Mensch may reside on either or both plug-in cards. The LowICCard has been mapped to a base address of: $01:0000. The HighICCard had been mapped to a base address of: $40:0000.

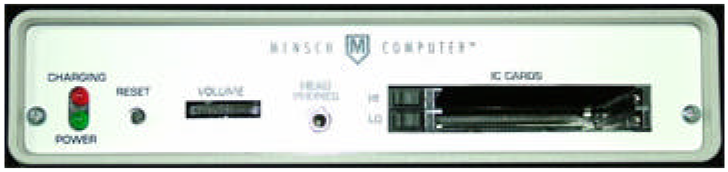

Figure 6
IC Card Slots

Developers are encouraged to design their applications software to work *with* the Mensch Operating System. This is most easily accomplished by locating their applications in the external memory cards and using the subroutine library to access standard features of the system.

LowICCard

This bottom slot is labeled “LO” on the front panel. It is mapped into memory such that the lowest available address is: $01:0000. The highest address usable in this slot is: $3F:FFFF.

HighICCard

This top slot is labeled “HI” on the front panel. It is mapped into memory such that the lowest available address is $40:0000. The highest address usable in this slot is: $BF:FFFF.

Figure 7
Mensch Computer
IC Memory Cards

DOS™ File Support

IC cards conforming to the PCMCIA Type II standard are used on many portable palmtop and laptop computers. Most of these portables are IBM-compatible and therefore use some version of the DOS operating system. This allows the cards to be treated as file devices, like floppies, when transporting data. The Mensch subroutine library provides support which allows programs to access DOS-compatible data files on the IC memory cards. (Refer to: **Programming The Mensch Computer, Using DOS-Compatible File Support** for more information.)

Display

The Mensch uses a liquid crystal display (LCD) which offers a 2.40" by 4.25" viewing area. The LCD and associated electronics are mounted in a low profile case (7"W x 5.5"H x 1"D). The lightweight case is attached to a swivel bracket which may hang on a wall or sit on a flat surface. It supports both character mode and graphics mode. It will display text as sixteen lines of forty characters each. The graphics resolution is 240 horizontal and 128 vertical dots. This module contains a Densitron LCD (LM3229A128G240SNG) and a Toshiba controller (T6963C) [6] board.

Figure 8
Mensch Computer
Display Module

Contrast Control

The readability of the display may be adjusted using the contrast potentiometer located on the right-hand side of the unit.

[6] The data sheet on the LM3229A128G240SNG and **Application Notes for the T6963C LCD Graphics Controller** from Densitron provide a detailed description of this display and its operation.

Cabling and Connections

A ribbon cable with 24 pin connectors connects the Mensch LCD module to the Mensch CPU module. This is a symmetrical cable[7] and the connectors are keyed for proper insertion. Connect either end of the cable to the CPU module using the matching connector on the rear panel.

Figure 9
Mensch Computer CPU Module
Rear Panel

Plug the remaining end of the cable into matching connector on bottom of the LCD display module encasement. (See Figure 8)

There is a small adaptor board between the T6963C controller and the 24-pin connector. This accommodates the contrast control potentiometer.

Programming Support

The Mensch Operating System provides several library subroutines for programmers to use when writing to the LCD screen.

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Those library subroutines which support generalized output streams may be configured to write to the LCD screen. Refer to the description of the: CONTROL_OUTPUT subroutine for specific details.

7 The ribbon cable is symmetrical, such that there is direct one-to-one connection between each pin and it's corresponding, identically numbered, pin on other connector. (Refer to: **Appendix D.4 – LCD Display Cable Connector (Pinouts)** for signal descriptions.)

Display Support Subroutines	
_Box	_VLine
_Circle	_WrDec
_ClearColor	CLEAR_LCD_DISPLAY
_ClearFill	CLEAR_TO_END_OF_LINE (Text line)
_CONTROL_DISPLAY (Power)	DISP_LCD_STRING (@ Text Cursor Position)
_DISP_LCD_HEADER	MENU_POINT
_DO_MAIN_MENU	MENU_SETUP
_HLine	MOVE_PAGE_TO_BUFF
_Line	MOVE_BUFFER_TO_LCD
_Point	POSITION_PIXEL
_PtScreen	POSITION_TEXT_CURSOR (@ Row & Column)
_SetColor	RD_LCD_STRNG
_SetFill	RETRIEVE_DISPLAY_STATUS
_SetText	WRITE_PIXEL
_SetGraph	WR_LCD_STRNG
_SetGraphText	WRITE_LCD_CHARACTER (@ Text Cursor Position)
_TIME_DATE_CHK	

Alternative I/O Usage

Other LCD modules may be used with the T6963C controller module, providing they do not require different signals. This allows the Mensch to be a development/prototyping platform for many different products. Developers should realize that the firmware library subroutines are specific to the display which is provided with the Mensch. Changing displays may also require custom support software.

Note: The W65C22 chip is mapped to address range: $00:DF00 through $00:DF1F. The least significant four address bits select internal registers: A0=RS0, A1=RS1, A2=RS2, and A3=RS3.

Keyboard

A low-profile keyboard is provided with the Mensch. The keyboard features a full ASCII keyboard, cursor control keys, and twelve function keys. There is a W65C134 micro-controller in the keyboard unit. It scans the keyboard and communicates serially with the CPU module of the Mensch Computer.

Figure 10
Mensch Computer
Keyboard

Connecting the Keyboard

The keyboard end of the cable is permanently wired to the keyboard. The detachable end uses modular/CMOS and connects to the CPU module through the keyboard (marked: "KYBD") serial port connector. This corresponds to serial port #0 on the W65C265 micro-controller.

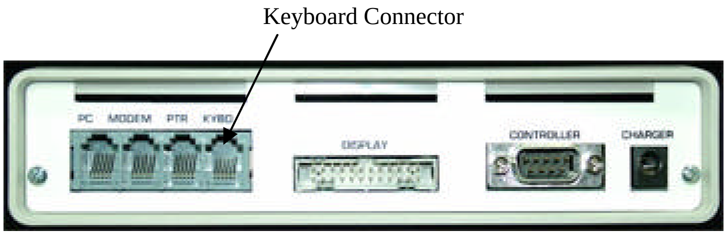

Figure 11
Mensch Computer CPU Module
Rear Panel: Keyboard Connector

The keyboard provided with the Mensch is specific to this product. Even though the keyboard does attach to a serial port of the CPU module, this is not a *PC-compatible* keyboard. Attempting to use a non-Mensch keyboard may cause damage to the unit or to the Mensch.

Programming Support

The Firmware provides several library subroutines for programmers to use when accessing the Mensch keyboard.

Keyboard Support Subroutines
_RETRIEVE_KEYBOARD_STATUS
_CONTROL_KEYBOARD_PORT
_GET_KEYBOARD_CHARACTER
_SEND_BYTE_TO_KEYBOARD
_SELECT_COMMON_BAUD_RATE

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Those library subroutines which support generalized input/output streams can communicate with the keyboard. Refer to the description of CONTROL_INPUT for specific details.

Keyboard Alternatives

Developers may choose the Mensch as a prototyping platform for product applications which do not require a full ASCII keyboard. These configurations may choose to use serial port #0 for other purposes. The default configuration for the keyboard serial port is: 9600 baud, 8 data bits, no parity, and 1 stop bit. (Refer to: **Serial Port Programming Considerations** for more information.)

WARNING!
Developers should note that while the W65C265 micro-controller has four serial ports, it allows only two baud rate generators. In the Mensch Computer configuration, the Modem port has an independent user-selectable baud rate. The other three serial ports, *including the keyboard*, are driven by a common oscillator. Changing this source will affect all three ports.

Printer

The Mensch's serial printer port and associated firmware have been tested with several EPSON-compatible printers. A special adaptor cable is available for connecting the serial printer to the Mensch CPU module. Refer to the documentation associated with the specific printer for details on usage.

Connecting the Printer module

The printer port on the CPU module (marked: "PTR") corresponds to serial port #1 on the W65C265 micro-controller.

Figure 12
Mensch Computer CPU Module
Rear Panel: Printer Connector

All serial ports on the Mensch CPU module use 6-pin modular connectors. These provide only CMOS logic levels, but can be adapted to virtually any standard serial interface (i.e. 20mA, 60mA, RS-232, RS-422, RS-423, RS-485, ect.). Most popular printers have an RS-232 (DTE) interface when used as a serial device. The special adaptor cable converts logic levels on the modular connector to appropriate RS-232 signals on a DB-25 male connector. This can attach to the DB-25 female connector on the printer's serial interface.

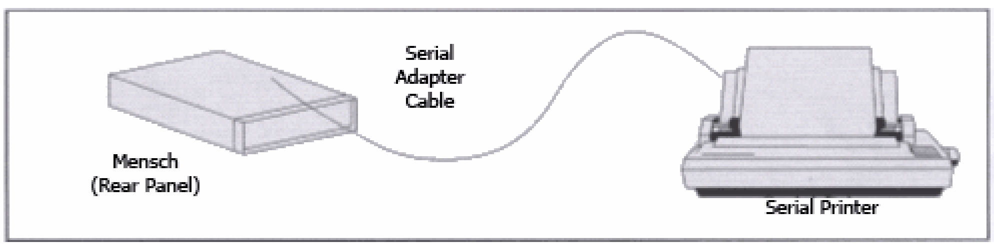

Figure 13
Printer Cabling

The printer cable internally converts CMOS signal levels from the CPU module to the RS-232 signal levels on the printer. This serial printer interface cable is available from WDC, but may require an adaptor. The printer interface cable is connected to the CPU module via the printer (marked: "PTR") serial port. (Refer to: **Mensch Schematics** for more detailed information.)

Connecting Other Printers

Other serial printers may be used with the Mensch. The default configuration for the printer port is: 9600 baud, 8 data bits, no parity, and 1 stop bit.

Programming Support

The Firmware provides several library subroutines for programmers to use when accessing the serial printer.[8]

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Those library subroutines which support generalized input/output streams may be configured to communicate with the printer port. Refer to the descriptions of the : CONTROL_INPUT and CONTROL_OUTPUT subroutines for specific details.

Printer Support Subroutines

	_PtLn
_CONTROL_PRINTER_PORT (ON/OFF)	
	_PtCode
_GET_PRINTER_BYTE (from printer port)	
	_SetText*
_PRINT_BYTE (Send via printer port)	
	_SetGraph *
_RETRIEVE_PRINTER_PORT_STATUS	
	_SetGraph Text*
SELECT_COMMON_BAUD_RATE	
	_PtScreen*

* NOTE: These subroutines are used to support the *print screen* function which automatically copies the LCD memories to the printer port.

Alternative I/O Usage

Developers may choose the Mensch as a prototyping platform for product applications which do not require a serial printer. These configurations may choose to use serial port #1 for other purposes. (Refer to: **Serial Port Programming Considerations** for more information.)

WARNING!
Developers should note that while the W65C265 micro-controller has four serial ports, it allows only two baud rate generators. In the Mensch Computer configuration, the Modem port has an independent user-selectable baud rate. The other three serial ports, *including the printer*, are driven by a common oscillator. Changing this source will affect all three ports.

Another alternate usage involves eliminating serial port #1 entirely and reconfiguring the pin #6 on the W65C265S to use the *pulse-width measurement* (PWM) feature. That is beyond the scope of this manual. Developers should refer to: **W65C265S INFORMATION, SPECIFICATION, AND DATA SHEET** from WDC for specific details. The important point to note, is that using the W65C265S in the Mensch Computer configuration does not exclude this option.

[8] Most of these printer support subroutines rely on the *X-On/X-Off Protocol* when communicating with the printer. This prevents accidentally overflowing the printer's input buffer. Users should be sure that their printer also has been configured for the X-On/X-Off operation.

Modem

Several 2400 baud modems have been tested for use with the Mensch. Any external Hayes-compatible modem which can operate as fast as 2400 baud on an ordinary telephone line should be acceptable. [9] The default configuration for the modem port is: 2400 baud, 8 data bits, no parity, and 1 stop bit. Charging the configuration for the modem port does not affect the other serial ports. (Refer to specific modem documentation for details.)

Connecting Modems

The modem port on the CPU module corresponds to serial port #2 on the W65C265 micro-controller. All serial ports on the Mensch CPU module use 6-pin modular connectors. These provide only CMOS logic levels, but can be adapted to virtually any standard serial interface.

Figure 14
Mensch Computer CPU Module
Rear Panel: Modem Connector

Most external modems have an RS-232 (DCE) interface. The Mensch special adaptor cable converts +5 volt logic levels on the modular connector to appropriate RS-232 signals on a DB-25 male connector. This can attach to the DB-25 female connector on modem's serial interface.

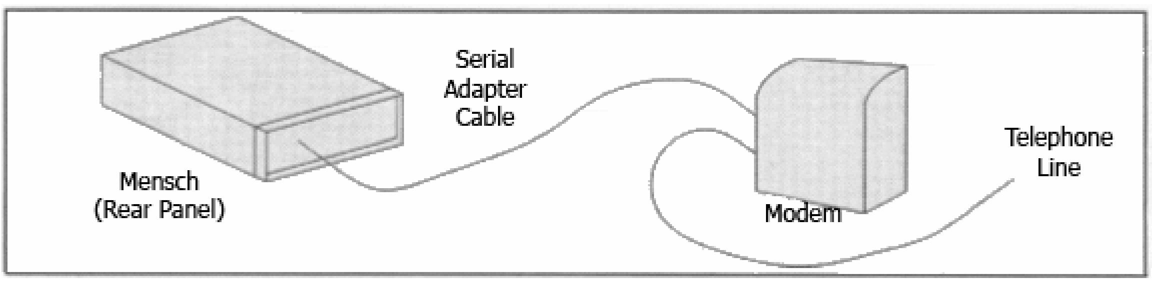

Figure 15
Modem Cabling

The modem cable internally converts CMOS signal levels from the CPU module to the RS-232 signal levels on the modem. This serial modem interface cable is available from WDC, but may require an adaptor. The modem interface cable is connected to the CPU module via the modem (marked: "MODEM") serial port. (Refer to: **Mensch Schematics** for more detailed information.)

9 This aspect: *true Hayes compatibility* is very important. All modems are not necessarily compatible with the Hayes standard command set. Some are partially compatible, supporting only a subset of the commands.

Programming Support

The Firmware provides several library subroutines for programmers to use when accessing the modem.

Modem Support Subroutines	
_CONTROL_MODEM_PORT	GET_MODEM_RESPONSE
_GET_MODEM_BYTE	MODEM_ANSWER
_RETRIEVE_MODEM_PORT_STATUS	MODEM_DIAL
_SELECT_MODEM_BAUD_RATE	MODEM_HANG_UP
_SEND_A_MODEM_BYTE	MODEM_REDIAL
_SEND_MODEM_STRING	

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Those library subroutines which support generalized input/output streams may be configured to communicate with the modem port. Refer to the description of the: CONTROL_INPUT and CONTROL_OUTPUT subroutines for specific details.

WARNING!
Developers should note that while the W65C265 micro-controller has four serial ports, it allows only two baud rate generators. In the Mensch Computer configuration, the Modem port has an independent user-selectable baud rate. The other three serial ports are driven by a common oscillator.

Alternative I/O Usage

Developers may choose the Mensch as a prototyping platform for product applications which do not require a modem. These configurations may choose to use serial port #2 for other purposes. (Refer to: **Serial Port Programming Considerations** for more information.)

PC Link

Information may be transferred between computers via physical media or through data communications. The most common methods of data transfer between systems involve serial links. One serial port on the Mensch has been reserved for this purpose. It has been labeled: “PC” because IBM PC-compatibles comprise the bulk of the personal computers in use.

PC Link Connector

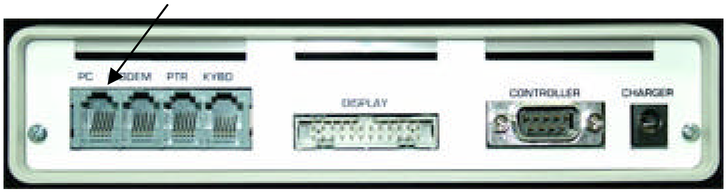

Figure 16
Mensch Computer CPU Module
Rear Panel: PC Link Connector

All serial ports on the Mensch CPU module use 6-pin modular connectors. These provide only CMOS logic levels, but can be adapted to virtually any standard serial interface. (Refer to: **Mensch Schematics** for more detailed information.)

Connecting the Mensch Computer to a PC

The PC port on the CPU module corresponds to serial port #3 on the W65C265 micro-controller. Normal serial ports on PC-compatible computers are configured as *Data Terminal Devices (DTE)* for RS-232 usage.

There are two special adaptor cables which may be used. One converts logic levels on the modular connector to appropriate RS-232 (DCE) signals on a DB-25 female connector. This can attach to the DB-25 male connector on a PC’s *XT-style* serial interface. The other converts logic levels on the modular connector to appropriate RS-232 (DCE) signals on a DB-9 female connector. This attaches to the DB-9 male connector on a PC’s *AT-style* serial interface.

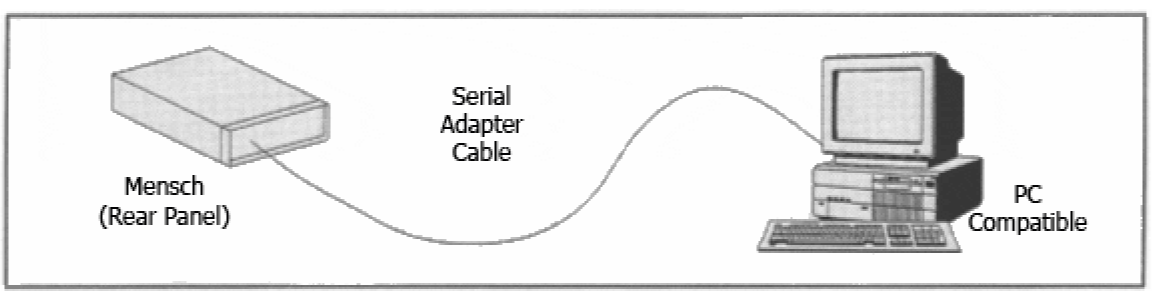

Figure 17
PC Link Cabling

The PC Link cable internally converts CMOS signal levels from the CPU module to the RS-232 signal levels on the IBM-compatible’s serial port. This serial interface cable is available from WDC, but may require an adaptor. (See Figure 12)

The Mensch is shipped with the MenschWorks software which includes PC link support. Terminal emulation software is needed in the PC. Other RS-232 (DTE) peripherals may be interfaced using the PC cables and appropriate software.

Direct Connection To Another Mensch Computer

A special cable is needed even when two Mensch Computers are directly connected via the PC serial link. The *transmit* and *receive* signals must be reversed. Likewise, the *Data Terminal Ready (DTR)* and *Data Set Ready (DSR)* must also be reversed.

Connecting To Other Personal Computers

The PC link may be used to connect to other personal computers, given proper cables and support software. Most popular computers have some terminal emulation capabilities. Depending upon the type of interface, a special cable may be required. Basically, their serial communication ports must provide *transmit data* and *receive data* and handshaking signals which correspond to *Data Set Ready (DSR)* and *Data Terminal Ready (DTR)*. Cables to support RS-232 communication are described in detail in **Mensch Schematics**. These cables are available and may be obtained from WDC.

Programming Support

The Firmware provides several library subroutines for programmers to use when using the PC link.

PC Link Support Subroutines

_CONTROL_PC_PORT

_RETRIEVE_PC_PORT_STATUS

GET_BYTE_FROM_PC

SELECT_COMMON_BAUD_RATE

SEND_BYTE_TO_PC

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Those library subroutines which support generalized input/output streams may be configured to communicate with the PC link port. Refer to descriptions of CONTROL_INPUT and CONTROL_OUTPUT for specific details.

Alternative I/O Usage

Developers may choose the Mensch as a prototyping platform for product applications which do not require interconnection to a personal computer. These configurations may choose to use serial port #3 for other purposes. (Refer to: **Serial Port Programming Considerations** for more information.)

Warning!

Developers should mote that while the W65C265 micro-controller has four serial ports, it allows only two baud rate generators. In the Mensch computer configuration, the Modem port has an independent user-selectable baud rate. The other three serial ports, *including the PC link*, are driven by a common oscillator. Changing this source will affect all three ports.

Controller

The SEGA™ 6-button Arcade Pad game controller has been tested and can be used with the Mensch Computer. When the MenschWorks software accepts controller input, it assumes total compatibility with the SEGA™ 6-button Arcade Pad.

Figure 18
SEGA Game Controller

Connecting the SEGA Game Controller

It is connected through the game controller port on the rear panel of the Mensch Computer.

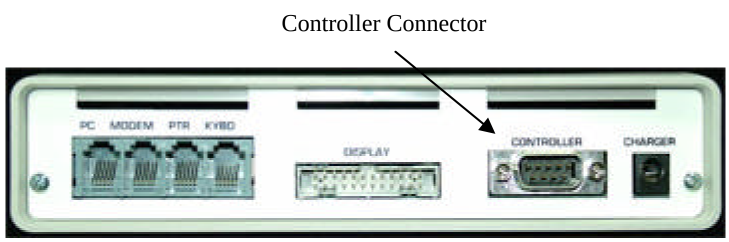

Figure 19
Mensch Computer CPU Module
Rear Panel: Controller Connector

The controller port has been mapped as $00:DFE0 in the address space of the Mensch Computer configuration.

Connecting Other Game Controllers

Several game controller products are marked as SEGA™ compatible. Usually, this just means that they plug into the same 9-pin connector as the SEGA™ 6-button Arcade Pad. Some of these products, such as the SG™ ProPad have many additional switches or significantly different configurations. If such a controller is attached to the Mensch, the signals on the 9-pin connector can be read. Developers must provide their own software for interpretation.

Programming Support

The Firmware provides some library subroutines for programmers to use when accessing the controller.

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. (Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.)

Controller Support Subroutines

_CONTROL_CONTROLLER_PORT

_GET_CONTROLLER_DATA

_RETRIEVE_CONTROLLER_STATUS

Interpreting Controller Status Codes

There are only nine pins available on the game controller connector. One is used to supply the unit with +5 volts, and another is ground. This leaves only seven pins for everything else.

Pin #	Signal Name	Port Pin Identifier
1	(See Text.)	PB0
2	(See Text.)	PB1
3	(See Text.)	PB2
4	(See Text.)	PB3
5	+5 Volts	
6	(See Text.)	PB4
7	(See Text.)	PB5
8	Ground	
9	(See Text.)	PB6

The most significant bit of the port (PB7) is used as an output to switch the supply voltage to the controller connector.

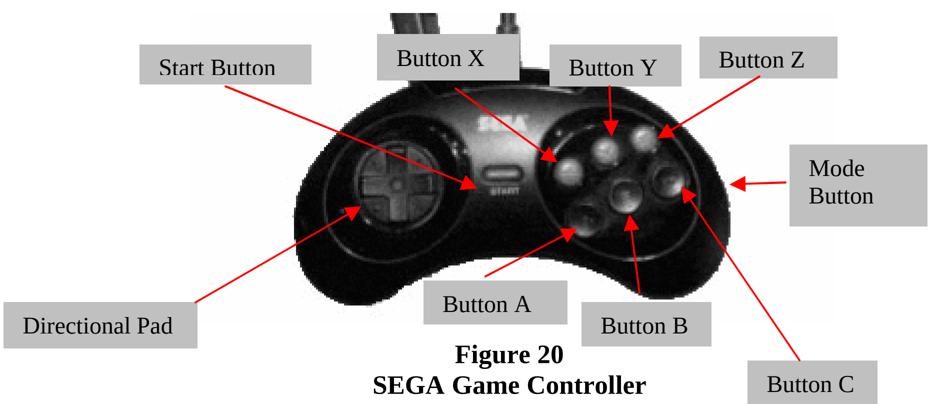

Figure 20
SEGA Game Controller

Switch encoding may be interpreted from the following table:

PB6	PB5	PB4	PB3	PB2	PB1	PB0	Notes
Start	0	A	-	-	Down	Up	
C	1	B	Right	Left	Down	Up	

Alternative I/O Usage

Developers may choose the Mensch as a prototyping platform for product applications which do not require a game controller. These configurations may choose to use this port for other purposes. It should be noted that only seven of the eight bits are normally user definable. The **MSB** will still control the +5 volt supply to the connector. There is a jumper (**JMP4**) which may be used to change this feature and allow the user to define the entire 8-bit port.

Power Subsystem

The Mensch has been designed to operate either from an external power source, or an internal rechargeable battery pack. If both are available and connected at the same time, the battery pack will be recharged from the external supply.

There are indicators on the front panel of the Mensch which show that the external power is available, and also when the batteries are charging.

External Charger/Power Module

Each Mensch is shipped with an external charger module. This UL-approved power adaptor plugs into a standard AC outlet and provides power to the Mensch. It attaches to the rear panel via the "CHARGER" jack.

When the charger/power module is attached and providing power, the lower green LED, labeled: "POWER", will glow. The top red LED, labeled: "CHARGING", will glow when the batteries are charging.

Figure 21
Mensch Computer
Charger/Power Module

Charger Connector

Figure 22
Mensch Computer
Rear Panel: Charger Connector

The external charger/power module is a generic item. If necessary, the user should be able to buy another *off-the-shelf* from local sources. Replacement should be easy, without having to place a special order with WDC. The charger/power module shipped by WDC with each developer's system provides (12 VDC @ 500 mA) more power than the Mensch requires.

Internal Battery Pack

A rechargeable battery pack is installed and connected inside of the Mensch CPU module before shipment. (See Figure 25) It should not need replacement. If the initial checkout procedure indicates that the battery pack is not functional, refer to **Appendix A, Replacing The Battery Pack** for instructions.

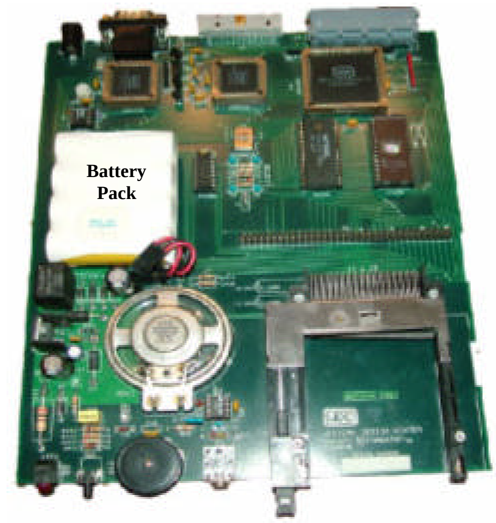

Figure 23
Mensch Computer
Rechargeable Battery Pack

Alternate Power Configurations

The Mensch Computer provides a development platform for applications which may use the W65C265 micro-controller in quite different configurations. Programs may be developed and their logic tested even if their final configuration does not require a keyboard or LCD display. The W65C265 is viable in circuits wherein the supply voltage may be less than 3 volts. Developers should be aware that some elements of the Mensch Computer may not operate at such lower voltages. (Refer to: **Mensch Computer Schematics** for complete details.)

Initial Checkouts

The first step in checking the Mensch involves assembling the components previously described. Only the CPU module, keyboard and display (with appropriate special cables), and power supply are necessary for checkout. The controller, modem and printer (with appropriate special cables), and PC are optional.

Applying Power

Power is available to the W65C265 chip and "slow" clock circuitry whenever a battery is attached to the internal battery connector of the Mensch, or external power is applied. There is no power ON/OFF switch. A special *low-power mode* feature reduces power consumption when the system is inactive. A power-ON reset should occur whenever the battery pack is first attached, or when external power is applied without a battery pack. A *triggered reset* may be initiated manually, by pressing the **RESET** button on the front panel.

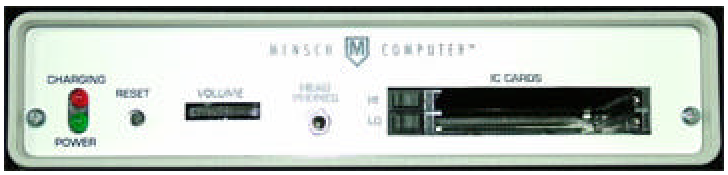

Figure 24
Mensch Computer
Front Panel

If everything is correctly configured and functional, the LCD display should initialize and present the MAIN MENU.

```
03-02-95              MENSCH COMPUTER              12:34:56
         MAIN MENU

>   1)   SETUP MENU
    2)   DEBUG ROUTINES
    3)   TEST MENU
    4)   PCMCIA CARD MENU
    5)   GOTO PROGRAM
    6)   LOADs & DUMPs
    7)   ROM MONITOR
    8)   RUN PCMCIA SHELL

USE CURSOR UP/DOWN & ENTER TO SELECT
```

Figure 25
MAIN MENU

If the MAIN MENU does not appear as expected, confirm that power is available, and press the **RESET** button to force the reset initialization sequence to execute.

System Status Bar

The system status bar is the top line of the LCD display. It will show the current system date and time, and the Mensch Computer title. The current battery status will appear only if the battery needs to be recharged. (See Figure 14)

The *time field* will appear as eight characters on the right end of the system status bar. Time will be displayed in the following format: “hh:mm:ss”, wherein: “hh” represents *hours* on a twenty-four hour clock; “mm” represents *minutes*; and “ss” represents *seconds*. (00:00:00 = midnight / 12:00:00 = noon) If the system is operating properly, the time field should update every second.

The *date field* will appear as eight characters on the left end of the system status bar. The date will be displayed in the following format: “mm/dd/yy” wherein: ‘mm’ represents the *month* number; ‘dd’ represents the *day of the month*; and “yy” represents the least significant two digits of the *year*.

The *title*: “Mensch Computer” will appear in the middle of the system status bar.

The *battery status field* will appear in the middle of the system status bar, overwriting part of the *title*, whenever the batteries need charging. If the system is operational, the title should normally read: “MENSCH COMPUTER”.

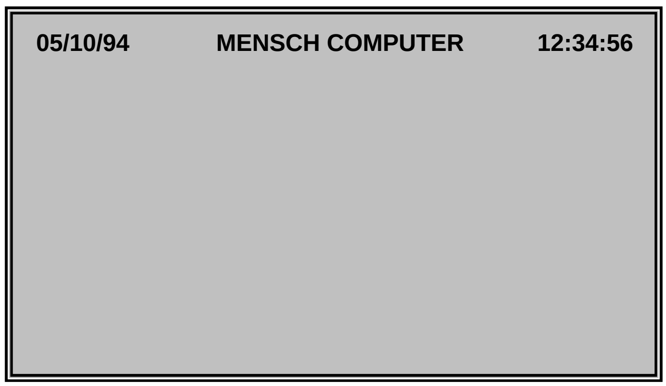

Figure 26
System Status Bar
Battery Condition: Normal (Charged)

The “BATTERY LOW” indication in the middle of the status bar means that the batteries are in need of recharging.

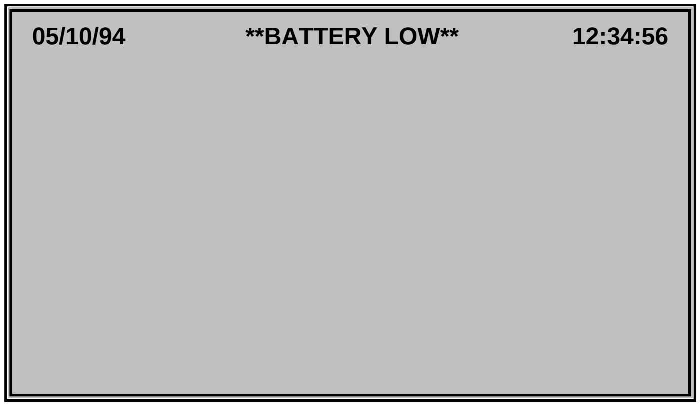

Figure 27
System Status Bar
Battery Condition: LOW (Needs Charging)

Continued operation is not recommended when the batteries are weak. The external charger/power module should be attached and the battery pack should be recharged. If this problem persists, then the battery pack will need to be replaced. Refer to **Appendix A – Replacing The Battery Pack** for further instructions.

Mensch Operating System

The Mensch Operating System performs all necessary system initialization and background support operations. It also provides a menuing system wherein the user may access utilities and application programs.

Initially, a suite of related applications: MenschWorks will be available on an IC card for use with the Mensch Operating System. Its features include a text editor, and a filer.

System Management

The Mensch Operating System resides in EPROM and was designed to support the specific configuration of the Mensch Computer. It uses certain features of the more generic internal Mensch ROM Monitor of the W65C265S chip. The operating system also provides for interaction via the modem, and support for interaction across the PC Link interface.

Reset

Reset may be either a triggered reset or a power-on reset. There is no simple way for the firmware to differentiate which reset occurred. However, some semaphores 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.

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 will not be re-initialized.

Initialization Sequence

The following sequence is performed when power-up (or triggered RESET) occurs, to get the Mensch Computer into a normal operating mode. This description is intended as an overview only; not a specific, line by line analysis of the program code.

INITIAL CONDITION

POWER OFF or *any* previous powered state, including LOW-POWER MODE.

RESET OCCURS! (This may be POWER ON or triggered RESET.)

STAGE #1 (Performed by Mensch Monitor)

"Essential initialization" - Disable interrupts - Reset stack - Clear decimal mode

"Enable External Memory"

Check location $00:8000 - $00:8002 for the string 'WDC'. **If** the string is there; Transfer control to Firmware program in EPROM memory. (JMP $00:8004) **Else** This alternate path is not applicable to this discussion of the Firmware. (Refer to: the Mensch Monitor ROM, Reference Manual for more details.)

STAGE #2 (Performed by the EPROM Mensch Operating System.)

“Miscellaneous initialization”
- Start the fast clock, delay while it becomes stable, and then switch to fast clock.
- Initialize the RAM interrupts vectors.
- Setup the 1 second time-of-day clock interrupt.
- Setup serial I/O buffers & pointers in RAM.

Check the Time-of-Day clock checksum.
If the clock checksum has been corrupted;
- Reset the Time-of-Day clock.
- Reset the baud rate counters for the serial ports to their default values.
- Set up the control ports of the serial UARTs.

Enable interrupts

Construct the initial MAIN MENU display.

Wait for operator input, build secondary menus, and perform requested operations.

STAGE #3 (Performed by application software.)

Configure I/O as desired for application software.

Time-Of-Day Clock/Calendar

The Mensch Computer includes a time-of-day clock feature which operates even when the system is in low-power mode. The date and time are typically displayed, and updated in the top corners of the LCD screen. The time-of-day clock may also be set and read by application programs.

Alarm Function

A built-in alarm function uses the time-of-day clock. The user may set or check the alarm from the keyboard via utility programs, or within software applications via library subroutines. When the alarm times out, the firmware will beep the speaker until the *space bar* on the keyboard is pressed.

Programming Support

The Firmware provides library subroutines for programmers to use when accessing the time-of-day clock/calendar and alarm features.

Clock/Calendar/Alarm Support Subroutines
READ_DATE
SET_DATE
READ_TIME
SET_TIME
SET_ALARM
RESET_ALARM
READ_ALARM
GET_ALARM_STATUS

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

Power Management

The power management functions of the Mensch fall into two categories: (1) Battery Monitoring/Status Display and (2) Low-Power Mode Support.

Battery Monitoring

The condition of the battery pack in the Mensch is constantly monitored by the hardware. Background functions in the Firmware regularly check and display the battery status. If the battery pack is adequately charged, the title: "MENSCH COMPUTER" will appear normally in the middle of the Status Display Line.

If the battery condition is weak enough to jeopardize continued operation of the Mensch Computer, the indication: "BATTERY LOW" will appear in the middle of the Status Display Line, replacing the title. When this situation occurs, operation should be suspended until the battery pack can be recharged.

Low-Power Mode

When the Mensch keyboard is inactive for several minutes, the system will switch over to low-power mode. The LCD display will blank out. This action is taken to reduce power consumption and extend the battery life. The Mensch Computer may be reactivated from low-power mode by RESET, or by pressing any key on the keyboard.

Basically, the Mensch Computer enters low-power mode by performing the following sequence:

- Shut down all interrupts (except Time-Of-Day clock.)
- Clear any pending interrupts.
- Reset the stack to ($00:)01FF.
- Enable the power down routine.
- Switch to the slow (default) clock and then shut off the fast clock.
- Configure I/O ports to inputs.

The power down routine will service the time-of-day interrupt.

The support program for low-power mode resides in the on-board RAM and ROM of the W65C265 micro-controller. Other circuitry on the board may be shut down, but power should not be removed from the W65C265 chip itself. These background operations in the Firmware maintain the time-of-day clock. The support software for this resides in the masked ROM of the W65C265 chip. It can continue to operate even in low-power mode.

If a physical reset occurs while the system is in low-power mode, the normal reset initialization sequence will be performed.

Voltage Detection Circuitry

The Mensch hardware includes a *voltage detection circuit* which may be monitored by software. Application software may check to determine whether the system is operating off power from batteries or the external charger/power module. If the system is using batteries alone, the condition of the batteries may be monitored to avoid problems.

Programming Support

The Firmware provides a library subroutine for programmers to use when checking the *voltage detection status*: CHECK_VOLTAGE.

Another library subroutine allows the user software to place the system in low-power mode. It is: ENGAGE_LOW_POWER_MODE.

Custom applications may only use some of the features and elements of the Mensch. These programs may selectively manipulate power switching controls over subsystems. Several Firmware library subroutines have been provided for this purpose.

Power Management Support Subroutines
CONTROL_CONTROLLER_PORT
CONTROL_DISPLAY
CONTROL_KEYBOARD_PORT
CONTROL_MODEM_PORT
CONTROL_PC_PORT
CONTROL_PRINTER_PORT
CONTROL_SPEAKER_AMP

Programmers may use these and other library subroutines to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information.

Menuing Support

The Firmware program of the Mensch is intended to be *User Friendly*. Whenever possible, the mode changing commands require only a single keypress. Some modes may display a menu of additional options. The user may target a specific menu item using the *up-arrow* (↑) and *down- arrow* (↓) keys, or the controller left button. Any response will not be final until the **ENTER** key has been pressed. If the user chooses to cancel a response, this may be accomplished by pressing the **ESC** (Escape) key, before pressing **ENTER**.

Programming Support

The Firmware provides a library subroutine for programmers to use when developing their own menus for the LCD screen.

Menuing Support Subroutines
MENU_SETUP
MENU_POINT
DISP_LCD_HEADER
TIME_DATE_CHK
CHECK_YN
GET_HILO
GET_BIN_NUM

Programmers may use these and other library subroutines to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information.

Main Menu

The MAIN MENU display will appear upon system reset. This menu offers access to a variety of setup, test, and utility functions.

```
03-02-95             MENSCH COMPUTER            12:34:56
          MAIN MENU

>   1)    SETUP MENU
    2)    DEBUG ROUTINES
    3)    TEST MENU
    4)    PCMCIA CARD MENU
    5)    GOTO PROGRAM
    6)    LOADs & DUMPs
    7)    ROM MONITOR
    8)    RUN PCMCIA SHELL

USE CURSOR UP/DOWN & ENTER TO SELECT
```

Figure 28
MAIN MENU

The MAIN MENU also allows the user to execute the PCMCIA Shell program, the MenschWorks application example, or user-supplied application programs on IC cards.

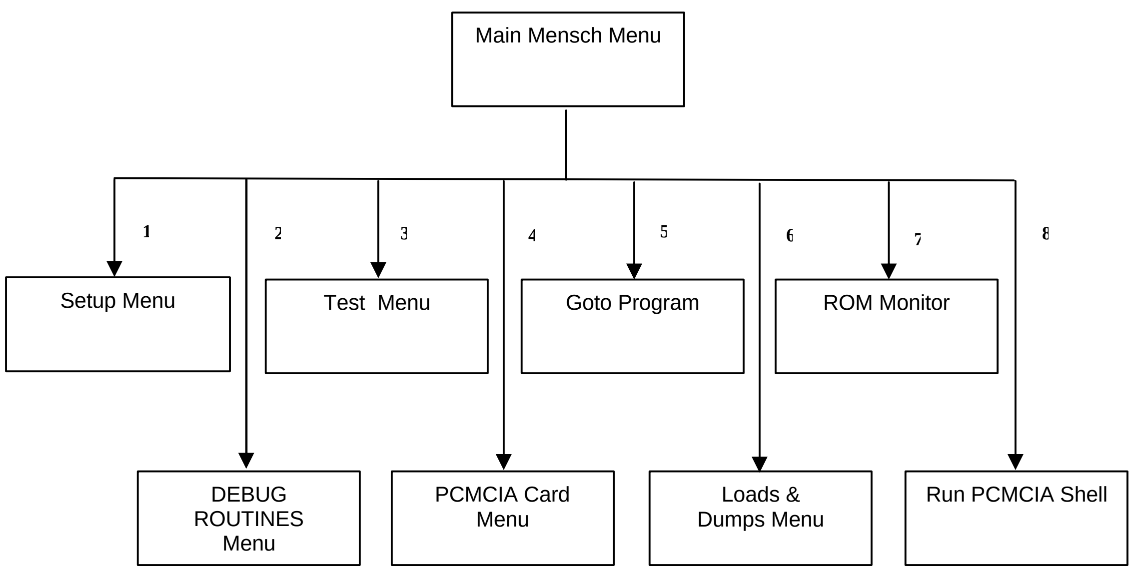

Figure 29
MAIN MENU Tree

1) SETUP MENU (MAIN MENU Option)

The SETUP MENU is used to configure some of the key components of the Mensch.

```
06-18-94            MENSCH COMPUTER            12:34:56
                    SETUP MENU

>  1)     DISPLAY & SET TIME
   2)     DISPLAY & SET DATE
   3)     DISPLAY & SET ALARM
   4)     Reserved
   5)     Reserved
   6)     INITALIZE MODEM
   7)     VIEW MODEM SETUP
   8)     RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SELECT
```

Figure 30
SETUP MENU

This is the menu to select when the system date and time need to be reset. It also provides a means of setting or disabling the system alarm function.

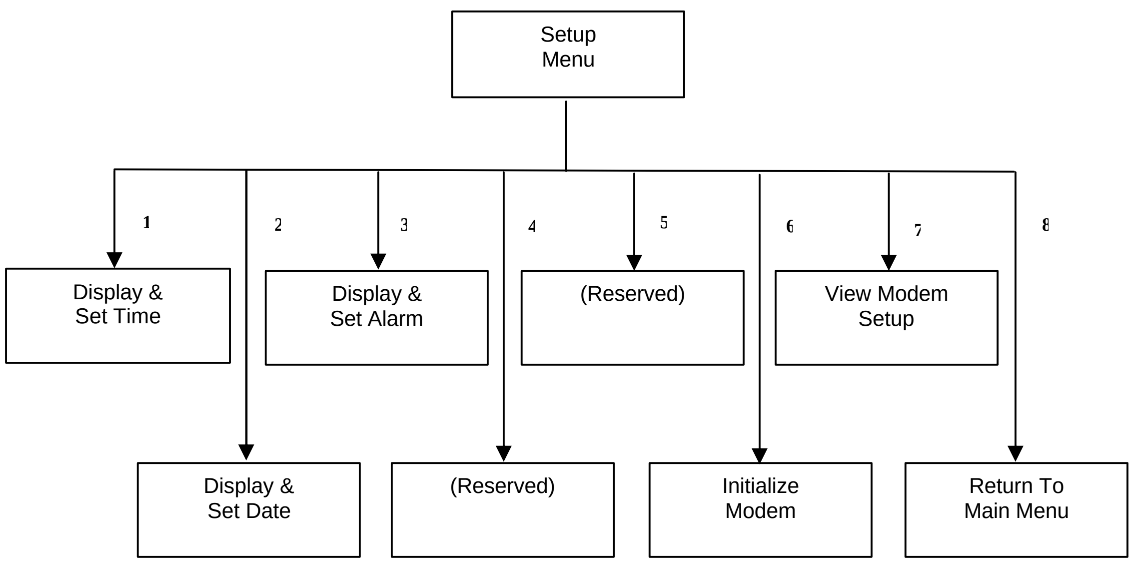

Figure 31
SETUP MENU Tree

In addition, there are two items by which the user may view the current setup options on a Hayes-compatible modem, or send it a predefined initialization sequence.

1) DISPLAY & SET TIME (SETUP MENU Option)

This SETUP MENU item allows the user to enter a new time value for the system time-of-day clock/calendar feature of the Mensch.

```
11-14-94           MENSCH COMPUTER            12:34:56
           SETUP MENU

>   1)     DISPLAY & SET TIME
    2)     DISPLAY & SET DATE
    3)     DISPLAY & SET ALARM
    4)     Reserved
    5)     Reserved
    6)     INITALIZE MODEM
    7)     VIEW MODEM SETUP
    8)     RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SELECT
           Time is    12:34:56
Time Format = HH:MM:SS
```

Figure 32
DISPLAY & SET TIME

The *time* entry format (HH:MM:SS) is shown above, wherein: HH = hours

MM = Minutes

SS = Seconds

2) DISPLAY & SET DATE (SETUP MENU Option)

This SETUP MENU item allows the user to enter a new date value for the system time-of-day clock/calendar feature of the Mensch Computer.

```
03-02-95           MENSCH COMPUTER            12:34:56
           SETUP MENU

    1)     DISPLAY & SET TIME
>   2)     DISPLAY & SET DATE
    3)     DISPLAY & SET ALARM
    4)     Reserved
    5)     Reserved
    6)     INITALIZE MODEM
    7)     VIEW MODEM SETUP
    8)     RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SELECT
           Date is    03-02-95
Date Format = MM-DD-YY
```

Figure 33
DISPLAY & SET DATE

The *date* entry format (MM-DD-YY) is shown above, wherein: MM = Month

DD = Day of Month

YY = Year (1994 = 94)

3) DISPLAY & SET ALARM (SETUP MENU Option)

This SETUP MENU item allows the user to disable or set the built-in alarm feature of the Mensch Computer.

```
03-02-95            MENSCH COMPUTER            12:34:56
           SETUP MENU

     1)    DISPLAY & SET TIME
     2)    DISPLAY & SET DATE
>    3)    DISPLAY & SET ALARM
     4)    Reserved
     5)    Reserved
     6)    INITALIZE MODEM
     7)    VIEW MODEM SETUP
     8)    RETURN TO MAIN MENU

Alarms is OFF / DOWN & ENTER TO SELECT
Turn Alarm On  Y/N?
```

Figure 34
DISPLAY & SET ALARM

If the alarm function is not active when this item is selected, then the prompt: "Turn Alarm On Y/N" will appear. A response of: 'N' ("NO") will cancel this operation and return to the SETUP MENU normal display. If the user responds: 'Y' ("YES"), then another prompt will appear:

```
06-18-94            MENSCH COMPUTER            12:34:56
           SETUP MENU

     1)    DISPLAY & SET TIME
     2)    DISPLAY & SET DATE
>    3)    DISPLAY & SET ALARM
     4)    Reserved
     5)    Reserved
     6)    INITALIZE MODEM
     7)    VIEW MODEM SETUP
     8)    RETURN TO MAIN MENU

Alarms is OFF / DOWN & ENTER TO SELECT
Turn Alarm On  Y/N?
Alarm Format = HH:MM:SS
```

Figure 35
Alarm Entry Prompt

The alarm entry format (HH:MM:SS) is shown above, wherein: HH = Hours
MM = Minutes
SS = Seconds

When the alarm conditions are satisfied, the firmware will begin beeping the speaker. The alarm sound may be terminated by pressing the *space bar* on the keyboard.

4) Reserved and

5) Reserved (SETUP MENU Option)

These options are not yet assigned. They have been reserved for future use. When either of these menu items is selected, the following screen will appear:

```
06-18-94          MENSCH COMPUTER          12:34:56

                Function not available !
```

Figure 36
SETUP MENU Options: #4 & #5

6) INITIALIZE MODEM (SETUP MENU Option)

This SETUP MENU item allows the user to initialize the modem. The following screen appears when this item is selected:

```
06-18-94          MENSCH COMPUTER          12:34:56
```

Figure 37
INITIALIZE MODEM
1st Screen Blinks Quickly

It will only appear for a moment, to be replaced by another:

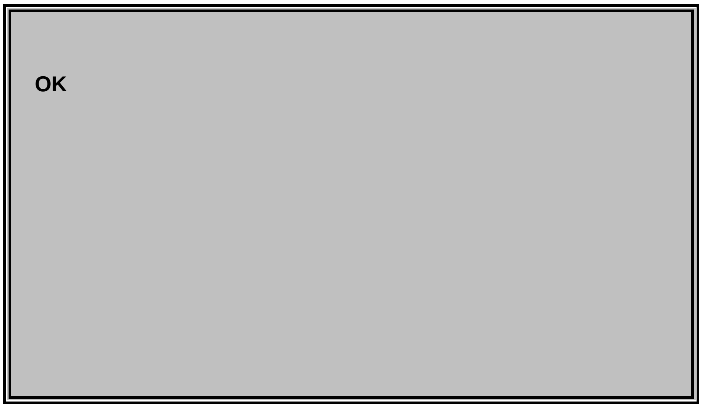

Figure 38
INITIALIZE MODEM
2nd Screen (Approx. 5 sec.)

This screen will remain while the modem initialization commands are being sent. Under normal conditions, this should only last about five seconds.

After about five seconds, the modem ID, read from the modem, will appear:

Figure 39
INITIALIZE MODEM
3rd Screen w/Modem ID

This third screen showing the modem ID will remain until *two* keys are pressed. Control will be returned to the SETUP MENU.

If this option is selected, but no modem is attached, then the following screen will be displayed:

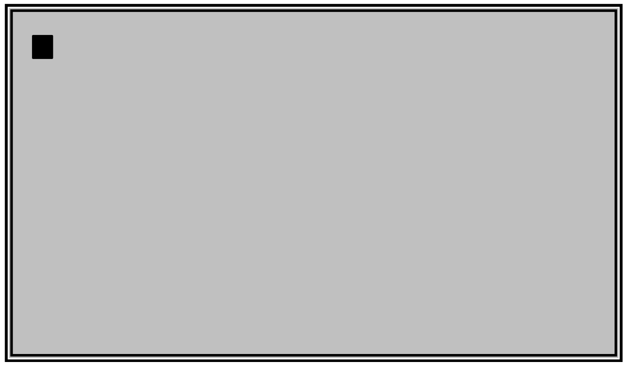

Figure 40
INITIALIZE MODEM

Pressing **ESC** (Escape) twice will return control to the SETUP MENU.

7) VIEW MODEM SETUP (SETUP MENU Option)

This SETUP MENU item allows the user to view the modem setup information. It does this by reading it back directly from the modem. The following screen will appear when this item is selected:

```
ACTIVE PROFILE:
 B1  E0  L2  M1  Q0  V1  X4  Y0  &C1  &D2  &G0  &J0
&L0  &P0  &Q0  &R0  &S0  &X0  &Y0
S00:002  S01:000  S02:043  S03:013  S04:010
S05:008  S06:002  S07:030
S08:002  S09:006  S10:014  S12:050  S14:00H
S16:00H  S8:000  S21:30H
S22:76H  S23:17H  S25:005  S26:001  S27:40H

STORED PROFILE 0:
B1  E0  L2  M1  Q0  V1  X4  Y0  &C1  &D2  &G0  &J0
&L0  &P0  &Q0  &R0  &S0  &X0
S00:002  S14:88H  S18:000  S21:30H  S22:76H
 -----  MORE  -----
```

Figure 41
VIEW MODEM SETUP
1st Screen

Pressing any key will display the second page of modem setup information.

```
S23:17H  S25:005  S26:001
S27:40H

STORED PROFILE 1:
B1  E1  LZ  M1  Q0  V1  X4  Y0  &C1  &D2  &G0  &J0
&L0  &P0  &Q0  &R0  &X0
S00:000  S14:AAH  S18:000  S21:30H  S22:76H
S23:17H  S25:005  S26:001
S27:40H

TELEPHONE NUMBERS:
&Z0=
&Z1=
&Z2=
&Z3=
-----  MORE  -----
```

Figure 42
VIEW MODEM SETUP
2nd Screen

If the modem setup information fills or exceeds two screens, additional screens will be available. This is indicated by: "--MORE--"on the bottom of the display. Pressing any key will cause the next screen to appear.

The modem setup information will usually be terminated by: "OK" which indicates that the modem is ready for another command.

```
OK
```

Figure 43
VIEW MODEM SETUP
3rd Screen

The final screen will remain for about three seconds and then return control to the SETUP MENU.

---

8) RETURN TO MAIN MENU (SETUP MENU Option)

Pressing **ESC** (Escape) or selecting this option returns control to the MAIN MENU.

2) DEBUG MENU (MAIN MENU Option)

This menu provides a list of options to allow users to directly manipulate the environment in which programs execute. These options include dumping and/or modifying memory, setting breakpoints to interrupt execution, and displaying the register contents of application programs. When used with other features such as loading programs, or executing from an address, or even accessing the Mensch ROM Monitor, the debug functions are powerful tools.

```
06-18-94          MENSCH COMPUTER          12:34:56
             DEBUG MENU

>   1)       Alter Memory
    2)       Display Registers
    3)       SET Breakpoint
    4)       FILL Memory
    5)       DUMP to Screen
    6)       ASCII Screen Dump
    7)       Reserved
    8)       RETURN TO MAIN MENU

USE CURCOR UP / DOWN & ENTER TO SELECT
```

Figure 44
DEBUG MENU

The **ESC** (Escape) key may be used to cancel these menu operations and return to the MAIN MENU.

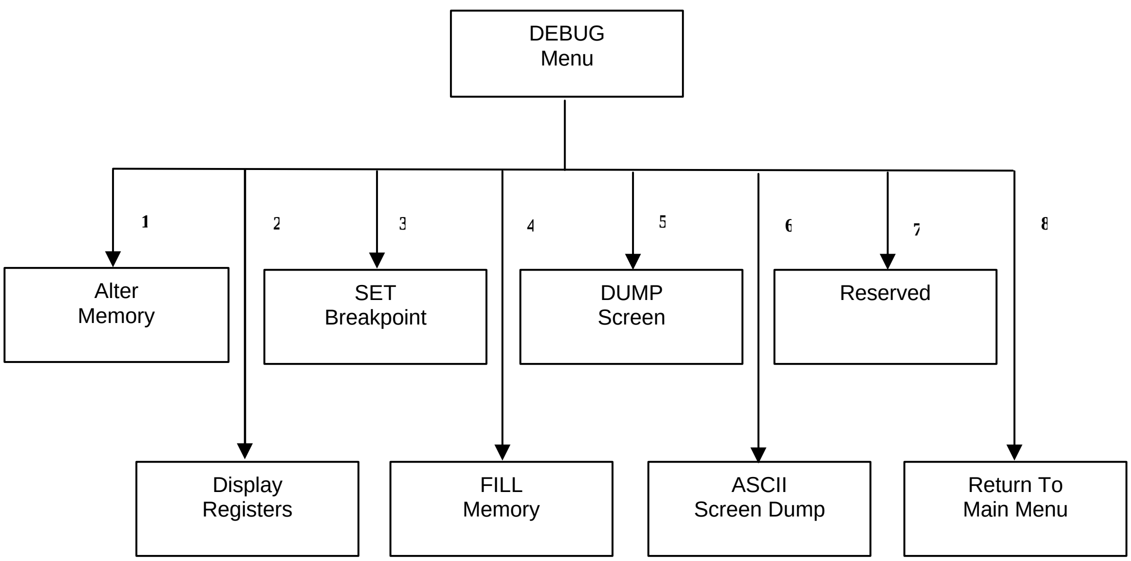

Figure 45
DEBUG MENU Tree

1) ALTER MEMORY (DEBUG MENU Option)

This DEBUG MENU item allows the user to change the contents of RAM locations. First, a prompt will appear requesting the first address to be modified. Any valid address may be entered, but only RAM locations can be changed.

```
Enter Address   BB:AAAA
```

Figure 46
Alter Memory Prompt

When a valid address response has been entered, then the currents of sixteen locations, beginning at the specified address, will be displayed. The cursor will be positioned below the first location, and the user may begin entering new data into consecutive locations.

```
Address   4    5    6    7    8    9    A    B
00:1234   26   85   20   0C   00   00   01   40
00:123C   41   42   35   20   44   46   20   42
00:1234
```

Figure 47
Alter Memory Display

The cursor will advance by spacing over the next field as each value is typed. Backspacing is supported, however only the **ENTER** key will exit and return control to the DEBUG MENU. The **ESC** (Escape) key may behave unpredictably, and should not be used.

2) DISPLAY REGISTERS (DEBUG MENU Option)

This DEBUG MENU item allows the user to view the working registers available to application programs. When an executing application encounters a *breakpoint*, a **BRK** instruction, it returns control to the operating system. First, it will copy the contents of all registers into some pseudo-registers in RAM. These working registers will be used to restore the real registers when an application program is started. It is these RAM locations which are viewed by this menu selection.

```
PCntr     Acc      Xreg     Yreg     Stack
34:0001   00 01    0A 34    00 01    0A 34
   DirRg   F    DBK
   00 01  DF 34

Status Reg
M   V   M   X   D   I   Z   C
1   1   0   1   1   1   1   1
```

Figure 48
Display Registers

When debugging an application the user may set a breakpoint at a key position in the program, and then examine the register contents when that breakpoint is encountered. Breakpoints may be set from option #3 of the DEBUG MENU.

Any key may be pressed to return to the DEBUG MENU, after the registers have been displayed.

3) SET BREAKPOINT (DEBUG MENU Option)

This DEBUG MENU item allows the user to set a breakpoint at a specific location. Basically, this involves storing a **BRK** instruction ($0000) at the target location.

```
Enter Address    BB:AAAA
```

Figure 49
Set Breakpoint

When execution resumes, the program may attempt to execute the target instruction. The **BRK** instruction will be executed instead. Control will be returned to the Mensch Operating System.

The **ESC** (Escape) key may be used to cancel this operation and return to the DEBUG MENU, instead of entering an address at the prompt.

4) FILL MEMORY (DEBUG MENU Option)

This DEBUG MENU item allows the user to fill a block of memory with a constant value.

```
Enter Lowest Address          BB:AAAA
```

Figure 50
FILL Memory: First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address      BB:AAAA      00:0000
Enter Highest Address     BB:AAAA
```

Figure 51
FILL Memory: Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address      BB:AAAA    00:S000
Enter Highest Address     BB:AAAA    00:SFFF
Enter Byte in HEX
```

Figure 52
FILL Memory: Third Prompt

Press any key to return to the SETUP MENU screen.

5) DUMP TO SCREEN (DEBUG MENU Option)

This DEBUG MENU item allows the user to examine a block of memory by dumping its contents, appropriately formatted in hexadecimal, to the LCD screen.

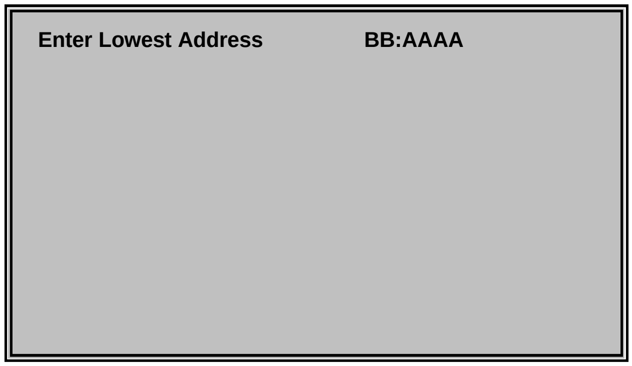

Figure 53
DUMP to Screen: First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

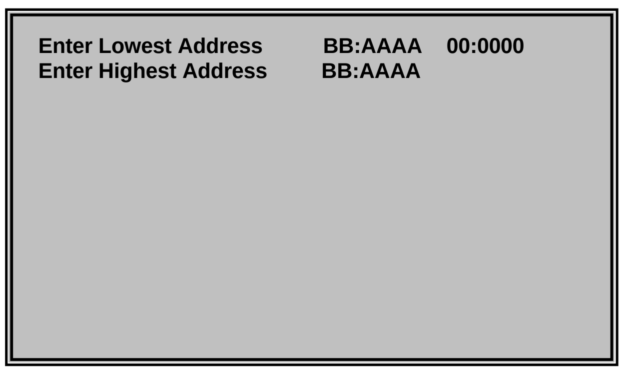

Figure 54
DUMP to Screen: Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will display the first ninety-six locations of the memory block. The address and eight locations will be displayed on each line in hexadecimal format.

```
1   Address   0   1   2   3   4   5   6   7
2
3   00:0000   5C  D3  CE  00  5C  E1  CE  00
4   00:0008   5C  00  82  00  5C  00  82  00
5   00:0010   5C  00  82  00  5C  AB  C2  00
6   00:0018   5C  D1  ED  00  5C  00  82  00
7   00:0020   5C  21  81  00  5C  00  5C  00
8   00:0028   00  00  00  00  00  00  00  00
9   00:0030   00  00  00  00  03  00  03  00
10  00:0038   04  04  84  04  00  04  00  04
11  00:0040   03  04  81  01  00  00  00  00
12  00:0048   0D  D9  00  00  48  D9  00  04
13  00:0050   DF  D9  01  20  4B  C6  00  04
14  00:0058   6C  02  00  05  00  A7  0F  00
15
16
```

Figure 55
DUMP to Screen: Data Display

Only twelve lines may be displayed at a time. The user may press any key to display successive screens of data, until reaching the end of the selected memory block. After the last location of the memory block has been displayed, any keypress will return to the DEBUG MENU.

6) ASCII Screen Dump (DEBUG MENU Option)

This DEBUG MENU item allows the user to examine a block of memory by dumping its contents, appropriately formatted in ASCII, to the LCD screen.

```
Enter Lowest Address      BB:AAAA
```

Figure 56
ASCII Screen Dump: First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address      BB:AAAA 00:0000
Enter Highest Address     BB:AAAA
```

Figure 57
ASCII Screen Dump: Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will echo the characters as they are sent to the PC Link serial port. The LCD screen may be difficult to read because the data is in S28 record format.

```
00:000 \ * * * \ * * * \ * * * \ * * *
00:0010 \ * * * \ * * * \ * * * \ * * *
00:0020 \ . * * * * * * * * * * * * * *
00:0030 * * * * * * * * * * * * * * * *
00:0040 * * * * * * * * * * * K * * *
00:0050 * * * * K * * * 1 * * * * * * *
00:0060 * * * * * * * * * * * * * * * *
00:0070 * 7 * * * * B * K * * * K * * *
00:0080 K * * * K * * * K * * * K * * *
00:0090 K * * * K * * * K * * * K * * *
00:00A0 K * * * * * * * K * * * K * * *
00:00B0 K * * * K * * * K * * * K * * *
```

Figure 58
ASCII Screen Dump: Data Display

Press any key to return to the DEBUG MENU.

7) Reserved (DEBUG MENU Option)

This option is not yet assigned. It has been reserved for future use. When this menu item is selected, the following screen will appear.

```
08-18-94          MENSCH COMPUTER          12:34:56
               Function not available  !¦
```

Figure 59
DEBUG ITEM #7

Pressing any key will return to the DEBUG MENU.

8) RETURN TO MAIN MENU (DEBUG MENU Option)

Pressing **ESC** (Escape) or selecting this option on the DEBUG MENU returns control to the MAIN MENU.

3) TEST MENU (MAIN MENU Option)

The TEST MENU is used to conduct simple checks on some of the key elements of the Mensch Computer.

```
06-18-94          MENSCH COMPUTER          12:34:56
            TEST MENU

>   1)      KEYBOARD TEST
    2)      DTMF TEST
    3)      MODEM TEST
    4)      TEST PRINTER
    5)      GAME CONTROLLER
    6)      SOFTWARE VERSION
    7)      GRAPHICS TEST
    8)      RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SELECT
```

Figure 60
TEST MENU

Besides testing the keyboard, audio circuitry, modem, printer, and game controller, this menu allows the user to review what version and date are in the EPROM firmware.

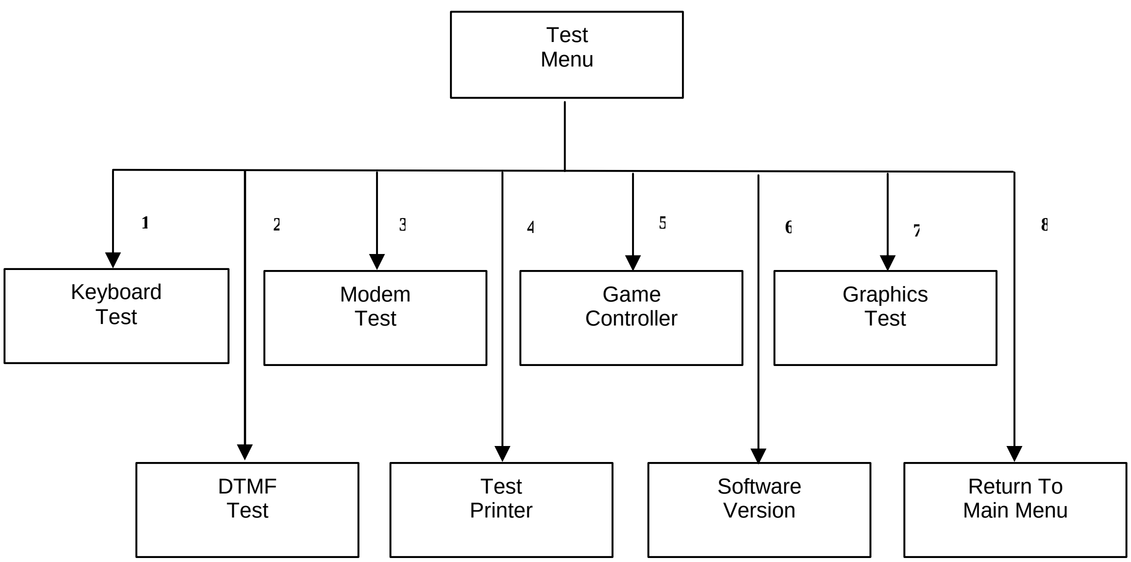

Figure 61
Test Menu Tree

1) KEYBOARD TEST (TEST MENU Option)

This TEST MENU item allows the user to check the serial keyboard path.

```
KEYBOARD TEST _ use ESC key to exit
```

Figure 62
KEYBOARD TEST

When the above screen appears, the program will echo any key pressed by the user. The **ESC** (Escape) key may be used to cancel this operation and return to the TEST MENU.

If a normally displayable character does not echo properly; reset the system and try again. If the problem persists, then there may be a malfunction in the keyboard module (or alternative source), or the cabling, or the Mensch Computer itself. Further isolation involves replacing the keyboard with a *known good unit*, and retesting.

2) DTMF TEST (TEST MENU Option)

This TEST MENU item will cause a brief burst of sound as DTMF tones are gated through the amplifier to the speaker.

```
06-18-94            MENSCH COMPUTER            12:34:56
                  TEST MENU

      1)      KEYBOARD TEST
      2)      DTMF TEST
      3)      MODEM TEST
>     4)      TEST PRINTER
      5)      GAME CONTROLLER
      6)      SOFTWARE VERSION
      7)      GRAPHICS TEST
      8)      RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 63
TEST MENU – DTMF TEST

The following screen will appear when this item is selected:

```
DTMF TEST – use ESC key to exit
```

Figure 64
DTMF TEST Screen

Pressing any of the keys on the keyboard which corresponds to keys on a telephone pad will generate the appropriate DTMF combination as an audio burst from the speaker. Acceptable keys are: *0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, #, and *.* Refer to the description of the *SEND_DTMF_DIGIT* subroutine for details about which tone pairs are associated with each key. Pressing any other key will return control to the TEST MENU.

3) MODEM TEST (TEST MENU Option)

This TEST MENU item allows the user to interact directly with the modem. The following screen appears when this item is selected:

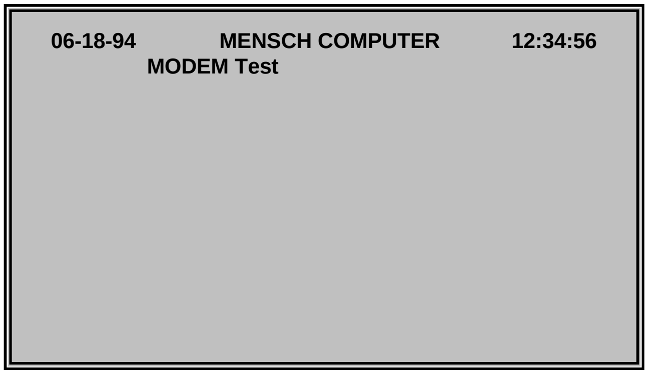

Figure 65
MODEM TEST

While not exactly a test in itself, this selection may be used to command the modem to perform its own internal diagnostics. Hayes-compatible modems typically include the following built-in tests:

Command String	Description
AT&T0	End current test, if there is a test in progress.
AT&T1	Initiate local analog loopback test.
AT&T2	Not used.
AT&T3	Initiate local digit loopback test.
AT&T4	Grant remote digital loopback request from remote mode.
AT&T5	Deny remote digital loopback request from remote mode.
AT&T6	Initiate remote digit loopback test.
AT&T7	Initiate remote digit loopback with self-test.
AT&T8	Initiate local analog loopback with self-test.

Users should consult the documentation for their own modem to determine what other modem tests or commands may also be used with that product.

Pressing **ESC** (Escape) eventually returns control to the TEST MENU. There may be a delay of several seconds before the **ESC** key is recognized.

4) TEST PRINTER (TEST MENU Option)

This TEST MENU item allows the user to test the printer path by sending a repeating stream of character data to the serial printer port. If the communication link is sound, this test should effectively exercise the printer hardware[10]. Serious problems in the printer itself, may manifest themselves when this test is run. If the printer does not print as expected, then check power, cabling, and confirm that the printer is enabled. Control will return to the TEST MENU upon successful completion of this test.

```
06-18-94            MENSCH COMPUTER            12:34:56
              TEST MENU

     1)   KEYBOARD TEST
     2)   DTMF TEST
     3)   MODEM TEST
>    4)   TEST PRINTER
     5)   GAME CONTROLLER
     4)   SOFTWARE VERSION
     5)   GRAPHICS TEST
     6)   RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 66
TEST PRINTER

If there is no printer attached to the port, then the TEST MENU screen will just blink once when this item is selected.

10 Most late model printers have internal diagnostics. If this test does not operate correctly, then the printer self-test should be performed.

5) GAME CONTROLLER (TEST MENU Option)

This TEST MENU item allows the user to test the game controller and its port.

```
06-18-94            MENSCH COMPUTER            12:34:56
           TEST MENU

    1)   KEYBOARD TEST
    2)   DTMF TEST
    3)   MODEM TEST
    4)   TEST PRINTER
>   5)   GAME CONTROLLER
    6)   SOFTWARE VERSION
    7)   GRAPHICS TEST
    8)   RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 67
GAME CONTROLLER

Pressing the SPACE BAR will read the controller port and display any buttons on the controller which are currently down.

```
06-18-94          MENSCH COMPUTER          12:34:56

Start
A
B
C
Right
Left
Down
Up
```

Figure 68
GAME CONTROLLER

Pressing **ESC** (Escape) key will return to the TEST MENU.

6) SOFTWARE VERSION (TEST MENU Option)

This TEST MENU item will display the current version number of the EPROM firmware and also the date and time when it was generated.

```
MENSCH COMPUTER Version 3.01
   ©  Copyright  1994, 1995
Assembled   Mon   Feb   27   12:51:55    1995
```

Figure 69
SOFTWARE VERSION

Pressing any key will return to the TEST MENU.

7) GRAPHICS TEST (TEST MENU Option)

This option demonstrates the features of the LCD in text mode, graphics mode, and combined mode. It begins by drawing several geometric patterns in graphics mode. Then a number sequence ("065535") will repeat in text mode until all character positions are full. Both display modes will be enabled so the combined image will be displayed on the LCD screen.

While the combined image remains on the display, this test will selectively copy the images to the printer. It will first print only the graphics mode memory image, then the combined image, and finally the text.

```
06-18-94              MENSCH COMPUTER             12:34:56
             TEST MENU

     1)    KEYBOARD TEST
     2)    DTMF TEST
     3)    MODEM TEST
     4)    TEST PRINTER
     5)    GAME CONTROLLER
     6)    SOFTWARE VERSION
>    7)    GRAPHICS TEST
     8)    RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 70
GRAPHICS TEST

The test will wait for any key to be pressed before returning to the TEST MENU.

8) RETURN TO MAIN MENU (TEST MENU Option)

Pressing **ESC** (Escape) or selecting this option in the TEST MENU returns control to the MAIN MENU.

4) PCMCIA CARD MENU (MAIN MENU Option)

This MAIN MENU item allows the user to display a menu of available choices from the specified IC card. Usually, these menu items will be programs which the user may select for execution. Initially, the user will be prompted to select which IC card directory should be displayed.

```
Card  =  HI  or  LO :
```

Figure 71
PCMCIA CARD MENU PROMPT

After the user has selected an IC card, the menu for that card, if one exists, will appear.

```
06-18-94        MENSCH COMPUTER      12:34:56
          PROGRAM LIST

>     MENSCH WORKS   Applications
```

Figure 72
PCMCIA CARD MENU

The user may initiate the execution of the MENSCH WORKS application program in the above example by pressing the **ENTER** key. If the menu contained several items, then the vertical arrow keys would be used to position before selecting the program for execution.

If the card has been properly formatted, using the PCMCIA shell, but no programs is in the menu, the list will be empty:

```
06-18-94        MENSCH COMPUTER        12:34:56
          PROGRAM  LIST
```

Figure 73
PCMCIA CARD MENU
w/No Programs

If no menu information can be located on the specified IC card, or no card has been inserted, the following screen will appear:

```
06-18-94         MENSCH COMPUTER     12:34:56
          PROGRAM LIST
>

    Improper   or   Missing   Card
```

Figure 74
PCMCIA CARD MENU ERROR

The **ESC** (Escape) key may be used to cancel this operation and return to the MAIN MENU.

5) GOTO PROGRAM (MAIN MENU Option)

This MAIN MENU item will allow the user to specify any address desired and transfer execution to it.

```
06-18-94            MENSCH COMPUTER            12:34:56
          MAIN  MENU

     1)    SETUP MENU
     2)    DEBUG ROUTINES
     3)    TEST MENU
     4)    PCMCIA CARD MENU
>    5)    GOTO PROGRAM
     6)    LOADs & DUMPs
     7)    ROM MONITOR
     8)    RUN PCMCIA SHELL

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 75
GOTO PROGRAM

Obviously, control should only be transferred to valid executable programs. Otherwise, the consequences are unpredictable.

6) LOAD & DUMP MENU (MAIN MENU Option)

The LOAD & DUMP MENU is used to interact with another computer via the PC Link serial port. Typically, this will involve transferring S28 records between systems.

```
06-18-94          MENSCH COMPUTER          12:34:56
             LOAD & DUMP MENU

>   1)       KEYBOARD TEST
    2)       DTMF TEST
    3)       MODEM TEST
    4)       TEST PRINTER
    5)       GAME CONTROLLER
    6)       SOFTWARE VERSION
    7)       GRAPHICS TEST
    8)       RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 76
LOAD & DUMP MENU

This menu also allows the user to examine sections of memory by dumping their contents, appropriately formatted, to the LCD screen or printer serial port.

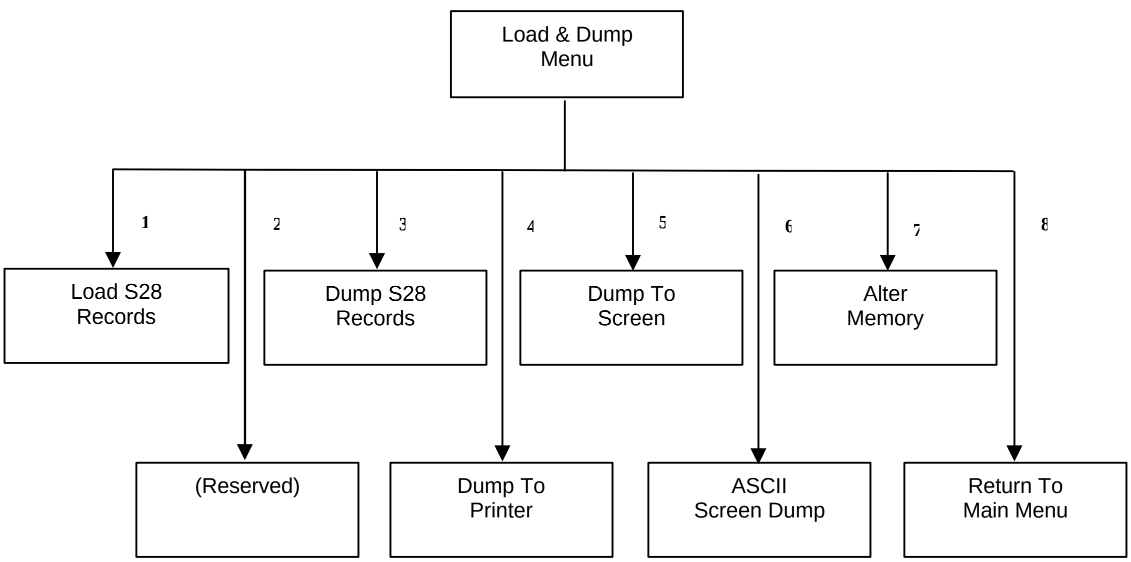

Figure 77
LOAD & DUMP Menu Tree

1) LOAD S28 Records (LOAD & DUMP MENU Option)

This LOAD & DUMP MENU item allows the user to load S28 records into memory from a host computer via the PC Link serial port.

```
06-18-94          MENSCH COMPUTER        12:34:56
                LOAD & DUMP MENU

>   1)      LOAD S28 Records
    2)      Reserved
    3)      DUMP S28 Records
    4)      DUMP to PRINTER
    5)      DUMP to Screen
    6)      ASCII Screen Dump
    7)      ALETER MEMORY
    8)      RETURN TO MAIN MENU

USE CURSOR UP/DOWN & ENTER TO SEELCT
```

Figure 78
LOAD S28 Records #1

When this option is selected, the Mensch is ready to accept “S28” records via the PC link. The screen will be cleared and the cursor will move down several lines.

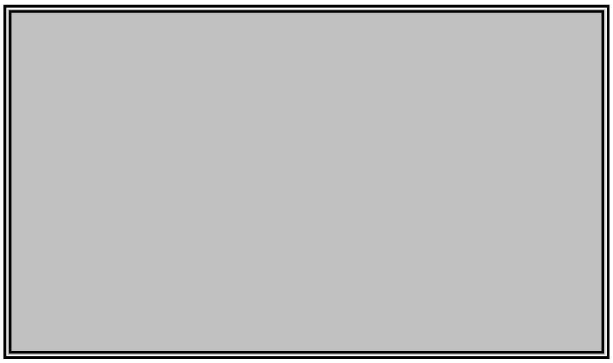

Figure 79
LOAD S28 Records #2

The LCD screen will remain in this state until records are received from the PC link. The user may cancel this operation at any time by pressing the **ESC** (Escape) key.

A 4-digit counter will appear on the screen and increment as each record is received. If more than ten thousand records are processed, the counter will wrap around and start at zero again. This loader performs only minimal validation of received records. It does detect checksum errors in properly formatted “S28” records. If a checksum error is detected, the loader will echo a ‘?’ (Question Mark) to the display.

```
7.                    00017.
```

Figure 80
LOAD S28 Records #3

The load operation terminates when the final record, usually "S804000000FB", is processed, or an '**ESC**' (Escape) character is detected from any enabled input stream. The loader will display a final status message near the bottom of the screen.

```
.                          0666.
Load is completed          OK
```

Figure 81
LOAD S28 Records #4

Any keypress may be used to acknowledge the status message and return control to the LOAD & DUMP MENU screen.

2) Reserved (LOAD & DUMP MENU Option)

This function is not yet assigned. It has been reserved for future use. When this menu items is selected, the following screen will appear:

```
06-18-94          MENSCH COMPUTER          12:34:56

                Function not available  !
```

Figure 82
LOAD & DUMP MENU Option #2

Press any key to return to the LOAD & DUMP MENU screen.

3) DUMP S28 Records (LOAD & DUMP MENU Option)

This LOAD & DUMP MENU item allows the user to send a block of memory to a host computer, via the PC Link serial port, as S28 records.

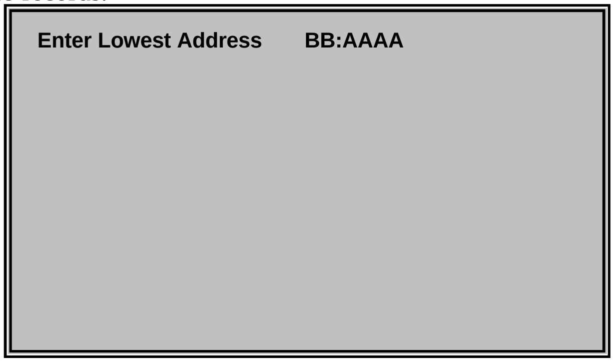

Figure 83
DUMP S28 Records First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

Figure 84
DUMP S28 Records Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will echo the characters as they are sent to the PC Link serial port. The LCD screen may be difficult to read because the data is in S28 record format.

```
S2140000005C5BC5005C69C5005C0002005C0002
0029
S2140000105C0082005C7287005C43EZ005C0002
0019
SZ140000205CZ60400360036000000000000000000
0059
S214000030000000000300030043039503990390
039B
SZ140000405303020300040004660000000028004
2A82
```

Figure 85
DUMP S28 Records - Brief

```
S2140000005C5BC5005C69C5005C0002005C0002
0029
S2140000105C0082005C7287005C43EZ005C0002
0019
SZ140000205CZ60400360036000000000000000000
0059
S214000030000000000300030043039503990390
039B
SZ140000405303020300040004660000000028004
2A82
SZ14000050DF89CF46B7BC002A6C02000500AF00
005F
S2140000606000000000FF0000D30230C81700
0444
S214000070020000F0460000000209442A020044
ZA5ASZ140000000Z0044ZA0200442A0200441A0
```

Figure 86
DUMP S28 Records – Long

Press any key to return to the LOAD & DUMP MENU.

4) DUMP to PRINTER (LOAD & DUMP MENU Option)

This LOAD & DUMP MENU item allows the user to examine a block of memory by dumping its contents, appropriately formatted, to the printer port.

```
Enter Lowest Address      BB:AAAA
```

Figure 87
DUMP to PRINTER First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address      BB:AAAA      00:0000
Enter Highest Address     BB:AAAA
```

Figure 88
DUMP to PRINTER Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will echo the characters as they are sent to the Printer serial port. The LCD screen may be difficult to read because the data has been formatted for the longer lines on the printer. Each line displays the address and sixteen bytes of data in hexadecimal format.

```
Address  0  1  2  3  4  5  6  7  8  9  A
 B  C  D  E  F
00:0000  5C  5B  C5  00  5C  69  C5  00  5C  00  02
  00  5C  00  82  00
00:0010  5C  00  82  00  5C  72  B7  00  5C  43  EZ
  00  5C  00  82  00
00:0020  5C 26  04  00  7D  00  7D  00  00  00  00
  00  00  00  00  00
00:0030  00  00  00  00  03  00  03  00  35  01  BF
  01  8D  02  93  82
00:0040  0E  02  A8  02  00  04  00  04  66  00  00
  00  0Z  80  04  ZA
00:0050  DF  89  CF  Z0  B7  BC  80  ZA  6C  02  00
  05  00  AF  00  00
```

Figure 89
DUMP to PRINTER - Brief

```
Address  0  1  2  3  4  5  6  7  8  9  A
 B  C  D  E  F
00:0000  5C  5B  C5  00  5C  69  C5  00  5C  00  02
  00  5C  00  82  00
00:0010  5C  00  82  00  5C  72  B7  00  5C  43  EZ
  00  5C  00  82  00
00:0020  5C 26  04  00  7D  00  7D  00  00  00  00
  00  00  00  00  00
00:0030  00  00  00  00  03  00  03  00  35  01  BF
  01  8D  02  93  82
00:0040  0E  02  A8  02  00  04  00  04  66  00  00
  00  0Z  80  04  ZA
00:0050  DF  89  CF  Z0  B7  BC  80  ZA  6C  02  00
  05   00  AF  00  00
00:0060  60  00  00  00  00  00  FF  00  00  BF  0
```

Figure 90
DUMP to PRINTER – Long

Press any key to return to the LOAD & DUMP MENU.

5) DUMP to Screen (LOAD & DUMP MENU Option)

This LOAD & DUMP MENU item allows the user to examine a block of memory by dumping its contents, appropriately formatted in hexadecimal, to the LCD screen.

```
Enter Lowest Address      BB:AAAA
```

Figure 91
DUMP to Screen First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address      BB:AAAA      00:0000
Enter Highest Address     BB:AAAA
```

Figure 92
DUMP to Screen Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will display the first ninety-six locations of the memory block. The address and eight locations will be displayed on each line in hexadecimal format.

```
Address   0  1  2  3  4  5  6  7

00:0000   5C D3 CE 00 5C E1 CE 00
00:0008   5C 00 82 00 5C 00 82 00
00:0010   5C 00 82 00 5C AB C2 00
00:0018   5C D1 ED 00 5C 00 82 00
00:0020   5C 21 81 00 5C 00 5C 00
00:0028   00 00 00 00 00 00 00 00
00:0030   00 00 00 00 03 00 03 00
00:0038   04 04 84 04 00 04 00 04
00:0040   03 04 81 01 00 00 00 00
00:0048   0D D9 00 00 48 D9 00 04
00:0050   DF D9 01 20 4B C6 00 04
00:0058   6C 02 00 05 00 A7 0F 00
```

Figure 93
DUMP to Screen Display

Only twelve lines may be displayed at a time. The user may press any key to display successive screens of data, until reaching the end of the selected memory block. After the last location of the memory block has been displayed, any keypress will return to the LOAD & DUMP MENU.

6) ASCII Screen Dump (LOAD & DUMP MENU Option)

This LOAD & DUMP MENU item allows the user to examine a block of memory by dumping its contents, appropriately formatted in ASCII, to the LCD screen.

```
Enter Lowest Address      BB:AAAA
```

Figure 94
ASCII Screen Dump First Prompt

When this option is selected, it will display the above prompt for the *lowest* address in the memory block. The user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

```
Enter Lowest Address       BB:AAAA      00:0000
Enter Highest Address      BB:AAAA
```

Figure 95
ASCII Screen Dump Second Prompt

After the user has entered a *lowest* address, the above prompt for *highest* address will appear. Again, the user may cancel this operation by pressing the **ESC** (Escape) key, instead of entering an address.

The LCD screen will echo the characters as they are sent to the PC Link serial port. The LCD screen may be difficult to read because the data is in S28 format.

```
00:000 \ * * * \ * * * \ * * * \ * * *
00:0010 \ * * * \ * * * \ * * * \ * * *
00:0020 \ . * * * * * * * * * * * * * *
00:0030 * * * * * * * * * * * * * * * *
00:0040 * * * * * * * * * * * * K * * *
00:0050 * * * * K * * * 1 * * * * * * *
00:0060 * * * * * * * * * * * * * * * *
00:0070 * 7 * * * * B * K * * * K * * *
00:0080 K * * * K * * * K * * * K * * *
00:0090 K * * * K * * * K * * * K * * *
00:00A0 K * * * * * * * K * * * K * * *
00:00B0 K * * * K * * * K * * * K * * *
```

Figure 96
ASCII Screen Dump Display

Press any key to return to the SETUP MENU screen.

7) ALTER MEMORY (LOAD & DUMP MENU Option)

This option item allows the user to change the contents of RAM locations. First, a prompt will appear requesting the first address to be modified. Any valid address may be entered, but only the RAM location can be changed.

```
Enter Address        BB:AAAA
```

Figure 97
ALTER MEMORY Prompt

When a valid address response has been entered, then the currents of sixteen locations, beginning at the specified address, will be displayed. The cursor will be positioned below the first location, and the user may begin entering new data into consecutive locations.

```
Address    4    5    6    7    8    9    A    B

00:1234    26   85   20   0C   00   00   01   40
00:123C    41   42   35   28   44   46   28   42
00:1234
```

Figure 98
ALTER MEMORY Edit Screen

The cursor will advance by spacing over to the next field as each value is typed. Backspacing is supported, however only the **ENTER** key will exit and return control to the LOAD & DUMP MENU. The **ESC** (Escape) key may behave unpredictably, and should not be used.

---

8) RETURN TO MAIN MENU (LOAD & DUMP MENU Option)

Pressing **ESC** (Escape) or selecting this option returns control to the MAIN MENU.

7) ROM MONITOR (MAIN MENU Option)

Selecting the ROM MONITOR option allows the PC Link to gain access to the internal debugger of the W65C265 chip. The PC, or other host, must be executing a communications program such as terminal emulation, in order to use the Mensch ROM Monitor. When this menu item is selected, the startup message and prompt will be sent via the PC link.

```
MENSCH ROM Version 2.03
     ©  Copyright 1994
Assembled   Wed   Dec  14   11:09:04   1994

PCntr      Acc        Xreg      Yreg      Stack
00:E344    01 00      E0 CC     00 E1     01 FB

     DirRg   F      DBK
     00 00   22     00

Status Reg
N    V    M    X    D    I    Z    C
0    0    1    0    0    0    1    0
>  (Enter Any Command Here!)
```

Figure 99
ROM Monitor Startup Prompt

The following commands are available from the PC, using the Mensch ROM Monitor.

Command Summary

<table>
<tr><th>Command</th><th>Usage</th></tr>
<tr><td>A</td><td>ALTER registers.</td></tr>
<tr><td>B</td><td>Set BREAKPOINT</td></tr>
<tr><td>D</td><td>DISPLAY memory block in hexadecimal.</td></tr>
<tr><td>F</td><td>FILL memory block with constant.</td></tr>
<tr><td>G</td><td>GO to address (JML execution).</td></tr>
<tr><td>H or ?</td><td>Display HELP Menu</td></tr>
<tr><td>J</td><td>Jump to subroutine using 24-bit address.
(JSL execution)</td></tr>
<tr><td>M</td><td>Examine/Change MEMORY location.</td></tr>
<tr><td>N</td><td>Display/Change current DATE.</td></tr>
<tr><td>R</td><td>Display REGISTERS.</td></tr>
<tr><td>S</td><td>Read ‘S28’ record format.</td></tr>
<tr><td>T</td><td>Examine/Change current TIME</td></tr>
<tr><td>U</td><td>USER command prefix.</td></tr>
<tr><td>W</td><td>WRITE block of memory as ‘S28’ records.</td></tr>
<tr><td>|</td><td>Quick access: Examine/change registers.</td></tr>
<tr><td>/</td><td>Quick access: Examine/change memory.</td></tr>
<tr><td>></td><td>Quick access: Display NEXT memory location.</td></tr>
<tr><td><</td><td>Quick access: Display PREVIOUS location.</td></tr>
<tr><td>(space)</td><td>Quick access: Display current memory location.</td></tr>
<tr><td>^ C</td><td>Cancel current operation.</td></tr>
</table>

Figure 100
ROM Monitor Commands

This is just a summary. Full details of the commands and their proper usage may be found in the **Mensch Monitor ROM REFERENCE MANUAL.**

8) RUN PCMCIA SHELL (MAIN MENU Option)

The PCMCIA Shell is a command interpreter. This MAIN MENU item allows the user to enter and execute “DOS-like” commands which relate to the PCMCIA DOS-compatible file emulation.

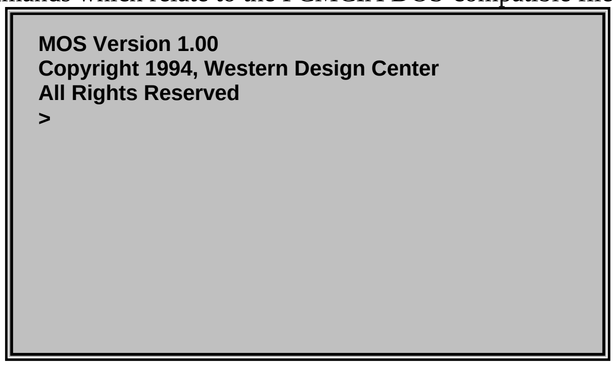

Figure 101
RUN PCMCIA SHELL

When the above prompt (‘>’) appears, the user may enter a command (i.e. FORMAT, DIR, ect.) or type “EXIT” to return to the MAIN MENU.

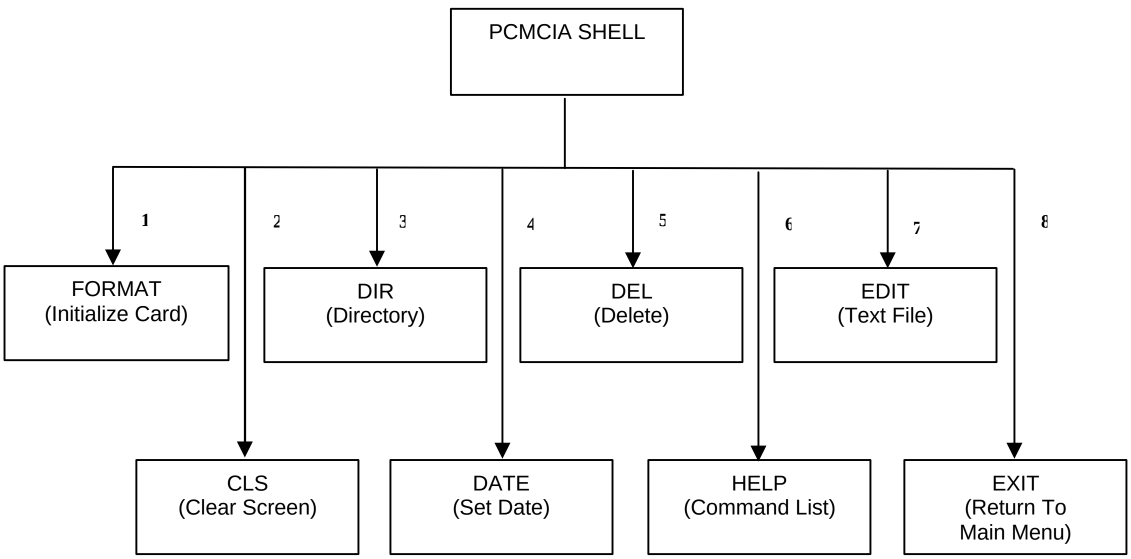

Figure 102
PCMCIA Shell Options

The available commands may change, but typing “HELP” should always display the current list.

Programming The Mensch Computer

The Firmware provides some subroutines to allow *user-provided* software to perform basic I/O functions with the keyboard, display, modem, printer, PC link interface, and controller. The *user-provided* software will access these subroutines through a vector table in EPROM.

These subroutines will pass arguments through registers and use the carry-bit as an error flag. Normally, the carry bit will return clear, indicating normal execution. The carry bit will be set if an error occurred. If further qualification of the error is appropriate, a code will be returned in register-A.

Serial Port Programming Considerations

The W65C265 micro-controller chip contains four serial communication ports. These have been allocated on the Mensch as follows: S0 = Keyboard, S1 = Printer, S2= Modem, and S3 = PC Link. The Menschworks application and firmware library subroutines are available to support these uses. Within limits, the serial port subroutines can be used as generic drivers in other configurations, such as: two printers or two modems instead of one each.

Baud Rate Generation

The most important consideration when using more than two serial ports on the W65C265 micro-controller involves *baud rate selection*. While there are four serial ports, there are only two timers[11] available for baud rate generation. Therefore, when three or four serial ports are used, at least one baud rate generator must be shared.

The Mensch Computer is initially configured to operate the keyboard, printer, and PC link from a single baud rate generator. This is purely a pragmatic decision, which assumes that the user has control of local devices. The modem may communicate with a remote system, and therefore should have an independent baud source.

Programmers may reconfigure which port uses which timer by writing a custom configuration pattern to the *Transmit Control Register (TCR)* of the W65C265 chip. The TCR is mapped to address: $00:DF42. Basically, each bit of the most significant nibble of the register selects either Timer 3 (0) or Timer 4 (1) as the baud source. Likewise, the LSB must be clear if Timer 4 is used.

Developers who want to use the serial port registers directly should consult the **W65C265 INFORMATION SPECIFICATION AND DATA SHEET** for details of operation.

[11] Each internal UART of the W65C265 may select either **Timer 3** or **Timer 4** as a baud rate source. Timer 4 may, in some W65C265 configurations, be used for *Pulse Input/Output* instead of baud rate generation. In that case, Timer 3 must be used by all four serial ports.

Support Subroutines

The Firmware support library provides two subroutines to set baud rates in the standard configuration.

Baud Rate Support Subroutines
_SELECT_MODEM_BAUD_RATE (for modem port only)
SELECT_COMMON_BAUD_RATE (for all ports except modem)

Each subroutine allows the following baud rates: 110,150, 300, 600, 1200, 1800, 2400, 4800, 9600, 14400, 19200, 38400, 57600, and 115000.

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Keyboard I/O Software

Most application software for the Mensch will interact with users via the keyboard and display. Therefore, it is important to understand how the keyboard operates. The keyboard provided with the Mensch Computer will automatically go in low power mode during periods of inactivity. This inactivity can be forced by a command from the Mensch CPU.

Decoding Keyboard Status

Programs may use the *RETRIEVE_KEYBOARD_STATUS* subroutine to determine the current status of the keyboard serial port. This status will reveal whether or not the keyboard has been disabled. If the keyboard is active, status will also show availability and overflow conditions for the transmit and receive buffers.

Decoding Keyboard Input

Application programs may accept keyboard input via the *GET_KEYBOARD_CHARACTER* subroutine. Serial input subroutines in the firmware library use an option parameter. This specifies whether the subroutine should return or wait when no character is available. The Mensch keyboard operates in two modes. The *ASCII Code Mode* transmits single-byte ASCII characters. Under this mode, most keys will generate different codes when pressed with the **CTRL**, **ALT**, or **SHIFT** keys. These modifier keys do not generate a code themselves.

One code is generated for every key in *Keyscan Code Mode*, including each **CTRL**, **SHIFT**, and **ALT** key. This mode may be useful for applications which need to redefine the meaning of keys, or implement custom keyboards.

Complete descriptions of all keyboard codes are provided in **Appendix E – Keycode To ASCII Conversion Table**.

Commanding The Keyboard

The keyboard provided with the Mensch Computer recognizes several commands from the CPU module. Two firmware library subroutines are available to help application programs command the keyboard. The first: *SEND_BYTE_TO_KEYBOARD* is a generic serial output subroutine for the keyboard port. It may be used to send *any* characters to *any* serial device attached to that port.

The Mensch Computer Keyboard has several configurable parameters that control the operation of the keyboard. Among the configurable items are: Change the repeat time and the repeat rate; Set the time before keyboard goes into low-power operation; Read out the current Firmware version; and set the stat of the LED's.

- **Repeat Time:**

The repeat time controls how soon after a single key has been pressed, when the keyboard will begin processing the key as a repeating key. The number sent to the keyboard is in 10's of milliseconds. For example, to set the keyboard to start repeating a key after 500 ms, the data sent to the keyboard would be 50.

- **Repeat Rate:**

The repeat rate controls how soon, after a key has begun repeating, that another key code would be sent to the Mensch. Again, this number is in 10's of milliseconds.

Repeat Time and Repeat Rate are programmed at the same time. The computer must transmit an ASCII '**R**' and then the 2 digits for the repeat time value followed by the 2 digits for the repeat rate value.

- **Keyboard Idle Time:**

This is the number of seconds that the keyboard will wait for another key to be pressed prior to going into low-power mode. When the keyboard is in low power operation, the LED's will be turned off but keyscans will still occur but not at the higher speed during normal operation. When a key is pressed, the keyboard will resume the higher speed and scan the keyboard for the key that woke it up. All LED's will be restored to their respective states prior to shut down.

The keyboard idle time is set by sending an ASCII '**I**' followed by 2 digits for the number of seconds before idling. Because only 2 digits can be transmitted, the idle time may be set from 1 second to 99 seconds.

- **Setting LED States:**

This function can be used to turn an LED on or turn it off. The data sent via this command is bit oriented. The low bit represents the Num Lock LED and the 2^{nd} bit represented the Caps Lock LED. The data is sent by sending an ASCII '**L**' followed by 1 byte that contains the states for the LED's.

For example, if you wanted to turn on the Caps Lock LED and turn off the Num Lock LED you would send: 'L' 0x02. To turn on both LED's you would send 'L' 0x03.

- **Reading Firmware Version:**

The Mensch Computer may also read out the current firmware version from the processor. To do this the computer would send an ASCII '**V**'. The keyboard will then begin transmitting a string of characters followed by the ACK (0x06) character. The string being sent will be variable in length.

Support Subroutine

The Firmware support library provides several subroutines to facilitate keyboard I/O in application programs.

Keyboard Support Subroutines
_CONTROL_KEYBOARD_PORT
_GET_KEYBOARD_CHARACTER
_RETRIEVE_KEYBOARD_STATUS
_SEND_BYTE_TO_KEYBOARD
SELECT_COMMON_BAUD_RATE

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Those library subroutines which support generalized input/output streams may be configured to communicate with the keyboard. Refer to the descriptions of CONTROL_INPUT and CONTROL_OUTPUT for specific details.

Printing From Application Programs

It is often useful for application programs to generate hardcopy output. Programs on the Mensch will use the serial printer port via library subroutines. These routines allow the program check the printer port for availability, poll the serial printer, and send characters to the device to be printed. When not needed, the printer port may be *turned: OFF* to conserve power.

Support Subroutines

The Firmware support library provides several subroutines to facilitate printer usage in application programs.

Printer Support Subroutines	
	_PtLn
_CONTROL_PRINTER_PORT (ON/OFF)	_PtCode
_GET_PRINTER_BYTE (from printer port)	_SetText*
_PRINT_BYTE (Send via printer port)	_SetGraph*
_RETRIEVE_PRINTER_PORT_STATUS	_SetGraphText*
SELECT_COMMON_BAUD_RATE	_PtScreen*

*NOTE: These subroutines are used to support the *print screen* function which automatically copies the LCD memories to the printer port.

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Those library subroutines which support generalized input/output streams may be configured to communicate with the printer port. Refer to the descriptions of the: CONTROL_INPUT and CONTROL_OUTPUT subroutines for specific details.

Modem Communications

Many types of applications require communications with a remote system. This usually is achieved by using modems and a telephone link. The Mensch allows user application programs to do this via the serial modem port. Library subroutines are available so the software can check the modem port for availability, poll and configure Hayes-compatible modems, receive and send characters to another modem. The modem port on the Mensch has an independent baud rate generator which may configure as needed. When not in use, the modem port may be *turned: OFF* to conserve power.

Support Subroutines

The Firmware support library provides several subroutines to facilitate modem communications in application programs.

Modem Support Subroutines	
_CONTROL_MODEM_PORT	GET_MODEM_RESPONSE
_GET_MODEM_BYTE	MODEM_ANSWER
_RETRIEVE_MODEM_PORT_STATUS	MODEM_DIAL
_SELECT_MODEM_BAUD_RATE	MODEM_HANG_UP
_SEND_A_MODEM_BYTE	MODEM_REDIAL
_SEND_MODEM_STRING	

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Those library subroutines which support generalized input/output streams may be configured to communicate with the modem port. Refer to the descriptions of the: CONTROL_INPUT and CONTROL_OUTPUT subroutines for specific details.

PC Link Programming

The PC Link serial port may also be used to communicate with another system via a modem. This port is limited, because it uses the common baud rate generator in the normal Mensch configuration. Application programs may choose to use this port in a variety of ways. Initially it has been allocated for use as a direct connection link to a PC or other desktop/portable computer. This will be helpful to developers who do most of their programming and editing on an external development system.

Support Subroutines

The Firmware support library provides several subroutines to facilitate communication via the PC link serial port in application programs.

PC Link Support Subroutines
_CONTROL_PC_PORT
_RETRIEVE_PC_PORT_STATUS
GET_BYTE_FROM_PC
SELECT_COMMON_BAUD_RATE
SEND_BYTE_TO_PC

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Those library subroutines which support generalized input/output streams may be configured to communicate with the PC link port. Refer to the descriptions of the: CONTROL_INPUT and CONTROL_OUTPUT subroutines for specific details.

Controller Port Usage

Developers may choose the Mensch as a prototyping platform for product applications which do not require a game controller. These configurations may choose to use this port for other purposes. It should be noted that only seven of the eight bits are normally user definable. The MSB will still control the +5 volt supply to the connector. There is a jumper (**JMP4**) which may be used to change this feature and allow the user to define the entire 8-bit port.

Game Programming

The most significant bit of the port (PB7) is used as an output to switch the supply voltage to the controller connector. If the controller has been turned: **OFF**, via the CONTROL_CONTROLLER_PORT subroutine, then the other returned status bits will be misleading.

Switch encoding may be interpreted from the following table:

PB6	PB5	PB4	PB3	PB2	PB1	PB0	Notes
Start	0	A	-	-	Down	Up	
C	1	B	Right	Left	Down	UP	

Support Subroutines

There are only several subroutines which relate directly to using the controller port within application programs:

Controller Support Subroutines

_CONTROL_CONTROLLER_PORT

_GET_CONTROLLER_DATA

_RETRIEVE_CONTROLLER_STATUS

Detailed descriptions of these library subroutines are provided in: **Appendix B – Firmware Subroutine Library**.

Using the LCD Display

The LCD display provided with the normal Mensch configuration supports both character mode and graphics mode. It will display text as sixteen lines of forty characters each. The graphics resolution is 240 horizontal and 128 vertical dots. Other modes are available and application programs may choose to use them.

This LCD module itself contains a Densitron LCD (LM3229A128G240SNG) and a Toshiba controller (T6963C) [12] board. This controller board is capable of driving other LCD displays. Some developers may choose a different one in their configuration.

The firmware library subroutines support the normal Mensch configuration. They may not work in other modes or with other LCD displays.

Accessing The Display

The firmware allows programmers to check and control the configuration and availability of the LCD display from application programs. Specific functions are available to set the display mode to text, graphics, or both. In text mode and graphics mode there are options regarding how output is displayed. When both modes are used, there are options regarding how the images are combined.

Support Subroutines

Library subroutines available to programmers when writing configuring or checking LCD are listed in the following table:

LCD Configuration Support Subroutines
_CONTROL_DISPLAY *(Power)*
RETRIEVE_DISPLAY_STATUS

12 The data sheet on the LM3229A128G240SNG and **Application Notes for the T6963C LCD Graphics Controller** from Densitron provide a detailed description of this display and its operation.

Displaying Text

Basically, text may be written to the display after positioning the cursor at the desired location. As each character is written, the cursor will advance to the next position. When the cursor reaches the last position, at line 16 and column 40, it will not advance. Any additional characters will just overwrite the last location. This is also true if the *right arrow* or *down arrow* character is used to advance the cursor.

In a similar manner, the *backspace*, *left arrow*, and *up arrow* characters cannot "back up" the cursor beyond the upper left corner of the screen, at line 1 and column 1.

Support Subroutines

Library subroutines for programmers to use when writing to the LCD screen in text mode:

LCD Text Support Subroutines
CLEAR_LCD_DISPLAY
POSITION_TEXT_CURSOR (@ Row & Column)
CLEAR_TO_END_OF_LINE (Text line)
WRITE_LCD_CHARACTER (@ Text Cursor Position)
DISP_LCD_STRING (@ Text Cursor Position)
WR_LCD_STRNG
RD_LCD_STRNG
MOVE_PAGE_TO_BUFF
MOVE_BUFFER_TO_LCD
_WrDec

Those library subroutines which support generalized output streams may also be configured to write to the LCD screen. Refer to the description of the: CONTROL_OUTPUT subroutine for specific details.

A special set of library subroutines has been provided to assist developers in producing their own menus on the LCD screen.

Menuing Support Subroutines
_CHECK_YN
_DISP_LCD_HEADER
_DO_MAIN_MENU
_GET_BIN_NUM
_Get_HiLo
_TIME_DATE_CHK
MENU_POINT
MENU_SETUP

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

Displaying Graphics

Programmers intending to use the Mensch LCD screen in graphics mode are advised to study the data sheet on the LM3229A128G240SNG LCD from Densitron, and the associated application notes on the Toshiba controller board.

Support Subroutines

Library subroutines for programmers to use when plotting graphics on the LCD screen:

LCD Graphics Support Subroutines	
CLEAR_LCD_DISPLAY	_Line
POSITION_PIXEL (@ Coordinates: H, V)	_HLine
WRITE_PIXEL (@ Graphics Cursor Position)	_VLine
_SetColor	_SetFill
_ClearColor	_ClearFill
_Point	_Box
	_Circle

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

LCD Screen Printing

The contents of the Mensch LCD screen may be copied to the printer using firmware functions. Options allow programmers to select only text memory, only graphics memory, or both when dumping the LCD screen to the printer port.

Support Subroutines

The specific library subroutines for programmers to use when dumping the LCD screen are listed in the following table:

LCD Print Screen Support Subroutines
_SetText
_SetGraph
_SetGraphText
_PtScreen

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

Sound Generation

The Mensch is built around the W65C265 which has the essential circuitry to perform PWM I/O. This offers an inexpensive, simple, “low-tech” approach to customized digital audio in consumer products. Two of the timers are configurable as digital sine-wave tone generators. These are suitable for DTMF or modem usage. The W65C265 has already been proven in “Caller ID” products. This system is an ideal development platform for ADSI[13] telephones.

Details on using the tone generators directly is covered in the **W65C265S INFORMATION, SPECIFICATION, AND DATA SHEET** form WDC. Example configuration values for the Mensch wherein the main frequency is 3,6864 MHz are shown in the following table:

DTMF Tone (Hz)	Register Value	Error (%)
697	$014A	0.133
770	$012A	0.073
852	$010D	0.156
941	$00F4	0.062
1209	$00BE	0.225
1336	$00AB	0.263
1477	$009B	0.005
1633	$008C	0.063

Support Subroutines

Library subroutines for programmers to use when attempting to generate sound with the Mensch Computer:

Audio Support Subroutines
_SEND_BEEP
_SEND_DTMF_DIGIT
CONTROL_SPEAKER_AMP
CONTROL_TONES

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

[13] The *Analog Display Services Interface (ADSI)* has been defined by Bell Communications Research to enable a visual context-sensitive user interface to telephone network services.

DOS-Compatible File Support

The Mensch has two slots for IC memory cards. They each appear to be memory blocks to regular programs. The firmware library provides subroutines which can read and manipulate DOS-compatible "files" on these IC memory cards. This allows the exchange of data with DOS-compatible laptop computers on a common physical media.

The DOS-compatible *File Allocation Table* or FAT always resides at the low end of any formatted IC card. The LOW IC card of the Mensch is physically mapped to begin at Bank #1. Most IC cards only decode as many address lines as the actually need. This allows the Mensch to view Bank #0 of an IC card as the *last* bank on the card. Therefore, an IC card with a proper DOS-compatible file structure, inserted into the LOW slot, may be decoded according to the following table:

Card Size	Card Mapped Banks Range	FAT Mapped To Bank #
128K	1-2	2
256K	1-4	4
512K	1-8	8
1M	1-$10	$10
2M	1-$20	$20
4M or Larger	Not usable for DOS-compatible files.	Lowest bank and above $3F lost.

Support Subroutines

The following library subroutines are available to application programs when accessing the IC memory cards as DOS-compatible file devices:

FCLOSE	FGETW	FPUTBLOCK	GETDFREE
FDELETE	FILELENGTH	FPUTC	IS_CARD_INSERTED
FGETBLOCK	FINDFIRST	FPUTS	LOG_DRIVE
FGETC	FOPEN	FPUTW	OS_SHELL
FGETS	FORMAT	FSEEK	SELECT_DISK

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

System Functions

Many elements of the Mensch are not typically within the domain of any particular kind of application program. The time-of-day clock/calendar, selectable alarm function, and power management are examples of such elements.

Time-Of-Day Clock/Calendar

The time-of-day clock feature of the Mensch Computer operates even when the system is in low-power mode. This is because it resides in the internal RAM and ROM of the W65C265S itself. The time-of-day clock may also be set and read by application programs.

Support Subroutines

Library subroutines for programmers to use when accessing the time-of-day clock/calendar:

Clock/Calendar Support Subroutines
READ_DATE
SET_DATE
READ_TIME
SET_TIME

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

Programmable Alarm

A built-in alarm function uses the time-of-day clock. The user may set or check the alarm from the keyboard via utility programs, or within software applications via library subroutines.

Support Subroutines

The library subroutines which support application programs in accessing the programmable alarm include:

Alarm Support Subroutines
GET_ALARM_STATUS
READ_ALARM
RESET_ALARM
SET_ALARM

This basic set may be used as building block functions. Programmers may use them to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information on using these subroutines.

Interval Tracking

Two kinds of interval tracking support are available in the firmware subroutine library. The first uses software timers to monitor *timeout events*, while the other uses the STOPWATCH function to measure *elapsed time* intervals.

Counter Usage

There are five software timers, 16-bit variables, which may be initialized via the *_SET_COUNT* subroutine. The value in each timer will be decremented at regular (1/100th Sec.) intervals, until it reaches zero. These software timers are just 16-bit *down-counters* which may be read via the *_RD_COUNT* subroutine.

Stopwatch Usage

The STOPWATCH function may be used as an event timer. The STOPWATCH is a 32-bit counter which counts up at regular (1/100th Sec.) intervals.

The STOPWATCH may be reset to zero via the _CLR_STPWTCH subroutine.

The current value of the STOPWATCH may be read via the _RD_STPWTCH subroutine.

Power Management

A *voltage detection circuit* in the Mensch hardware may be monitored by software. Application software may check to determine whether the system is operating off power from batteries or the external charger/power module. If the system is using batteries alone, the condition of the batteries may be monitored to avoid problems. The Firmware provides a library subroutine for programmers to use when checking the voltage detection status: *CHECK_VOLTAGE*.

If the battery condition is marginal, the *ENGAGE_LOW_POWER_MODE* subroutine allows the user software to place the system in low-power mode. Applications may also use some of the features and elements of the Mensch which permit programs to selectively manipulate power switching controls over subsystems.

Support Subroutines

Several Firmware library subroutines have been provided to facilitate power management within the system:

Power Management Support Subroutine
_CHECK_VOLTAGE
_CONTROL_CONTROLLER_PORT
_CONTROL_DISPLAY
_CONTROL_KEYBOARD_PORT
_CONTROL_MODEM_PORT
_CONTROL_PC_PORT
_CONTROL_PRINTER_PORT
_ENGAGE_LOW_POWER_MODE
CONTROL_SPEAKER_AMP

Programmers may use these and other library subroutines to develop more sophisticated libraries of their own. Refer to: **Appendix B – Firmware Subroutine Library** for more information.

Mensch FORTH Support

Charles Moore, of the National Radio Astronomy Observatory, developed the Forth Programming language in the 1960's. He intended to use it specifically in instrumentation and control applications related to radio astronomy. Forth became a popular development tool during the 1970's among a broad base of hackers and hobbyists.

The Forth Interest Group was formed in 1978 to encourage the interchange of ideas and experiences with Forth. The first fig- Forth standard was defined in 1979 and later updated in 1983. Implementations of Forth have been made available for various platforms. Many Forth compilers have been released into the public domain.

The popularity of Forth arises from the ease of use. Programs can be easily written and debugged in Forth, even by beginners. It is a very transportable language, so programs written on one system can be executed on another.

The Western Design Center expects to see a 16-bit implementation of Forth executing on the Mensch, within 1Q95. It is an adaptation of an earlier version of Forth for the W65C802. This involved the Apple IIgs configuration which differs from the Mensch. Mensch Forth will be available from a third-party developer and through WDC.

Appendices

Appendix A – Replacing The Battery Pack

The battery pack may be replaced by first removing the screws from the front panel of the CPU module.

Figure 103
Mensch Computer
Front Panel

Next, remove the plate and frame allowing the main board to slide out of the encasement.

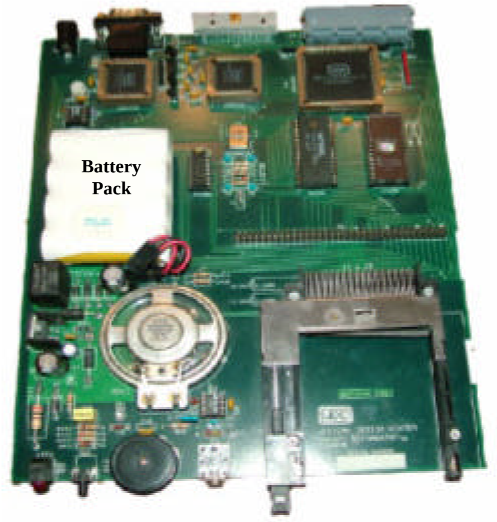

Figure 104
Replacing The Battery Pack

Unplug the existing battery pack and remove it from the board. The battery connector is keyed and cannot be incorrectly installed. Using the guides, slide the main board back into the encasement. Replace the front panel, frame, and screws.

Appendix B – Firmware Subroutine Library

The ROM Monitor and Mensch Operating System provide some subroutine vectors for basic I/O and various useful functions in the Mensch. Some of these subroutines allow *user-provided* software to access the keyboard, display, modem, printer, PC link interface and controller. Others relate to power management, audio output, timing, or data conversion.

Library Vector Table

The vector tables for these subroutines begin at specific locations in the W65C265S ROM ($00:E000), and the EPROM of the Mensch ($00:8010). The bases of these tables and the respective offsets of defined vectors will not move as new vectors are defined.

Power Management Support

_CHECK_VOLTAGE	$00:806B
_ENGAGE_LOW_POWER_MODE	$00:806E
(_CONTROL_CONTROLLER_PORT	*$00:8065)*
(_CONTROL_DISPLAY	*$00:805C)*
(_CONTROL_KEYBOARD_PORT	*$00:8059)*
(_CONTROL_PRINTER_PORT	*$00:805F)*
(_CONTROL_MODEM_PORT	*$00:8062)*
(_CONTROL_PC_PORT	*$00:811F)*
(CONTROL_SPEAKER_AMP	*$00:8104)*

Development Interface Support

XS28IN	$00:E081
DumpS28	$00:E012

Clock/Calendar/Alarm Support

GET_ALARM_STATUS	$00:E030
READ_ALARM	$00:E051
READ_DATE	$00:E054
READ_TIME	$00:E057
RESET_ALARM	$00:E05A
SET_ALARM	$00:E06F
SET_DATE	$00:E075
SET_TIME	$00:E078

Audio Output Support

_SEND_BEEP	$00:8074
_SEND_DTMF_DIGIT	$00:8071
CONTROL_TONES	$00:E009
(CONTROL_SPEAKER_AMP	*$00:8104)*

Game Controller Support

_GET_CONTROLLER_DATA	$00:8068
_RETRIEVE_CONTROLLER_STATUS	$00:8056
(_CONTROL_CONTROLLER_PORT	*$00:8065)*

Serial I/O – Baud Rate Generators

(_SELECT_MODEM_BAUD_RATE	*$00:8047)*
(SELECT_COMMON_BAUD_RATE	*$00:E060)*

Serial I/O – Keyboard

_GET_KEYBOARD_CHARACTER	$00:8023
_RETRIEVE_KEYBOARD_STATUS	$00:8020
_SEND_BYTE_TO_KEYBOARD	$00:8026
(_CONTROL_KEYBOARD_PORT	*$00:8059)*
(SELECT_COMMON_BAUD_RATE	*$00:E060)*

Serial I/O – Printer

_GET_A_PRINTER_BYTE	$00:8041
_PRINT_A_BYTE	$00:803E
_PtLn	$00:81A7
_PtCode	$00:81A4
_RETRIEVE_PRINTER_PORT_STATUS	$00:803B
(_CONTROL_PRINTER_PORT	*$00:805F)*
(SELECT_COMMON_BAUD_RATE	*$00:E060)*

Serial I/O – Modem

_GET_MODEM_BYTE	$00:8050
_RETRIEVE_MODEM_PORT_STATUS	$00:8044
_SEND_A_MODEM_BYTE	$00:804A
_SEND_MODEM_STRING	$00:804D
GET_MODEM_RESPONSE	$00:8116
MODEM_ANSWER	$00:8119
MODEM_DIAL	$00:80F8
MODEM_HANG_UP	$00:80FB
MODEM_REDIAL	$00:811C
(_CONTROL_MODEM_PORT	*$00:8062)*
(_SELECT_MODEM_BAUD_RATE	*$00:8047)*

Serial I/O – PC Link

_RETRIEVE_PC_PORT_STATUS	$00:8053
GET_BYTE_FROM_PC	$00:8033
SEND_BYTE_TO_PC	$00:8063
(_CONTROL_PC_PORT	*$00:811F)*
(SELECT_COMMON_BAUD_RATE	*$00:E060)*

General I/O Stream Support

_CONTROL_INPUT	$00:80DD
_CONTROL_OUTPUT	$00:80E0
BACKSPACE	$00:E003
GET_3BYTE_ADDR	$00:E02D
GET_CHR	$00:E036
GET_PUT_CHR	$00:E03C
GET_STR	$00:E03F
PUT_CHR	$00:E04B
PUT_STR	$00:E04E
SEND_CR	$00:E066
SEND_HEX_OUT	$00:E06C
SEND_SPACE	$00:E069
WR_3_ADDRESS	$00:E07E

Print Screen Support

_PtScreen	$00:81AA
_SetText	$00:81B3
_SetGraph	$00:81B0
_SetGraphText	$00:81AD

Liquid Crystal Display Support – General

CLEAR_LCD_DISPLAY	$00:802C
RETRIEVE_DISPLAY_STATUS	$00:8029
(_CONTROL_DISPLAY	*$00:805C)*

LCD Support – Text Mode

_WrDec	$00:8180
CLEAR_TO_END_OF_LINE	$00:8032
DISP_LCD_STRING	$00:8038
MOVE_PAGE_TO_BUFF	$00:80FE
MOVE_BUFFER_TO_LCD	$00:8101
POSITION_TEXT_CURSOR	$00:802F
RD_LCD_STRNG	$00:80DA
WR_LCD_STRNG	$00:80D7
WRITE_LCD_CHARACTER	$00:8035

LCD Support – Graphics Mode

_Box	$00:8198
_Circle	$00:8195
_ClearColor	$00:8189
_ClearFill	$00:818F
_GetGrStatus	$00:81BF
_GetPoint	$00:81BC
_HLine	$00:819E
_Line	$00:8192
_Point	$00:81A1
_SetColor	$00:8186
_SetFill	$00:818C
_VLine	$00:819B

Menu Support

_CHECK_YN	$00:80F2
_DISP_LCD_HEADER	$00:80EF
_DO_MAIN_MENU	$00:8107
_TIME_DATE_CHK	$00:80EC
Get_HiLo	$00:80F5
MENU_SETUP	$00:80E6
MENU_POINT	$00:80E9

IC Card Support – General

IS_CARD_INSERTED	$00:808C

IC Card Support – PCMCIA Disk Emulation

_OS_SHELL	$00:8077
DIR_COMMND	$00:804D
FCLOSE	$00:8092
FDELETE	$00:80C2
FGETBLOCK	$00:80B3
FGETC	$00:80AA
FGETS	$00:80B0
FGETW	$00:80AD
FILELENGTH	$00:80BC
FINDFIRST	$00:80B9
FOPEN	$00:808F
FORMAT	$00:807A
FPUTBLOCK	$00:80A7
FPUTC	$00:809E
FPUTS	$00:80A4
FPUTW	$00:80A1
FSEEK	$00:8098
LOG_DRIVE	$00:807D
SELECT_DISK	$00:80D1
FNSPLIT	$00:80B6
STRCMP	$00:80C5
DISPLAY_PCMCIA_ERROR	$00:8080

Timing and Counting

_CLR_STPWTCH	$00:8128
_RD_COUNT	$00:8125
_RD_STPWTCH	$00:812B
_SET_COUNT	$00:8122

Miscellaneous

_Bin2BCD	$00:8183
_GET_BIN_NUM	$00:80E3
_INIT_DP_POINTER	$00:8110
_RESTORE_DP_POINTER	$00:8113
_START	$00:810A

ROM Monitor Subroutines

Alter_Memory	$00:E000
ASCBIN	$00:E087
BIN2DEC	$00:E08B
BINASC	$00:E08F
DO_LOW_POWER_PGM	$00:E00C
DUMPREGS	$00:E00F
Dump_1_line_to_Output	$00:E015
Dump_1_line_to_Screen	$00:E018
Dump_to_Output	$00:E01B
Dump_to_Printer	$00:E01E
Dump_to_Screen	$00:E021
Dump_to_Screen_ASCII	$00:E024
Dump_It	$00:E027
FILL_Memory	$00:E02A
GET_HEX	$00:E039
Get_Address	$00:E042
Get_E_Address	$00:E045
Get_S_Address	$00:E048
HEXIN	$00:E093
IFASC	$00:E097
ISDECIMAL	$00:E09B
ISHEX	$00:E09F
RESET	$00:E084
SBREAK	$00:E05D
SET_Breakpoint	$00:E072
UPPER_CASE	$00:E0A3
VERSION	$00:E07B

Library Subroutine Descriptions

The Firmware provides some subroutines to allow *user-provided* software to perform basic I/O functions with the keyboard, display, modem, printer, PC link interface, and controller. The *user-provided* software will access these subroutines through a vector table in EPROM[14].

These subroutines will generally pass arguments through registers and use the carry-bit as an error/exception flag. Normally, the carry-bit will return clear, indicating normal execution. The carry-bit will be set if an error occurred or a notable condition was detected. If further qualification of the error is appropriate, a code will be returned in register-A. Refer to the individual description of each subroutine for specific details.

_Bin2BCD

DESCRIPTION:
This subroutine will convert a 16-bit unsigned value into an equivalent BCD number.

VECTOR:
$00:8183 _Bin2BCD

EXPECTS:
The value to be converted must appear in 16-bit register-X.

RETURNS:
Least significant digits of the BCD number are returned in 16-bit register-X and the most significant BCD digits in 8-bit register-A.

Example: If Value = 65535 Then Register-A = $06
Register-X = $5535

ERRORS:
No meaningful errors are detected.

---

[14] The Western Design Center, Inc. recommends that software developers use an *include file* in their source programs, which assigns symbolic names to these vectors. A current file is available from WDC.

_Box

DESCRIPTION:
This subroutine will plot an orthogonal box on the display in graphics mode.

VECTOR:
$00:8198 _Box

EXPECTS:
The first corner coordinates of the box must be in the most significant bytes of 16-bit register-X and 16-bit register-Y. The coordinates of the diagonally opposite corner must be in the least significant bytes of the same registers.

RETURNS:
Normal operation returns with the carry-bit = *clear*.

ERRORS:
If either coordinate in register-X exceeds 239 or either coordinate in register-Y exceeds 127, then this subroutine will return with the carry-bit = *set* and the box will not be drawn.

_CHECK_VOLTAGE

DESCRIPTION:
This subroutine will sample the voltage sensor status.

VECTOR:
$00:806B _CHECK_VOLTAGE

EXPECTS:
No input arguments.

RETURNS:
Voltage sensor status in 8-bit register-A:

$00	=	External power is available.
$01	=	Running on batteries with adequate power available.
$FF	=	Running on batteries and the batteries are low.

ERRORS:
No meaningful errors are detected. The carry-bit normally will return *clear*, but will be *set* if register-A is non-zero.

_CHECK_YN

DESCRIPTION:
This subroutines will check for "Yes" or "No" responses from keyboard input.

VECTOR:
$00:80F2 _CHECK_YN

EXPECTS:
No input arguments.

RETURNS:
Carry-bit *set* means that an **ESC** (Escape) character was detected.

Carry-bit *clear* means an appropriate "Yes or No" response character has been detected. The code for this will be in the 8-bit register-A, wherein:

0 = 'N' or 'n' for "No"
or
1 = 'Y' or 'y' for "Yes"

ERRORS:
No meaningful errors are detected.

_Circle

DESCRIPTION:
This subroutine will plot a circle on the display in graphics mode.

VECTOR:
$00:8195 _Circle

EXPECTS:
Center of circle coordinates in 16-bit register-X and 16-bit register-Y.

Radius of circle in 8-bit register-A.

RETURNS:
Normal operation returns with the carry-bit = *clear*.

ERRORS:
If the coordinate in register-X exceeds 239 or the coordinate in register-Y exceeds 127, then this subroutine will return with the carry-bit = *set* and the circle will not be drawn.

NOTE:
If any portion of the circle has coordinates outside of the boundaries of the display, then that part will not be drawn.

_ClearColor

DESCRIPTION:

This subroutine will clear the COLOR flag used by other graphics plotting subroutines. The flag is used to determine whether pixel points should be plotted in white or black.

(Clear = White / Set = Black)

VECTOR:

$00:8189 _ClearColor

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors are detected.

_ClearFill

DESCRIPTION:

This subroutine will clear the FILL flag used by other graphics plotting subroutines. The flag is used to determine whether or not the plotting subroutines should fill the shapes when they draw them.

(Set = Fill / Clear = No Fill)

VECTOR:

$00:818F _ClearFill

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors are detected.

_CLR_STPWTCH

DESCRIPTION:

The STOPWATCH function may be used as an event timer. This subroutine will reset the STOPWATCH to zero. The STOPWATCH is a 32-bit counter which counts up at regular (1/100th Sec.) intervals. The current value of the STOPWATCH may be read via the _RD_STPWTCH subroutine.

VECTOR:

$00:8128 _CLR_STPWTCH

EXPECTS:

No input arguments.

RETURNS:

No arguments returned. All received registers are saved and restored before returning.

ERRORS:

No meaningful errors are detected.

_CONTROL_CONTROLLER_PORT

DESCRIPTION:

This subroutine allows the programmer to enable or disable the supply voltage delivered to the controller port connector.

VECTOR:

$00:8065 _CONTROL_CONTROLLER_PORT

EXPECTS:

Control code in 8-bit register-A:

Zero = Supply OFF
Any
Non-Zero = Supply ON

RETURNS:

No arguments returned, no registers changed.

ERRORS:

No meaningful errors.

CAUTION:

It is not meaningful to call this subroutine if the circuit board in the CPU module has been configured to use the controller port as an ordinary 8-bit I/O port. (Refer to the appropriate schematic diagrams for more information.)

_CONTROL_DISPLAY

DESCRIPTION:

This subroutine allows the programmer to reset or manipulate the LCD display characteristics. It can enable or disable the text display, or the graphics display. If both are enabled, this routine can define how they interact.

VECTOR:

$00:805C _CONTROL_DISPLAY

EXPECTS:

Control code in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
0	0	0	0	0	0	0	0	Display is turned: OFF.
-	-	-	-	-	-	-	0	Cursor will not blink, if enabled.
-	-	-	-	-	-	-	1	Cursor will blink if enabled.
-	-	-	-	-	-	0	-	Text cursor: OFF
-	-	-	-	-	-	1	-	Text cursor: ON
-	-	-	-	-	0	-	-	Text display: OFF
-	-	-	-	-	1	-	-	Text display: ON
-	-	-	-	0	-	-	-	Graphics display: OFF
-	-	-	-	1	-	-	-	Graphics display: ON
-	-	0	0	-	-	-	-	OR graphic and text displays.
-	-	0	1	-	-	-	-	EXOR (exclusive OR) graphic and text displays.
-	-	1	0	-	-	-	-	Undefined option. (Sent to the T6963C LCD graphics controller.)
-	-	1	1	-	-	-	-	AND graphic and text displays.
-	0	-	-	-	-	-	-	Select 6x8 graphic mode.
-	1	-	-	-	-	-	-	Select 8x8 graphic mode.
1	-	-	-	-	-	-	-	Reset using pattern.
1	-	0	0	0	0	0	0	Reset using *default* setup parameters. Display = ON, Blinking cursor, and Text OR'ed w/6x8 Graphics.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

_CONTROL_INPUT

DESCRIPTION:

This subroutine sets up the input paths for *GET_CHR* and other stream oriented subroutines. It turns input stream ports ON & OFF.

VECTOR:

$00:80DD _CONTROL_INPUT

EXPECTS:

Control information in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	Do not affect keyboard input.
-	-	-	-	-	-	0	-	Do not affect printer port input.
-	-	-	-	-	0	-	-	Do not affect modem port input.
-	-	-	-	0	-	-	-	Do not affect PC port input.
0	-	-	-	-	-	-	1	Disable keyboard input.
0	-	-	-	-	-	1	-	Disable printer port input.
0	-	-	-	-	1	-	-	Disable modem port input.
0	-	-	-	1	-	-	-	Disable PC port input.
1	-	-	-	-	-	-	1	Enable keyboard input.
1	-	-	-	-	-	1	-	Enable printer port input.
1	-	-	-	-	1	-	-	Enable modem port input.
1	-	-	-	1	-	-	-	Enable PC port input.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

_CONTROL_KEYBOARD_PORT

DESCRIPTION:
This subroutine allows the programmer to clear the I/O buffers and configure the characteristics of the port. It does not select the word length, parity, or number of stop bits. These default to 8-bits, no parity, and one stop bit.

VECTOR:
$00:8059 _CONTROL_KEYBOARD_PORT

EXPECTS:
Keyboard port configuration code in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	No effect on port operation.
-	-	-	-	-	-	0	-	No effect on port operation.
-	-	-	-	-	0	-	-	No effect on port operation.
-	-	-	-	0	-	-	-	No effect on port operation.
-	-	-	0	-	-	-	-	No effect on port operation.
-	-	0	-	-	-	-	-	No effect on port operation.
-	0	-	-	-	-	-	-	No effect on port operation.
0	-	-	-	-	-	-	1	No effect on port operation.
0	-	-	-	-	-	1	-	Disable transmit interrupts.
0	-	-	-	-	1	-	-	Disable XON/XOFF handshaking.
0	-	-	-	1	-	-	-	Send “start” message to keyboard.
0	-	-	1	-	-	-	-	Set DTR signal: FALSE (DTR0 = 1)
0	-	1	-	-	-	-	-	Disable Echo mode.
0	1	-	-	-	-	-	-	Disable UART receive.
1	-	-	-	-	-	-	1	Clear port input buffer.
1	-	-	-	-	-	1	-	Clear port output buffer.
1	-	-	-	-	1	-	-	Enable XON/XOFF handshaking.
1	-	-	-	1	-	-	-	No effect on port operations.
1	-	-	1	-	-	-	-	Set DTR signal: TRUE (DTR0 = 0)
1	-	1	-	-	-	-	-	Enable Echo mode.
1	1	-	-	-	-	-	-	Enable UART receive.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors.

CAUTION:
The keyboard in the Mensch Computer configuration is driven by the common baud rate generator (T4). The *SELECT_COMMON_BAUD_RATE* routine may be used to change the baud rate for the printer and PC link ports, but it will also affect the keyboard port as well.

_CONTROL_MODEM_PORT

DESCRIPTION:

This subroutine allows the programmer to manipulate the supply voltage delivered to the modem serial port connector, clear the I/O buffers and configure the characteristics if the port. It does not select the word length, parity, or number of stop bits. These default to 8-bits, no parity, and one stop bit. The modem port in the Mensch Computer configuration is driven by an independent baud rate generator (T3). The *SELECT_MODEM_BAUD_RATE* routine may be used to change the baud rate for the modem port, without affecting other serial ports.

VECTOR:

$00:8062 _CONTROL_MODEM_PORT

EXPECTS:

Modem port configuration code in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	No effect on port operation.
-	-	-	-	-	-	0	-	No effect on port operation.
-	-	-	-	-	0	-	-	No effect on port operation.
-	-	-	-	0	-	-	-	No effect on port operation.
-	-	-	0	-	-	-	-	No effect on port operation.
-	-	0	-	-	-	-	-	No effect on port operation.
-	0	-	-	-	-	-	-	No effect on port operation.
0	-	-	-	-	-	-	1	No effect on port operation.
0	-	-	-	-	-	1	-	Disable transmit interrupts.
0	-	-	-	-	1	-	-	Disable XON/XOFF handshaking.
0	-	-	-	1	-	-	-	Disable port power.
0	-	-	1	-	-	-	-	Set DTR signal: FALSE (DTR2 = 1)
0	-	1	-	-	-	-	-	Disable Echo mode.
0	1	-	-	-	-	-	-	Disable UART receive.
1	-	-	-	-	-	-	1	Clear port input buffer.
1	-	-	-	-	-	1	-	Clear port output buffer.
1	-	-	-	-	1	-	-	Enable XON/XOFF handshaking.
1	-	-	-	1	-	-	-	Enable port power.
1	-	-	1	-	-	-	-	Set DTR signal: TRUE (DTR2 = 0)
1	-	1	-	-	-	-	-	Enable Echo mode.
1	1	-	-	-	-	-	-	Enable UART receive.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

NOTE:

Any serial port on the W65C265S micro-controller may be driven by either of the baud rate generators. The decision was made to allocate one baud rate source to the modem port and share the other among the remaining ports in the Mensch Computer configuration. This was based significantly upon the assumption that the keyboard, printer and PC link ports could be preset and would probably remain constant. Likewise, it was assumed that the modem port would frequently need to change its baud rate to accommodate external connections such as: BBS systems and on-line services.

_CONTROL_OUTPUT

DESCRIPTION:

This subroutine sets up the output paths for *PUT_CHR* and other stream oriented subroutines. It turns output stream ports ON & OFF.

VECTOR:

$00:80E0 _CONTROL_OUTPUT

EXPECTS:

Control information in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	Do not affect display output.
-	-	-	-	-	-	0	-	Do not affect printer port output.
-	-	-	-	-	0	-	-	Do not affect modem port output.
-	-	-	-	0	-	-	-	Do not affect PC port output.
0	-	-	-	-	-	-	1	Disable display output.
0	-	-	-	-	-	1	-	Disable printer port output.
0	-	-	-	-	1	-	-	Disable modem port output.
0	-	-	-	1	-	-	-	Disable PC port output.
1	-	-	-	-	-	-	1	Enable display output.
1	-	-	-	-	-	1	-	Enable printer port output.
1	-	-	-	-	1	-	-	Enable modem port output.
1	-	-	-	1	-	-	-	Enable PC port output.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

_CONTROL_PC_PORT

DESCRIPTION:

This subroutine allows the programmer to manipulate the supply voltage delivered to the PC link serial port connector, clear the I/O buffers and configure the characteristics if the port. It does not select the word length, parity, or number of stop bits. These default to 8-bits, no parity, and one stop bit.

VECTOR:

$00:811F _CONTROL_PC_PORT

EXPECTS:

PC port configuration code in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	No effect on port operation.
-	-	-	-	-	-	0	-	No effect on port operation.
-	-	-	-	-	0	-	-	No effect on port operation.
-	-	-	-	0	-	-	-	No effect on port operation.
-	-	-	0	-	-	-	-	No effect on port operation.
-	-	0	-	-	-	-	-	No effect on port operation.
-	0	-	-	-	-	-	-	No effect on port operation.
0	-	-	-	-	-	-	1	No effect on port operation.
0	-	-	-	-	-	1	-	Disable transmit interrupts.
0	-	-	-	-	1	-	-	Disable XON/XOFF handshaking.
0	-	-	-	1	-	-	-	Disable port power.
0	-	-	1	-	-	-	-	Set DTR signal: FALSE (DTR3 = 1)
0	-	1	-	-	-	-	-	Disable Echo mode.
0	1	-	-	-	-	-	-	Disable UART receive.
1	-	-	-	-	-	-	1	Clear port input buffer.
1	-	-	-	-	-	1	-	Clear port output buffer.
1	-	-	-	-	1	-	-	Enable XON/XOFF handshaking.
1	-	-	-	1	-	-	-	Enable port power.
1	-	-	1	-	-	-	-	Set DTR signal: TRUE (DTR3 = 0)
1	-	1	-	-	-	-	-	Enable Echo mode.
1	1	-	-	-	-	-	-	Enable UART receive.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

(MORE)

_CONTROL_PC_PORT (Continued)

CAUTION:

The PC link port in the Mensch configuration is driven by the common baud rate generator (T4). The *SELECT_COMMON_BAUD_RATE* routine may be used to change the baud rate for the PC link port, but it will also affect the printer and the keyboard ports as well.

NOTE:

Any serial port on the W65C265S micro-controller, including the PC link port, may be driven by either of the baud rate generators. The decision was made to allocate one baud rate source to the modem port and share the other among the remaining ports in the Mensch configuration. This was based significantly upon the assumption that the keyboard, printer, and PC link ports could be preset and would probably remain constant. Likewise, it was assumed that the modem port would frequently need to change its baud rate to accommodate external connections such as: BBS systems and on-line services. The PC link port may be configured to share the same baud rate source as the modem port, but care should be taken to avoid conflicts.

_CONTROL_PRINTER_PORT

DESCRIPTION:

This subroutine allows the programmer to manipulate the supply voltage delivered to the printer serial port connector, clear the I/O buffers and configure the characteristics if the port. It does not select the word length, parity, or number of stop bit. These default to 8-bits, no parity, and one stop bit. The printer port in the Mensch configuration is driven by the common baud rate generator (T4).

VECTOR:

$00:805F _CONTROL_PRINTER_PORT

EXPECTS:

Printer port configuration code in 8-bit register-A.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0	Meaning
-	-	-	-	-	-	-	0	No effect on port operation.
-	-	-	-	-	-	0	-	No effect on port operation.
-	-	-	-	-	0	-	-	No effect on port operation.
-	-	-	-	0	-	-	-	No effect on port operation.
-	-	-	0	-	-	-	-	No effect on port operation.
-	-	0	-	-	-	-	-	No effect on port operation.
-	0	-	-	-	-	-	-	No effect on port operation.
0	-	-	-	-	-	-	1	No effect on port operation.
0	-	-	-	-	-	1	-	Disable transmit interrupts.
0	-	-	-	-	1	-	-	Disable XON/XOFF handshaking.
0	-	-	-	1	-	-	-	Disable port power.
0	-	-	1	-	-	-	-	Set DTR signal: FALSE (DTR1 = 1)
0	-	1	-	-	-	-	-	Disable Echo mode.
0	1	-	-	-	-	-	-	Disable UART receive.
1	-	-	-	-	-	-	1	Clear port input buffer.
1	-	-	-	-	-	1	-	Clear port output buffer.
1	-	-	-	-	1	-	-	Enable XON/XOFF handshaking.
1	-	-	-	1	-	-	-	Enable port power.
1	-	-	1	-	-	-	-	Set DTR signal: TRUE (DTR1 = 0)
1	-	1	-	-	-	-	-	Enable Echo mode.
1	1	-	-	-	-	-	-	Enable UART receive.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

(MORE)

_CONTROL_PRINTER_PORT (Continued)

CAUTION:

The printer port in the Mensch configuration is driven by the common baud rate generator (T4). The *SELECT_COMMON_BAUD_RATE* routine may be used to change the baud rate for the printer port, but it will also affect the PC link and the keyboard ports as well.

NOTE:

Any serial port on the W65C265S micro-controller, including the printer port, may be driven by either of the baud rate generators. The decision was made to allocate one baud rate source to the modem port and share the other among the remaining ports in the Mensch configuration. This was based significantly upon the assumption that the keyboard, printer, and PC link ports could be preset and would probably remain constant. Likewise, it was assumed that the modem port would frequently need to change its baud rate to accommodate external connections such as: BBS systems and on-line services. The printer port may be configured to share the same baud rate source as the modem port, but care should be taken to avoid conflicts.

_DISP_LCD_HEADER

DESCRIPTION:

This subroutine will display the "MENSCH COMPUTER" header on the top line of the LCD screen. This may be useful to application programs building menus and formatted screens. The header will appear as:

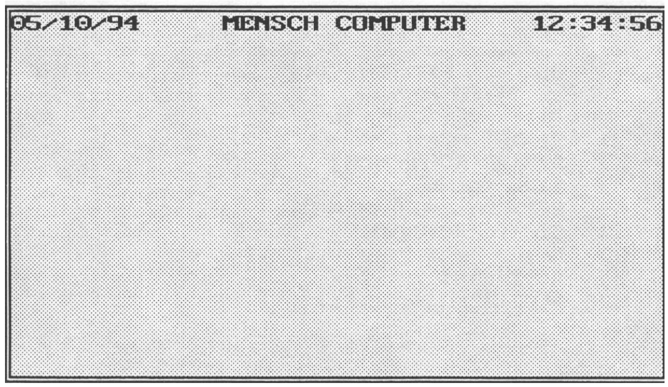

Figure 105
System Status Bar
Battery Condition: Normal (Charged)

VECTOR:

$00:80EF   _DISP_LCD_HEADER

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors are detected.

_DO_MAIN_MENU

DESCRIPTION:
This vector will transfer control to the MAIN MENU of the Mensch Operating System.

NOTE: This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:
$00:8107 _DO_MAIN_MENU

EXPECTS:
Not Applicable.

RETURNS:
Not Applicable

ERRORS:
Not Applicable

_ENGAGE_LOW_POWER_MODE

DESCRIPTION:
This vector will force the system into low-power mode.

NOTE: This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:
$00:806E _ENGAGE_LOW_POWER_MODE

EXPECTS:
No input arguments.

RETURNS:
This vector does not return.

ERRORS:
No meaningful errors.

_GET_A_PRINTER_BYTE (from printer port)

DESCRIPTION:
This subroutine will read the next available byte from the printer serial port input buffer. If the buffer is empty then the subroutine will return with the carry-bit set.

VECTOR:
$00:8041 _GET_A_PRINTER_BYTE

EXPECTS:
No input arguments.

RETURNS:
Received byte from printer serial port in 8-bit register-A.

ERRORS:
The carry-bit will return *clear* if a received data byte is available in 8-bit register-A. It will be *set* if no received data was available.

_GET_BIN_NUM

DESCRIPTION:

This subroutine will examine an ASCII hexadecimal digit string in a specified buffer and return the corresponding 16-bit binary number.

VECTOR:

$00:80E3 _GET_BIN_NUM

EXPECTS:

Buffer address for ASCII data as: Bank in 8-bit register-A

Offset in 16-bit register-X

RETURNS:

Normally returns with carry-bit *clear*, and the binary number in 16-bit register-X.

ERRORS:

Exceptions return with carry-bit *set*, and explanation code in the 8-bit register-A. Codes are:

$0D = no valid digits were received before an **ENTER** key was detected.

$1B = **ESC** (Escape) key was detected.

$FF = Non-Hexadecimal characters were entered.

NOTE:

This is a conversion subroutine which just examines data provided by the calling program. It does not, itself, "get" anything or perform any kind of *input* operations.

_GET_CONTROLLER_DATA

DESCRIPTION:

This subroutine will read the SEGA game controller and return data about the state of the switches.

VECTOR:

$00:8068 _GET_CONTROLLER_DATA

EXPECTS:

No input arguments.

RETURNS:

Returned byte in 8-bit register-A:

Bit #	Button
7 =	START
6 =	A
5 =	B
4 =	C
3 =	Right
2 =	Left
1 =	Down
0 =	Up

ERRORS:

Normally returns with the carry-bit = *clear*. It will return with the carry-bit = *set* if the controller port has been turned: OFF.

_GET_KEYBOARD_CHARACTER

DESCRIPTION:
This subroutine will read the next available byte from the keyboard serial port input buffer. If the buffer is empty, then the subroutine will return with the carry-bit set.

VECTOR:
$00:8023 _GET_KEYBOARD_CHARACTER

EXPECTS:
No input arguments.

RETURNS:
Received byte from keyboard serial port in 8-bit register-A.

ERRORS:
The carry-bit will return *clear* if received data is available in 8-bit register-A. It will be *set* if no received data was available.

_GET_MODEM_BYTE (from modem port)

DESCRIPTION:
This subroutine will read the next available byte from the modem serial port input buffer. If the buffer is empty, then the subroutine will return with the carry-bit set.

VECTOR:
$00:8050 _GET_MODEM_BYTE

EXPECTS:
No input arguments.

RETURNS:
Received byte from modem serial port in 8-bit register-A.

ERRORS:
The carry-bit will return *clear* if received data is available in 8-bit register-A. It will be *set* if no received data was available.

_GetGrStatus

DESCRIPTION:
This subroutine will retrieve the current status of the *color* and *fill* attributes.

VECTOR:
$00:81BF _GetGrStatus

EXPECTS:
No input arguments.

RETURNS:
Graphic status data returned in register-A:

Bit #	Meaning
0 – LSB	
1	
2	
3	Color: 0 = OFF 1 = ON
4	
5	
6	
7 – MSB	Fill: 0 = OFF 1 = ON

ERRORS:
No meaningful errors.

_GetPoint

DESCRIPTION:
This subroutine will read the current value of a specific graphic point on the LCD screen.

VECTOR:
$00:81BC _GetPoint

EXPECTS:
Horizontal coordinate in register-X. If this value exceeds 239, then the returned point value will be meaningless.

Vertical coordinate in register-Y. If this value exceeds 127, then the returned point value will be meaningless.

RETURNS:
Point value returned in register-A: 0 = White

NZ = Black

ERRORS:
No errors reported, but invalid coordinates will yield invalid results.

_HLine

DESCRIPTION:
This subroutine will plot a horizontal line on the LCD screen in graphics mode.

VECTOR:
$00:819E _HLine

EXPECTS:
Leftmost line origin coordinates in 16-bit register-X and 16-bit register-Y. Terminal coordinate in 8-bit register-A. Lines are drawn from left to right.

If the coordinate in register-X exceeds 239, then a value of zero will be used instead.

If the coordinate in register-A exceeds 239, then a value of 239 will be used instead.

RETURNS:
No arguments returned in register.

Normal operation returns with the carry-bit = *clear*.

ERRORS:
If the coordinate is register-Y exceeds 127, then this subroutine will return with the carry-bit = *set* and the line will not be drawn.

_INIT_DP_POINTER

DESCRIPTION:
This subroutine maintains an internal *stack* of Direct Page values, for later recovery by the _RESTORE_DP_POINTER subroutine. Upon entry, it saves the current DP register on this internal stack. It then sets the DP register to the value received in register-X

VECTOR:
$00:8110 _INIT_DP_POINTER

EXPECTS:
New *direct page pointer* value in 16-bit register-X.

RETURNS:
No arguments returned.

ERRORS:
No errors detected.

NOTE:
This subroutine performs no error checking. Its internal stack can only hold sixteen saved values for the DP register. Attempts to store more than the maximum (16) will *overflow* this stack and cause unpredictable results.

_Line

DESCRIPTION:
This subroutine will plot a line at any angle on the display in graphics mode.

VECTOR:
$00:8192 _Line

EXPECTS:
The starting coordinates of the line must be in the most significant bytes of 16-bit register-X and 16-bit register-Y. The ending coordinates of the line must be in the least significant bytes of the same registers.

RETURNS:
No arguments returned.

Normal operation returns with the carry-bit = *clear*.

ERRORS:
If either coordinate in register-X exceeds 239 or either coordinate in register-Y exceeds 127 then this subroutine will return with the carry-bit = *set* and the line will not be drawn.

_OS_SHELL

DESCRIPTION:
This is the normal entry vector to start the OS Shell program. This functions as a user command interpreter for IC card operations. A list of available commands will be displayed on the LCD screen in response to the "HELP" entry. Refer to the description of the MAIN MENU option for: "RUN PCMCIA SHELL". This provides an overview of how the OS Shell program operates.

NOTE: This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:
$00:8077 _OS_SHELL

EXPECTS:
Not Applicable.

RETURNS:
Not Applicable.

ERRORS:
Not Applicable.

_Point

DESCRIPTION:
This subroutine will plot one point at specified coordinates on the display in graphics mode.

VECTOR:
$00:81A1 _Point

EXPECTS:
Point coordinates in 16-bit register-X and 16-bit register-Y.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

NOTES:
If the coordinate in register-X exceeds 239 or the coordinate in register-Y exceeds 127, then this subroutine will not draw the point.

_PRINT_A_BYTE (Send via printer port)

DESCRIPTION:
This subroutine will queue one byte from 8-bit register-A to be sent to the serial printer port. The carry-bit will be set if the serial printer port cannot accept data, otherwise it will be clear upon return.

VECTOR:
$00:803E _PRINT_A_BYTE

EXPECTS:
Output byte in 8-bit register-A.

RETURNS:
No arguments returned.

ERRORS:
Carry-bit (Clear = OK / Set = Serial printer port cannot accept data.)

_PtCode

DESCRIPTION:
This subroutine will write an *escape* character (ESC = $1B) to the printer port followed by whatever data is in 8-bit register-A.

VECTOR:
$00:81A4 _PtCode

EXPECTS:
ASCII character in 8-bit register-A.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_PtLn

DESCRIPTION:
This subroutine will write a carriage-return (CR = $0D) and linefeed (LF = $0A) sequence to the printer port.

VECTOR:
$00:81A7 _PtLn

EXPECTS:
No input arguments.

RETURNS:
No argument returned.

ERRORS:
No meaningful errors are detected.

_PtScreen

DESCRIPTION:
This subroutine will dump the contents of the LCD screen to the printer. The DUMP flags may be configured to direct this dump to include either text or graphics or both. Refer to descriptions of: _SetGraph, _SetGraphText, and _SetText for more information.

VECTOR:
$00:81AA _PtScreen

EXPECTS:
No input arguments.

RETURNS:
No argument returned.

ERRORS:
No meaningful errors are detected.

_RD_COUNT

DESCRIPTION:
This subroutine may be used to check the current value of one of the five software timers. The software timers are 16-bit variables which may be initialized via the *_SET_COUNT* subroutine. The value in each timer will be decremented at regular (1/100th Sec.) intervals, until it reaches zero.

VECTOR:
$00:8125 _RD_COUNT

EXPECTS:
Timer number (0-4) in register-A.

RETURNS:
The current 16-bit counter value is returned in register-Y.

The state of the Zero-flag in the status register may be used to test the timer.

Register-A and register-X are saved upon entry and resorted before returning.

ERRORS:
Carry-bit (Clear = OK / Set = Invalid Counter Number specified.)

_RD_STPWTCH

DESCRIPTION:

The STOPWATCH function may be used as an event timer. This subroutine will read the current value of the STOPWATCH. The STOPWATCH is a 32-bit counter which counts up at regular (1/100th Sec.) intervals. The STOPWATCH may be reset to zero via the _CLR_STPWTCH subroutine.

VECTOR:

$00:812B _RD_STPWTCH

EXPECTS:

No input arguments.

RETURNS:

The 16-bit address (within Bank #0) of the 32-bit STOPWATCH counter in register-Y.

ERRORS:

No meaningful errors are detected.

_RESTORE_DP_POINTER

DESCRIPTION:

This subroutine removes last saved value of DP Pointer from an internal stack and stores it into the DP-Register. Pop's (No safety!)

VECTOR:

$00:8113 _RESTORE_DP_POINTER

EXPECTS:

No input arguments.

RETURNS:

No argument returned.

ERRORS:

No meaningful errors are detected.

NOTE:

This subroutine performs no error checking. It should only be used to recover values saved by the _INIT_DP_POINTER subroutine. If no values have been saved, its internal stacks can *underflow*. This could set the direct page (DP) register to a garbage value.

_RETRIEVE_CONTROLLER_STATUS

DESCRIPTION:
This subroutine will read the controller port.

VECTOR:
$00:8056 _RETRIEVE_CONTROLLER_STATUS

EXPECTS:
No input arguments.

RETURNS:
Controller status byte in 8-bit register-A, wherein:

Bit #7 (MSB) will be set (1) if the controller port has been disabled. Other bits will be irrelevant when this occurs.

If Bit #7 returns clear (0) then the lower seven bits will be the actual data read from the controller port.

ERRORS:
No meaningful errors.

NOTES:
If the controller has been turned: **OFF**, then the other returned status bits will be misleading. (Refer to the description of the *CONTROL_CONTROLLER_PORT* subroutine for more information.)

_RETRIEVE_KEYBOARD_STATUS

DESCRIPTION:
This subroutine will return the current status of the serial keyboard port.

VECTOR:
$00:8020 _RETRIEVE_KEYBOARD_STATUS

EXPECTS:
No input arguments.

RETURNS:
Keyboard port status byte in 8-bit register-A.

Bit #	Meaning
0 - LSB	Data in input buffer.
1	ESCape or ^C received.
2	XON/XOFF protocol mode.
3	Always zero (0).
4	DSR signal is TRUE (DSR0=0).
5	Echo mode is enabled.
6	Input buffer has overflowed.
7 - MSB	Output buffer has overflowed.

ERRORS:
No meaningful errors.

NOTES:
If the keyboard scanning has been turned: **OFF**, via the *SEND_BYTE_TO_KEYBOARD* subroutine, then the returned status bits may be misleading.

_RETRIEVE_MODEM_PORT_STATUS

DESCRIPTION:
This subroutine will return the current status of the serial modem port.

VECTOR:
$00:8044 _RETRIEVE_MODEM_PORT_STATUS

EXPECTS:
No input arguments.

RETURNS:
Modem port status byte in 8-bit register-A.

Bit #	Meaning
0 - LSB	Data in input buffer.
1	ESCape or ^C received.
2	XON/XOFF protocol mode.
3	Port power is ON.
4	DSR signal is TRUE (DSR2=0).
5	Echo mode is enabled.
6	Input buffer has overflowed.
7 - MSB	Output buffer has overflowed.

ERRORS:
No meaningful errors.

NOTES:
If the modem port supply voltage has been turned: **OFF**, via the *CONTROL_MODEM_PORT* subroutine, then the returned status bits may be misleading.

_RETRIEVE_PC_PORT_STATUS

DESCRIPTION:
This subroutine will return the current status of serial PC link port.

VECTOR:
$00:8053 _RETRIEVE_PC_PORT_STATUS

EXPECTS:
No input arguments.

RETURNS:
PC link port status byte in 8-bit register-A.

Bit #	Meaning
0 - LSB	Data in input buffer.
1	ESCape or ^C received.
2	XON/XOFF protocol mode.
3	Port power is ON.
4	DSR signal is TRUE (DSR3=0).
5	Echo mode is enabled.
6	Input buffer has overflowed.
7 - MSB	Output buffer has overflowed.

ERRORS:
No meaningful errors.

NOTES:
If the PC link port supply voltage has been turned: **OFF**, via the *CONTROL_PC_PORT* subroutine, then the returned status bits may be misleading.

_RETRIEVE_PRINTER_PORT_STATUS

DESCRIPTION:
This subroutine will return the current status of the serial printer port.

VECTOR:
$00:803B _RETRIEVE_PRINTER_PORT_STATUS

EXPECTS:
No input arguments.

RETURNS:
Printer port status byte in 8-bit register-A.

Bit #	Meaning
0 - LSB	Data in input buffer.
1	ESCape or ^C received.
2	XON/XOFF protocol mode.
3	Port power is ON.
4	DSR signal is TRUE (DSR1=0).
5	Echo mode is enabled.
6	Input buffer has overflowed.
7 - MSB	Output buffer has overflowed.

ERRORS:
No meaningful errors.

NOTES:
If the printer port supply voltage has been turned: **OFF**, via the *CONTROL_PRINTER_PORT* subroutine, then the returned status bits may be misleading.

_SELECT_MODEM_BAUD_RATE (for modem port only)

DESCRIPTION:

This subroutine will allow the program to reconfigure the baud rate generator which drives the modem serial port.

VECTOR:

$00:8047 _SELECT_MODEM_BAUD_RATE

EXPECTS::

Baud rate selection code in 8-bit register-A

0 =	110 Baud
1 =	150 Baud
2 =	300 Baud
3 =	600 Baud
4 =	1200 Baud
5 =	1800 Baud
6 =	2400 Baud
7 =	4800 Baud
8 =	9600 Baud
9 =	14400 Baud
A =	19200 Baud
B =	38400 Baud
C =	57600 Baud
D =	115000 Baud

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Unacceptable selection code)

_SEND_A_MODEM_BYTE (Send via modem port)

DESCRIPTION:

This subroutine will queue one byte from 8-bit register-A to be sent to the serial modem port. The carry-bit will be set if the serial modem port cannot accept data, otherwise it will be clear upon return.

VECTOR:

$00:804A _SEND_A_MODEM_BYTE

EXPECTS:

Output byte in 8-bit register-A.

RETURNS:

No argument returned.

ERRORS:

Carry-bit (Clear = OK / Set = Serial modem port cannot accept data.)

_SEND_BEEP

DESCRIPTION:
This subroutine will cause the speaker to beep.

VECTOR:
$00:8074 _SEND_BEEP

EXPECTS:
No input arguments.

RETURNS:
No argument returned.

ERRORS:
No meaningful errors.

_SEND_BYTE_TO KEYBOARD

DESCRIPTION:
This subroutine will queue one byte from 8-bit register-A to be sent to the serial keyboard port. The carry-bit will be set if the serial keyboard port cannot accept data, otherwise it will clear upon return.

Refer elsewhere in this manual to **Commanding The Keyboard** for a description of how this subroutine may be used.

VECTOR:
$00:8026 _SEND_BYTE_TO_KEYBOARD

EXPECTS:
Output byte in 8-bit register-A.

RETURNS:
No arguments returned.

ERRORS:
Carry-bit (Clear = OK / Set = Serial keyboard port cannot accept data.)

_SEND_DTMF_DIGIT (Send via tone generators)

DESCRIPTION:

This subroutine will use the tone generators: T5 and T6 gated to the speaker amplifier tot produce DTMF combinations of 55 ms duration.

VECTOR:

$00:8071 _SEND_DTMF_DIGIT

EXPECTS:

Output byte in 8-bit register-A, which must be the ASCII code corresponding to the desired DTMF key:

Character	Hex Code	Timer/Tone Generator: T5 (Hz)	Timer/Tone Generator: T6 (Hz)
0	$30	941	1336
1	$31	697	1209
2	$32	697	1336
3	$33	697	1477
4	$34	770	1209
5	$35	770	1336
6	$36	770	1477
7	$37	852	1209
8	$38	852	1336
9	$39	852	1477
A	$41	697	1633
B	$42	770	1633
C	$43	852	1633
D	$44	941	1633
#	$23	941	1477
Bell	$07	800	1200
*	$2A	941	1336

RETURNS:

No argument returned.

ERRORS:

Carry-bit (Clear = OK / Set = Unacceptable DTMF selection code.)

_SEND_MODEM_STRING

DESCRIPTION:

This subroutine will send a string of data or setup information to a HAYES-compatible modem.

VECTOR:

$00:804D _SEND_MODEM_STRING

EXPECTS:

Long pointer to the string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set.

RETURNS:

No arguments returned, however *a response will automatically be displayed on the LCD screen if the modem has been configured to return result codes.*

ERRORS:

Carry-bit (Clear = OK / Set = Command did not execute properly.)

_SET_COUNT

DESCRIPTION:

This subroutine may be used to initialize one of the five software timers. The software timers are 16-bit *down-counters* which may be read via the *_RD_COUNT* subroutine. The value in each timer will be decremented at regular (1/100th Sec.) intervals, until it reaches zero.

VECTOR:

$00:8122 _SET_COUNT

EXPECTS:

Timer number (0 – 4) in register-A.

Timeout value (Units = 1/100th Sec.) in 16-bit register-Y.

RETURNS:

All registers are saved upon entry and restored before returning.

ERRORS:

Carry-bit (Clear = OK / Set = Invalid Counter Number specified.)

_SetColor

DESCRIPTION:
This subroutine will set the COLOR flag used by other graphics plotting subroutines. The flag is used to determine whether pixel points should be plotted in white or black. (Clear = White / Set = Black)

VECTOR:
$00:8186 _SetColor

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_SetFill

DESCRIPTION:
This subroutine will set the FILL flag used by other graphics plotting subroutines. The flag is used to determine whether or not plotting subroutines should fill the shapes when they draw them.

(Set = Fill / Clear = No Fill)

VECTOR:
$00:818C _SetFill

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_SetGraph

DESCRIPTION:
This subroutine will configure the DUMP flags used by *_PtScreen* such that the subroutine will only dump the LCD graphics to the printer port.

VECTOR:
$00:81B0 _SetGraph

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_SetGraphText

DESCRIPTION:
This subroutine will configure the DUMP flags used by *_PtScreen* such that the subroutine will dump both LCD text and graphics to the printer port.

VECTOR:
$00:81AD _SetGraphText

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_SetText

DESCRIPTION:
This subroutine will configure the DUMP flags used by *_PtScreen* such that the subroutine will only dump the LCD text to the printer port.

VECTOR:
$00:81B3 _SetText

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors are detected.

_START

DESCRIPTION:
This is the normal restart point for the Mensch software. All previous setup conditions are lost.

> NOTE: This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:
$00:810A _START

EXPECTS:
Not Applicable.

RETURNS:
Not Applicable.

ERRORS:
Not Applicable.

_TIME_DATE_CHK

DESCRIPTION:
This subroutine will check and update the time and date on the first line of the display.

VECTOR:
$00:80EC _TIME_DATE_CHK

EXPECTS:
No input arguments.

RETURNS:
No arguments returned.

ERRORS:
No meaningful errors.

_VLine

DESCRIPTION:
This subroutine will plot a vertical line on the LCD screen in graphics mode.

VECTOR:
$00:819B _VLine

EXPECTS:
Line origin coordinates in 16-bit register-X and 16-bit register-Y. Terminal coordinate in 8-bit register-A. Lines are drawn from top to bottom.

If the coordinate in register-Y exceeds 127 then a value of zero will be used.

If the coordinate in register-A exceeds 127 then a value of 127 will be used.

RETURNS:
No arguments returned in registers.

Normal operation returns with the carry-bit = *clear*.

ERRORS:
If the coordinate in register-X exceeds 239 then this subroutine will return with the carry-bit = *set*, and the line will not be drawn.

_WrDec

DESCRIPTION:
This subroutine will write a 16-bit unsigned integer as a positive number (0-65535) in ASCII Decimal digits to the LCD screen.

VECTOR:
$00:8180 _WrDec

EXPECTS:
Output value in 16-bit register-X

RETURNS:
No arguments returned in registers.

ERRORS:
No meaningful errors are detected.

Alter_Memory

DESCRIPTION:

This is the subroutine invoked by typing the **'M'** command at the monitor prompt. Basically, **Alter_Memory** will request an address and accept input via the selected I/O streams.

The user must enter six ASCII-Hex digits to form a 24-bit address. The input format is:
BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After a valid address has been entered, **Alter_Memory** prints a single line memory dump starting at the specified address. It then prints a second line repeating the address and positioning under the contents of the first location. The user may input new hexadecimal character data, one byte at a time. This subroutine will automatically position under the next location as values are entered.

Entering a **SPACE** ($20) character skips the current location, without changing it, and positions on the next. The user may terminate this operation at anytime by typing **ENTER** ($0D).

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E000 Alter_Memory

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if any errors were detected.

ASCBIN

DESCRIPTION:

This subroutine will convert two ASCII HEX[15] 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 **TEMP**. 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 **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **ASCBIN** subroutine.

VECTOR:

$00:E087 ASCBIN

EXPECTS:

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 the global variable: **TEMP** ($00:0070).

RETURNS:

The resulting binary value will be returned in register-A.

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if either parameter was not an ASCII HEX digit.

BACKSPACE

DESCRIPTION:

This subroutine will output a **BS** (Backspace = $08) character to each of the selected output streams. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **BACKSPACE** subroutine.

VECTOR:

$00:E003 BACKSPACE

EXPECTS:

No input arguments.

RETURNS:

The carry-bit will always be *clear* upon completion.

ERRORS:

No errors reported.

---

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

BIN2DEC

DESCRIPTION:

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

Example:

```
        LDA HOURS           ; current HOURS (00-$17)
        JSL BIN2DEC         ; make it decimal (00-$23)
        JSL SEND_HEX_OUT
        …
```

VECTOR:

$00:E08B BIN2DEC

EXPECTS:

Binary value ($00-$63) in register-A.

RETURNS:

Equivalent value in packed decimal format ($00-$99) in register-A.

Values larger than $63 will not be properly converted.

The carry-bit will always be *clear* upon completion.

ERRORS:

No errors reported. 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.

BINASC

DESCRIPTION:

This subroutine will convert an 8-bit binary value in register-A into two ASCII HEX characters. Register-A returns the least significant character in ASCII. **Temp+1** returns the most significant character. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **BINASC** subroutine.

VECTOR:

$00:E08F BINASC

EXPECTS:

Binary value in 8-bit register-A.

RETURNS:

Least significant character in ASCII in Register-A.

Most significant character in ASCII in the global variable: **TEMP+1** ($00:0071).

ERRORS:

No errors reported.

CHANGE_DIRECTORY (Reserved!)

DESCRIPTION:

This vector currently invokes a non-functional "dummy" subroutine. It is reserved for subdirectory operations in future versions of the Mensch Operating System.

VECTOR:

$00:80CE CHANGE_DIRECTORY

EXPECTS:

Not Applicable.

RETURNS:

Not Applicable.

ERRORS:

Not Applicable.

CLEAR_LCD_DISPLAY (Entire text and/or graphics area)

DESCRIPTION:

This subroutine will erase the entire LCD display area. It can selectively clear only the text or graphics memory, or both, or neither.

VECTOR:

$00:802C CLEAR_LCD_DISPLAY

EXPECTS:

Control code in 8-bit register-A:

$00	=	No clearing operation.
$01	=	Clear text display memory only.
$02	=	Clear graphics display memory only.
$03	=	Clear all LCD memory.
Other	=	Invalid.

RETURNS:

No arguments returned.

ERRORS:

The carry-bit normally will return *clear*, but will be *set* if the control value in 8-bit register-A was invalid.

CAUTION:

It is not meaningful to call this subroutine if the supply voltage to the LCD display has been disabled. (Refer to *CONTROL_DISPLAY* for more information.)

NOTE:

Whenever the text display memory is cleared, the current text cursor position will be reset to: line=0, column=0. Likewise, whenever the graphics display memory is cleared, the current pixel coordinates will be reset to: row=0, column=0. In both cases, this corresponds to the upper-left corner of the display.

CLEAR_TO_END_OF_LINE (Text line)

DESCRIPTION:

This subroutine will erase the LCD screen from the current text cursor position to the end of the current text line. The text cursor will remain in its original position.

VECTOR:

$00:8032 CLEAR_TO_END_OF_LINE

EXPECTS:

No input arguments.

RETURNS:

No arguments returned, no registers changed.

ERRORS:

No meaningful errors.

CAUTION:

It is not meaningful to call this subroutine if the supply voltage to the LCD display has been disabled. (Refer to *CONTROL_DISPLAY* for more information.)

CONTROL_SPEAKER_AMP

DESCRIPTION:

This subroutine can turn the power to the speaker amplifier ON and OFF.

VECTOR:

$00:8104 CONTROL_SPEAKER_AMP

EXPECTS:

Control value in 8-bit register-A:

Zero	=	Disable amplifier supply voltage.
Any Non-Zero	=	Enable power to amplifier.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

CONTROL_TONES

DESCRIPTION:

This subroutine will configure timers: **T5** and **T6**, and gate either or both tone generators to the audio outputs: **TG0** and **TG1**. Configuration values for the timers may need to be calculated for each implementation. The value necessary to produce specific tones are dependent upon the frequency of the fast clock (FCLK) [16].

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **CONTROL_TONES** subroutine.

The speaker amplifier may be enabled/disabled independently. Programmers should use the *CONTROL_SPEAKER_AMP* subroutine before calling this routine, to make certain that the speaker amplifier output is enabled.

VECTOR:

$00:E009 CONTROL_TONES

EXPECTS:

Control code in 8-bit register-A:	0	=	Both tone generators disabled.
	1	=	Tone generator TG0 is enabled.
	2	=	Tone generator TG1 is enabled.
	3	=	Both tone generators enabled.
	Other	=	Invalid.

Configuration value for timer: **T5** in 16-bit register-X.

Configuration value for timer: **T6** in 16-bit register-Y.

RETURNS:

No arguments returned.

ERRORS:

The carry-bit normally will return *clear*, but will be *set* if the control code in 8-bit register-A was invalid.

CREATE_DIRECTORY (Reserved!)

DESCRIPTION:

This vector currently invokes a non-functional "dummy" subroutine. It is reserved for subdirectory operations in future versions of the Mensch Operating System.

VECTOR:

$00:80C8 CREATE_DIRECTORY

EXPECTS:

Not Applicable.

RETURNS:

Not Applicable.

ERRORS:

Not Applicable.

---

[16] A thorough description of the algorithm is provided in: W65C265S INFORMATION SPECIFICATION AND DATA SHEET. It also includes precalculated tables of value for typical FCLK frequencies and commonly needed tones: DTMF, modems, ect. This document and related literature is available from WDC.

DIR_COMMD

DESCRIPTION:

This subroutine outputs a file directory list on the LCD screen. If there are no files, then a message will appear. If the list requires more than one screen, then any keypress will page to the next screen.

VECTOR:

$00:80D4 DIR_COMMD

EXPECTS:

Card specifier in 8-bit register-A: $00 = Low Card Slot
$01 = High Card Slot

RETURNS:

Normal operation of this subroutine will return with the carry-bit = *clear*, and no other relevant arguments.

ERRORS:

If errors or exceptions were detected by this subroutine, it will return with the carry-bit = *set*. An error code for clarification will be returned in register-A. The error code may be interpreted as follows:

$01 = No card in slot
$02 = Invalid card

DISP_LCD_STRING

DESCRIPTION:

This subroutine will write a character string to the LCD display, at the current text cursor position, if the text display has been enabled. The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set.

Programmers should note that this subroutine should not be used unless they know where the cursor is positioned. If the cursor coordinates are outside the display area, the string will not appear. Refer to descriptions of the *CLEAR_LCD_DISPLAY* and *POSITION_TEXT_CURSOR* subroutines for additional information.

VECTOR:

$00:8038 DISP_LCD_STRING

EXPECTS:

Bank address of string in 8-bit register-A.

Pointer to string in 16-bit register-X.

RETURNS:

No arguments returned.

ERRORS:

The carry-bit normally will return *clear*. There are no meaningful errors specific to this subroutine, but it does call the *WRITE_LCD_CHARACTER* subroutine. Any errors detected at that level will be passed back by this subroutine. (Refer to the description of the *WRITE_LCD_SUBROUTINE* for more information.)

DISPLAY_PCMCIA_ERROR

DESCRIPTION:

This vector will display a PCMCIA error message on the LCD screen at the current text cursor position.

VECTOR:

$00:8080 DISPLAY_PCMCIA_ERROR

EXPECTS:

PCMCIA error code in 8-bit register-A. The error codes will be translated to text as follows:

Error Code	Message Text
01H	"NO CARD IN THAT SLOT"
02H	"NOT A VALID CARD"
03H	"CARD IS TOO BIG FOR SLOT"
04H	"SLOT SPECIFIED NOT VALID"
09H	"FORMAT COMPLETED"
10H	"SPECIFIED CARD NOT LOGGED"
11H	"SECTORS OUT OF RANGE"
12H	"CARD IS WRITE PROTECTED"
13H	"INVALID CARD FORMAT"
20H	"FILE OPEN MODE IS INVALID"
21H	"FILE NOT FOUND"
22H	"MAX FILES ALREADY OPEN"
23H	"FILE SIZE WAS ZERO"
24H	"END OF FILE REACHED"
30H	"FILE OPENED AS READ ONLY"
All Other	"UNDEFINED PC CARD ERROR"

RETURNS:

No arguments returned.

ERRORS:

No errors detected or reported.

DO_LOW_POWER_PGM

DESCRIPTION:

This vector will force the system into low-power mode. Basically, this involves the following steps:

1. Reset the *stack pointer* to $00:01FF.
2. Turn OFF all I/O.
3. Shut down all chip selects.
4. Perform *low power mode* maintenance loop.
   - Service interrupts from timer #1, 1/second.
     Update time-of-day clock/calendar and alarm.
   - Execute *User Check Program* subroutine located at: $00:01C0.

Step #4 will repeat, keeping the W65C265 in *low power mode*. This will continue until one of the following events:

- System **RESET** occurs.
- The **Alarm** function times out.
- The *User Check Program* subroutine initiates exit from *low power mode* to begin normal operation again.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of *low power mode*.

> NOTE: This is an entry vector, not a subroutine.
> **IT WILL NOT RETURN!**

VECTOR:

$00:E00C DO_LOW_POWER_PGM

EXPECTS:

Not Applicable.

RETURNS:

Not Applicable.

ERRORS:

Not Applicable.

Dump_1_line_to_Output

DESCRIPTION:

This subroutine will request the *starting address* via the selected output streams and accept responses via any of the selected input streams.

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_1_line_to_Output** will write a formatted header and one line, sixteen bytes, of memory dump data to each of the selected output streams.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E015 Dump_1_line_to_Output

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

Dump_1_line_to_Screen

DESCRIPTION:

This subroutine will request the *starting address* and accept input via any of the selected input streams.

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_1_line_to_Screen** will write a formatted header and two lines of eight bytes of memory dump data to each of the selected output streams.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E018 Dump_1_line_to_Screen

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

NOTE:

The name of this subroutine may be confusing to some readers. It may help to remember dump operations always begin with a header, and that one *dump line* always includes sixteen bytes from its starting address. When dumping to the 40-column LCD screen, this data must be reformatted as a header and two *display lines*. This is a generic output subroutine which will dump in *display line* format to all active output streams, even if the actual LCD display has been disabled.

Dump_It

DESCRIPTION:

This subroutine will perform a memory dump operation to all selected output streams. The memory range and dump configuration must be provided by the caller. It performs no error checking, as it assumes validation will be performed before parameters are passed. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **Dump_It** subroutine

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E027 Dump_It

EXPECTS:

Dump configuration parameter in 8-bit register-A.

Least significant bit:	#0	=	Output S28 + byte-count.
	1	=	Format for LCD (40 char/line).
Note: Some possible	2	=	Add spaces between data bytes & HEADER.
combinations	3	=	Add checksum.
are invalid or	4	=	8 bytes not 16 bytes per line.
unpredictable!	5	=	ONE LINE ONLY. *(Not Used by: Dump-It)*
	6	=	ASCII not Hex data.
Most significant bit:	#7	=	*(Not Used.)*

Non-zero number of dump data lines per page in 16-bit register-X. (Note: A header may also be printed.)

The 3-byte starting address must be loaded into the global variable: **TMP0** ($00:005D), **TMP0+1**, and **TMP0+2**. The least significant byte (LSB) of the 3-byte address must reside in **TMP0** and the MSB must be in **TMP0+2**.

The 3-byt ending address must be locked into the global variable: **TMP2** ($00:0063), **TMP2+1**, and **TMP2+2**. The least significant byte (LSB) of the 3-byte address must reside in **TMP2** and the MSB must be in **TMP2+2**.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors are detected or reported.

Dump_to_Output

DESCRIPTION:

This subroutine will request the *starting and ending addresses* and accept input via any of the selected input streams.

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_to_Output** will write a **Form-Feed** ($0C) and formatted header line to each of the selected output streams. This will be followed by up to sixty lines, of sixteen bytes each, of memory dump data. If the range of memory requires more than sixty lines of dump data, then another from-feed/header will be generated before more dump data is sent. This cycle will repeat until the entire specified block of memory has been dumped.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E01B Dump_to_Output

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

Dump_to_Printer

DESCRIPTION:

This subroutine will request the *starting and ending addresses* and accept input via the selected I/O streams.

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_to_Printer** will write a **Form-Feed** ($0C) and formatted header line each of the selected output streams. This will be followed by up to sixty lines, of sixteen bytes each, of memory dump data. If the range of memory requires more than sixty lines of dump data, then another form-feed/header will be generated before more dump data is sent. This cycle will repeat until the entire specified block of memory has been dumped.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E01E Dump_to_Printer

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

Dump_to_Screen

DESCRIPTION:

This subroutine will request the *starting and ending addresses* and accept input via the selected I/O streams.

The user must enter six ASCII-Hex digits to form the 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_to_Screen** will write a formatted header line to each of the selected output streams. This will be followed by up to twelve lines, of eight bytes each, of memory dump data. If the range of memory requires more than twelve lines of dump data, then the **Dump_to_Screen** subroutine will pause for input from any selected input streams. The user may enter any character to acknowledge this pause. Another header will be generated before more dump data is sent. This cycle will repeat until the entire specified block of memory has been dumped.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E021 Dump_to_Screen

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

Dump_to_Screen_ASCII

DESCRIPTION:

This subroutine will request the *starting and ending addresses* and accept input via the selected I/O streams.

The user must enter six ASCII-Hex digits to form the 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **Dump_to_Screen_ASCII** will write up to twelve lines, of sixteen bytes each, of ASCII dump data. Values which do not translate to pintable ASCII characters will appear as an apostrophe ('). If the range of memory requires more than twelve lines of dump data, then the **Dump_to_Screen_ASCII** subroutine will pause for input from any selected input stream. The user may enter any character to acknowledge this pause. Up to twelve more lines of dump data will be sent. This cycle will repeat until the entire specified block of memory has been dumped.

The final dump will be followed by a pause. Again, the user may enter any character to acknowledge this pause.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E024 Dump_to_Screen_ASCII

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

DUMPREGS

DESCRIPTION:

This is the subroutine invoked by typing the '**R**' command at the monitor prompt. Basically, **DUMPREGS** will write a formatted display of the register values as they were saved at the most recent monitor prompt. These values were used to initialize the registers prior to the monitor releasing control. This output will be sent to each of the selected output streams.

> NOTE: This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E00F DUMPREGS

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* upon completion.

ERRORS:

No errors reported.

DumpS28

DESCRIPTION:

This is the subroutine invoked by typing the '**W**' command at the monitor prompt. Basically, **DumpS28** will request *lowest and highest addresses* via each selected input stream. Then, it will dump the specified memory block in S28 loader format to all selected output streams. The last record written will begin with "S8", indicating that none follow.

S28 Format:		S2LLHHMMLWDDDDDDDD…CC
	where:	S2 = literally the ASCII characters: "S2",
		LL = length of the data + 4,
		HH = high byte of the address,
		MM = middle byte of the address,
		LW = low byte of the address,
		DD = one byte of data, next byte, ect
		CC = checksum (1's complement of the sum of the length, address, and data bytes.)

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After valid addresses have been entered, **DumpS28** will write S28 records for the entire memory block. If the *lowest address* is greater than the *highest address*, then only one sixteen byte record will be dumped. It will begin with the specified *lowest address*.

Note: This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operations of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E012 DumpS28

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

FCLOSE

DESCRIPTION:
This routine Closes the specified file in the PCMCIA Dos-compatible file emulation. If the specified file was not open, the subroutine will just return normally, without problems.

VECTOR:
$00:8092 FCLOSE

EXPECTS:
Register-A = File Handle of File to be Closed.

RETURNS:
If File Was Closed: Carry Bit = *Clear*.

> Note: Since the result will always be a closed file, the carry-bit will always return *clear*.

ERRORS:
No meaningful errors returned.

FDELETE

DESCRIPTION:
This routine Deletes the specified file from the card.

VECTOR:
$00:80C2 FDELETE

EXPECTS:
Register-X: Address of Filename Information
Register-A: Bank Code of Filename Information

RETURNS:
If File is Deleted:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$01 = No Card Found in Specified Slot.
$21 = File Not Found

FGETBLOCK

DESCRIPTION:

This routine reads a specific number of bytes from the specified file and stores the byte in a specified storage area.

VECTOR:

$00:80B3 FGETBLOCK

EXPECTS:

All parameters to this routine must be passed on the stack.

SP+4: File Handle of file.
SP+5: Address of Storage Location (24 bits).
SP+7: Length of Block to Read

Note: It is the responsibility of the Calling routine to restore the stack pointer upon return from this function.

RETURNS:

If Block read successfully:

Carry Bit = Clear

ERRORS:

Else, Carry Bit = Set

Register-A = Error Code:

$24 = End of File reached during read..

FGETC

DESCRIPTION:

This routine Reads a single byte from the specified file.

VECTOR:

$00:80AA FGETC

EXPECTS:

Register-Y: File handles to read from.

RETURNS:

If byte read successfully:

Carry Bit = Clear
Register-A = Byte Just Read

ERRORS:

Else, Carry Bit = Set

Register-A = Error Code:

$24 = End of File reached

FGETS

DESCRIPTION:
This routine reads a string of data from the specified file. Routine will continue reading from the file until a byte of $00 or EOF is found.

VECTOR:
$00:80B0 FGETS

EXPECTS:
Register-Y: File Handle to Read From
Register-X: Address to String Storage Location
Register-A: Bank Address of String Storage Location

RETURNS:
If String is Read successfully:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$24 = End of File reached

FGETW

DESCRIPTION:
This routine reads a Word of data from the specified file.

VECTOR:
$00:80AD FGETW

EXPECTS:
Register-Y: File Handle to read from.

RETURNS:
If Word is read successfully:
Carry Bit = Clear
Register-X = Word just read from file.

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$24 = End of File reached

FILELENGTH

DESCRIPTION:
This routine returns the current size of the specified file.

VECTOR:
$00:80BC FILELENGTH

EXPECTS:
Register-A: File Handle of File to test.

RETURNS:
Register-X: Low word of File Length
Register-Y: High word of File Length

ERRORS:
No meaningful errors returned.

FILL_Memory

DESCRIPTION:

This is the subroutine invoked by typing the '**F**' command at the monitor prompt. Basically, **FILL_Memory** will request *starting* and *ending addresses*, and a *fill constant* via all selected output streams. It will accept responses from any selected input streams.

The user must enter six ASCII-Hex digits to form each 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After a valid addresses have been entered, **FILL_Memory** will expect a *fill constant* as two hexadecimal characters. It will then write the *fill constant* value to every location in the specified memory block.

> Note: This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E02A FILL_Memory

EXPECTS:

No input arguments. This is an interactive subroutine which requests parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if any errors were detected.

FINDFIRST

DESCRIPTION:
This routine searches for a filename on a PCMCIA card. The filename must have been split up by FNSPLIT prior to calling this routine.

VECTOR:
$00:80B9 FINDFIRST

EXPECTS:
Register-X: Address of File Data Structure (FDS).
Register-A: Bank Code of File Data Structure (FDS).

RETURNS:
If file is found:
Carry Bit = Clear
Register-X = Address of Directory Entry for File
Register-A = Bank Code of Directory Entry.

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$21 = No File Found

FNSPLIT

DESCRIPTION:
This routine takes a string input and breaks it up into the separate pieces of a filename including card slot, path string, filename, and extension.

VECTOR:
$00:80B6 FNSPLIT

EXPECTS:
Pointer (24-bits) to file name string to be parsed:

Bank of filename string in 8-bit register-A.
Offset address of filename string in 16-bit register-X.

Offset pointer (in Bank #0) to store the parts in 16-bit register-Y.

RETURNS:
Filename data is separated and stored in file structure table for use by other PCMCIA routines.

ERRORS:
No meaningful errors returned.

NOTE:
The file name string should have one of the following formats:

LO:*name.extension*
or
HI:*name.extension*

Wherein: *name.extension* represent a valid MS-DOS file name[17].

---

[17] Future versions of the Mensch Operating System will allow multiple levels of directories and will provide complete *pathname* support.

FOPEN

DESCRIPTION:
This routine Opens the given file on the specified PCMCIA card.

VECTOR:
$00:808F FOPEN

EXPECTS:
Register-X = Address of Path String in Memory
Register-A = Bank address of Path String in Memory
Register-Y = File Open Code:
- 'r': Open file in read-only mode.
- 'w': Open File for writing. Overwrite existing file.
- 'a': Open file for append, or create new file.
- 'R': Open file for updating (Read and Write).
- 'W': Create file in update mode (Read and Write).
- 'A': Open file for append, or create new file.

RETURNS:
If File was successfully opened:
Carry Bit = Clear
Register-A = File Handle for subsequent usage

ERRORS:
Carry Bit = Set

Register-A = Error Code:
- $01 = No Card Found in the Slot.
- $02 = Boot Sector on Card is Invalid.
- $13 = Invalid File Access Table (FAT).
- $20 = Invalid Open mode.
- $21 = File no found for Read Mode.
- $22 = Too many files open to open this file.

FORMAT

DESCRIPTION:
This routine formats a PCMCIA RAM Card installed in the specified PCMCIA slot. The format used is standard MSDOS format.

VECTOR:
$00:807A FORMAT

EXPECTS:
Register-A: PCMCIA Slot Code: $00 = Low Card Slot
$01 = High Card Slot

RETURNS:
If RAM Card formatted properly:
Register-A = Number of 64K Banks on Card
Carry Bit = Clear

ERRORS:
If Card not formatted
Carry Bit = Set
Register-A = Error Code:
- $01 = No Card Found in the specified slot.
- $03 = Card is to big for the specified slot.

FPUTBLOCK

DESCRIPTION:
Writes a specific number of bytes into the specified file.

VECTOR:
$00:80A7 FPUTBLOCK

EXPECTS:
All parameters to this routine must be passed on the stack:
SP+4 = File Handle to write to.
SP+5 = Source Address (16 bits).
SP+7 = Bank Address of Source (8 bits).
SP+8 = Length to write (16 bits).

It is the responsibility of the calling routine to restore the stack pointer after this routine returns.

RETURNS:
If Block was written successfully:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$30 = File opened in Read-Only mode.

FPUTC

DESCRIPTION:
This routine writes a single byte to the specified file.

VECTOR:
$00:809E FPUTC

EXPECTS:
Register-A = Byte to be written into file.

Register-Y = File Handle of open file to write to.

This routine assumes that a file has been previously opened and will NOT check for an open file.

RETURNS:
If byte was written:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$30 = File is open as Read-Only

FPUTS

DESCRIPTION:
This routine writes a String of Bytes to the specified file. The string must be terminated by a $00 byte.

VECTOR:
$00:80A4 FPUTS

EXPECTS:
Register-Y: File Handle of Opened File.

Register-X: Address of String in memory.

Register-A: Bank Address of String.

RETURNS:
If string is written successfully:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$30 = File open in Read-Only mode.

FPUTW

DESCRIPTION:
This routine Write a Word (16 bits) of data to the specified file.

VECTOR:
$00:80A1 FPUTW

EXPECTS:
Register-X: Word to be written into file.

Register-Y: File Handle of file to be written to.

RETURNS:
If word was written successfully:
Carry Bit = Clear

ERRORS:
Else, Carry Bit = Set
Register-A = Error Code:
$30 = File was opened in Read-Only mode.

FSEEK

DESCRIPTION:
This routine moves the file pointer to the specific location within the specified file.

VECTOR:
$00:8098 FSEEK

EXPECTS:
File Handle in 8-bit register-A.

Desired byte location in file as 32-bit offset:

Low word of position to seek to in 16-bit register-X.

High word of position to seek to in 16-bit register-Y.

RETURNS:
No output arguments.

ERRORS:
No meaningful errors returned.

NOTE:
Internally, F_NEXT_ADDRESS ($00:102B) will point to the current address within the specified file. This pointer will be used by: FGETC, FGETW, FGETB, and other subroutines when accessing the file.

GET_3BYTE_ADDR

DESCRIPTION:

This subroutine accepts six ASCII-Hex digits, from the input streams selected by the CONTROL_INPU, to form a 24-bit address.

The input format is: BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address, to all output streams enabled by CONTORL_OUTPUT. "AAAA" is the offset address within the bank.

Refer to the description of CONTROL_INPUT for details about configuring input stream and selecting input sources.

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_3BYTE_ADDR** subroutine.

VECTOR:

$00:E02D GET_3BYTE_ADDR

EXPECTS:

No input arguments.

RETURNS:

No output arguments.

The 3-byte result is returned in the global variables: **TMP2** ($00:0063), **TMP2+1**, and **TMP2+2**. The bytes are ordered such that: **TMP2**=LSB and **TMP2+2**=MSB.

The subroutine will return with the carry-bit=*clear* if a proper 6-digit address has been received.

ERRORS:

The carry-bit will be returned *set* if any non-digit character is detected before all six ASCII-Hex digits have been received.

Get_Address

DESCRIPTION:

This subroutine will write the following string to all selected output streams:

"Enter Address: BB:AAAA"

IT then performs: **GET_3BYTE_ADDR** which accepts six ASCII-Hex digits, from any selected input stream, to form a 24-bit address.

The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit address, to each of the selected output streams. "AAAA" is the offset address within the bank.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **Get_Address** subroutine.

VECTOR:

$00:E042   Get_Address

EXPECTS:

No input arguments.

RETURNS:

No output arguments.

The 3-byte result is returned in the global variables: **TMP2** ($00:0063), **TMP2+1**, and **TMP2+2**. The bytes are ordered such that: **TMP2**=LSB and **TMP2+2**=MSB.

The subroutine will return with the carry-bit=*clear* if a proper 6-digit address has been received.

ERRORS:

The carry-bit will be returned *set* if any non-digit character is detected before all six ASCII-Hex digits have been received.

GET_ALARM_STATUS

DESCRIPTION:

This subroutine will retrieve the current status of the system alarm. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_ALARM_STATUS** subroutine.

EXPECTS:

No input arguments.

RETURNS:

Alarm status returned in 8-bit register-A:

Zero	=	Alarm has not been set.
Any Non-Zero	=	Alarm has been set.

The carry-bit will be set if the alarm has been triggered, otherwise it will be clear upon return.

ERRORS:

No meaningful errors.

NOTE:

Calling this subroutine will also automatically reset the alarm, if it has been triggered. Non-destructive testing may be accomplished by accessing the alarm flag directly from page #0. (Refer to the source code listing for specific details.)

GET_BYTE_FROM_PC

DESCRIPTION:

This subroutine will read the next available byte from the PC link serial port #3 input buffer. If the buffer is empty, then the subroutine will return with the carry-bit set. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_BYTE_FROM_PC** subroutine.

VECTOR:

$00:E033 GET_BYTE_FROM_PC

EXPECTS:

No input arguments.

RETURNS:

Received byte from PC link serial port in 8-bit register-A.

ERRORS:

The carry-bit will return *clear* if a received data byte is available in 8-bit register-A. It will be *set* if no received data was available.

GET_CHR

DESCRIPTION:

This subroutine will get a single character from the selected INPUT streams. IT will sample the selected input sources and returns the first character detected.

Refer to the description of CONTROL_INPUT for details about configuring the input stream and selecting input sources.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_CHR** subroutine.

VECTOR:

$00:8036 GET_CHR

EXPECTS:

No input arguments.

RETURNS:

Normally returns with carry-bit *clear*, and received character in 8-bit register-A. All other registers are saved upon entry and restored before returning.

ERRORS:

Exception returns carry-bit *set* if no input sources are enabled. No other meaningful errors are detected.

Get_E_Address

DESCRIPTION:

This subroutine writes the following prompt string to all selected output streams:

"Enter Highest Address: BB:AAAA"

It then performs: **GET_3BYTE_ADDR** which accepts six ASCII-Hex digits, from any selected input stream, to form a 24-bit address.

The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address, to each of the selected output stream. "AAAA" is the offset address within the bank.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **Get_E_Address** subroutine.

VECTOR:

$00:E045   Get_E_Address

EXPECTS:

No input arguments.

RETURNS:

No output arguments.

The 3-byte result is returned in the global variables: **TMP2** ($00:0063), **TMP2+1**, and **TMP2+2**. The bytes are ordered such that: **TMP2**=LSB and **TMP2+2**=MSB.

The subroutine will return with the carry-bit = *clear* if a proper 6-digit address has been received.

ERRORS:

The carry-bit will be returned *set* if any non-digit character is detected before all six ASCII-Hex digits have been received.

GET_HEX

DESCRIPTION:

This subroutine will get the next character from any selected input stream into register-A. If it is a **SPACE** ($20) character, then **GET_HEX** will return with the carry-bit=*set*. Otherwise, the subroutine will accept another character.

If either byte is not an ASCII Hex character, then this subroutine will also return with the carry-bit=*set*, but register-A will be cleared to: $00.

If both bytes were ASCII Hex characters, the pair will be evaluated to produce a single binary byte of the value represented by the two hexadecimal digits. The **GET_HEX** subroutine will return with the value in register-A and the carry-bit=*set*.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_HEX** subroutine.

VECTOR:

$00:E039 GET_HEX

EXPECTS:

No input arguments.

RETURNS:

The carry-bit will be *set*, and 8-bit register-A = $20 if the first character was a **SPACE**.

or

The carry-bit will be *set*, and 8-bit register-A = $00 if either input byte was not an ASCII HEX digit.

or

The carry-bit will be *clear* and 8-bit register-A will contain the binary value represented by the two hexadecimal digits input.

ERRORS:

No errors reported.

Get_HiLo

DESCRIPTION:

This subroutine will send the following message to all selected output streams:

'Card = HI or LO:'

It then waits for a response, terminated by **ENTER** ($0D) or **ESC** (Escape = $1B) from the selected input streams. The first two characters of the response will be evaluated for the answer: 'HI', 'hi', 'LO', or 'lo'.

VECTOR:

$00:80F5 Get_HiLo

EXPECTS:

No input arguments.

RETURNS:

Normally returns with the carry-bit *clear* and a code in register-A;

0 = LO
1 = HI

ERRORS:

The carry-bit will return *set* if a null string was entered, or if the entry was terminated with an **ESC** character, or if the response was not acceptable. The contents of register-A will be invalid.

GET_MODEM_RESPONSE (from modem port)

DESCRIPTION:

This subroutine will allow a programmer to input a string of response data from a "Hayes-compatible" modem attached to the MODEM port (#2).

VECTOR:

$00:8116 GET_MODEM_RESPONSE

EXPECTS:

Long pointer to a buffer for received string as follows:

Bank address of buffer in 8-bit register-A.

Offset address of buffer in 16-bit register-X.

The received string will be automatically terminated with a NULL ($00) character.

The buffer size must be at least one byte larger than the received string. (1 ≤ *size* ≤ 65535)

RETURNS:

No arguments returned, however the received string should be in the specified buffer.

The carry-bit will return *clear* if received data was available.

ERRORS:

If no received data was available, within a timeout interval (approximately: 1 second), this subroutine will return with the carry-bit *set*.

GET_PUT_CHR

DESCRIPTION:

This subroutine inputs and outputs a character to selected ports. It accepts the next available character from the selected INPUT streams. Then it sends the character to all activated OUTPUT streams, except the one corresponding to the specific character's source.

Refer to the description of CONTROL_INPUT for details about configuring the input stream and selecting input sources.

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_PUT_CHR** subroutine.

VECTOR:

$00:E03C GET_PUT_CHR

EXPECTS:

No input arguments.

RETURNS:

Returns normally with carry-bit *clear* and the received character in 8-bit register-A. All other registers are saved upon entry and restored before returning.

ERRORS:

Exception returns carry-bit *set* if no input sources are enabled. No other meaningful errors are detected.

Get_S_Address

DESCRIPTION:

This subroutine writes the following prompt string to all selected output streams:

"Enter Lowest Address: BB:AAAA"

It the performs: **GET_3BYTE_ADDR** which accepts six ASCII-Hex digits, from any selected input stream, to form a 24-bit address.

The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address, to each of the selected output streams. "AAAA" is the offset address within the bank.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **Get_S_Address** subroutine.

VECTOR:

$00:E048 Get_S_Adress

EXPECTS:

No input arguments.

RETURNS:

No output arguments.

The 3-byte result is returned in the global variables: **TMP2** ($00:0063), **TMP2+1**, and **TMP2+2**. The bytes are ordered such that: **TMP2**=LSB and **TMP2+2**=MSB.

The subroutine will return with the carry-bit = *clear* if a proper 6-digit address has been received.

ERRORS:

The carry-bit will be returned *set* if any non-digit character is detected before all six ASCII-Hex digits have been received.

GET_STR

DESCRIPTION:

This subroutine uses GET_PUT_CHR to receive characters and store them into a specified string buffer. The input string is terminated when an **ENTER** or **ESC** (Escape) character is detected. The completed string is terminated with a **NUL** ($00) character.

Refer to the description of CONTROL_INPUT for details about configuring the input stream and selecting input sources.

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **GET_STR** subroutine.

VECTOR:

$00:E03F GET_STR

EXPECTS:

Long pointer to string buffer as follows:

Bank address of string in 8-bit register-A.

Pointer to string in 16-bit register-X.

RETURNS:

No arguments returned. Received data string will be in specified string buffer. If no inputs have been selected, this subroutine will return immediately with an empty string.

Normally returns with the carry-bit *clear* if the string was ended by the **ENTER** character, or no inputs were enabled.

ERRORS:

Exception returns with the carry-bit *set* if an **ESC** (Escape) character was detected. No other errors are detected or reported.

GETDFREE

DESCRIPTION:

This subroutine computes the amount of free space available on a IC card using the DOS-compatible file structure.

VECTOR:

$00:80BF GETDFREE

EXPECTS:

Card specifier in 8-bit register-A:

$00 = Low card

$01 = High card

RETURNS:

Size of free space in bytes as 32-bit value, wherein:

Register-X = Most significant word size.

Register-Y = Least significant word size.

ERRORS:

No error conditions reported.

HEXIN

DESCRIPTION:

This subroutine will convert an ASCII HEX[18] character in register-A into 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 **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **HEXIN** subroutine.

Example:

```
LDA #'d'
JSL HEXIN
BCS Nothex3  ;Should never  branch
CMP #$0D
BNE Failed  ;Should never branch
…
LDA #'q'
JSL HEXIN
BCS NOTHEX3  ;Should always branch
….
```

VECTOR:

$00:E093 HEXIN

EXPECTS:

The ASCII hexadecimal character in register-A corresponds to the significant nibble of the result.

RETURNS:

The resulting binary value will be returned in register-A.

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if the parameter was not an ASCII HEX digit. (Note: The contents of register-A may be modified even if the conversion is not performed.)

---

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

IFASC

DESCRIPTION:

This subroutine will check the parameter byte in register-A. If the byte is a valid ASCII character[19] it will return with the carry-bit *clear*. The carry-bit will be *set* upon return if the character was not ASCII. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **IFASC** subroutine.

Example:

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

VECTOR:

$00:E097 IFASC

EXPECTS:

Input parameter byte in 8-bit register-A.

RETURNS:

The carry-bit will be *clear* of the data byte in register-A corresponds to a visible ASCII character.

If the parameter byte passed in register-A is not a valid ASCII character, the **IFASC** subroutine will return with the carry-bit *set* in the status register.

ERRORS:

No errors reported.

IS_CARD_INSERTED

DESCRIPTION:

This routine tests if a RAM card is installed in the specified PCMCIA slot.

VECTOR:

$00:808C IS_CARD_INSERTED

EXPECTS:

Register-A:	PCMCIA Slot Code:	$00 = Low Card Slot
		$01 = High Card Slot

RETURNS:

If RAM Card is installed:

Carry Bit = Clear

ERRORS:

Else, Carry Bit = Set

---

[19] This manual uses the term: "ASCII" when referring to *visible* 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.

ISDECIMAL

DESCRIPTION:

This subroutine will check the parameter byte in register-A. If the byte is a valid ASCII decimal digit ($30-$39), 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 **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **ISDECIMAL** subroutine.

Example:

```
LDA #$FA
JSL ISDECIMAL
BCC DEC_YES  ;Should never  branch
…
LDA #’7’
JSL ISDECIMAL
BCS NOT_DEC  ;Should always branch
….
```

VECTOR:

$00:E09B ISDECIMAL

EXPECTS:

Input parameter byte in 8-bit register-A.

RETURNS:

The carry-bit will be *clear* if the data byte register-A corresponds to an ASCII decimal character.

If the parameter byte passed in register-A is not valid ASCII decimal character, the **ISDECIMAL** subroutine will return with the carry-bit *set* in the status register.

ERRORS:

No errors reported.

ISHEX

DESCRIPTION:

This subroutine will check the parameter byte in register-A. If the byte is a valid ASCII Hexadecimal digit ($30-$39, $41-$46, and $61-$66), 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 Hexadecimal digit. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **ISHEX** subroutine.

Example:

```
LDA #$FA
JSL ISHEX
BCC HEX_YES  ;Should never  branch
…
LDA #'C'
JSL ISHEX
BCS NOT_HEX  ;Should always branch
….
```

VECTOR:

$00:E09F ISHEX

EXPECTS:

Input parameter byte in 8-bit register-A.

RETURNS:

The carry-bit will be *clear* if the data byte in register-A corresponds to an ASCII Hexadecimal character. ASCII Hexadecimal characters in the range: $61-$66 will be converted to upper case. Therefore, an input parameter of: $63 will be returned as: $43 in register-A.

If the parameter byte passed in register-A is not a valid ASCII Hexadecimal character, the **ISHEX** subroutine will return with the carry-bit *set* in the status register.

ERRORS:

No errors reported.

LOG_DRIVE

DESCRIPTION:

This routine switches the current drive to the card in the specified PCMCIA slot.

VECTOR:

$00:807D LOG_DRIVE

EXPECTS:

Register-A: PCMCIA Slot code: $00 = Low Card Slot
$01 = High Card Slot

RETURNS:

If RAM Card Logged to properly:
Carry Bit = Clear

ERRORS:

Else, Carry Bit = Set
Register-A = Error Code
$01 = No Card Found in the specified slot
$02 = Boot Sector on specified card is Invalid
$13 = Invalid File Access Table (FAT)

MENU_POINT

DESCRIPTION:

This subroutine does the following:

1. Asks for a keyboard character.
2. Keeps the time and date current on the LCD header.
3. Moves the cursor up & down in first column as arrow keys are pressed.
4. Returns to the calling program with carry-bit *clear* and ASCII code for ‘1’ through ‘8’ corresponding to the current menu line when the **ENTER** key is pressed.
5. Returns to the calling program with carry-bit *clear* and ASCII code for ‘1’ through ‘8’ if such a key is pressed.
6. Returns to the calling program with carry-bit *set* if the **ESC** (Escape) key is pressed.

VECTOR:

$00:80E9 MENU_POINT

EXPECTS:

No input arguments.

RETURNS:

Normally returns with carry-bit *clear*, indicating that a menu selection has been made. The selection code will be returned in the 8-bit register-A. The possible codes are:

$31 = Selection ‘1’ Other responses are ignored.
$32 = Selection ‘2’
$33 = Selection ‘3’
$34 = Selection ‘4’
$35 = Selection ‘5’
$36 = Selection ‘6’
$37 = Selection ‘7’
$38 = Selection ‘8’

ERRORS:

Exception returns with carry-bit *set*, indicating that the **ESC** key was pressed. No other exceptions or errors are detected or reported.

MENU_SETUP

DESCRIPTION:

This subroutine will:

1. Clear the LCD screen
2. Write the “MENSCH COMPUTER” header.
3. Write a specified menu string starting on line #3 of the LCD screen.
4. Positions cursor on line #3, column #0.

VECTOR:

$00:80E6 MENU_SETUP

EXPECTS:

Long pointer to the menu string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set.

RETURNS:

No arguments returned.

ERRORS:

No errors detected or reported.

MODEM_ANSWER

DESCRIPTION:

This subroutine forces the external modem to go off-hook, and generate a carrier tone. It assumes that the attached modem is “Hayes-compatible” and will accept commonly used “AT” command sequences.

VECTOR:

$00:8119 MODEM_ANSWER

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

If the modem has been configured to automatically return result codes, a response may have been received.

Used the GET_MODEM_RESPONSE subroutine to check for a result code.

ERRORS:

Carry-bit (*Clear* = OK / *Set* = Command did not execute properly.)

MODEM_DIAL

DESCRIPTION:

This subroutine forces the external modem to go off-hook, wait for a dial tone, and then dial a telephone number. It assumes that the attached modem is "Hayes-compatible" which will accept commonly used "AT" command sequences.

VECTOR:

$00:80F8 MODEM_DIAL

EXPECTS:

Long pointer to the telephone number string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set. The string must be fewer than 65535 bytes long.

RETURNS:

No arguments returned.

If the modem has been configured to automatically return result codes, a response may have been received.

Use the GET_MODEM_RESPONSE subroutine to check for a result code.

ERRORS:

Carry-bit (*Clear* = OK / *Set* = Command did not execute properly.)

MODEM_HANG_UP

DESCRIPTION:

This subroutine forces the external modem to go on-hook, and hangup the telephone line. It assumes that the attached modem is "Hayes-compatible" and will accept commonly used "AT" command sequences.

VECTOR:

$00:80FB MODEM_HANG_UP

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

If the modem has been configured to automatically return result codes, a response may have been received.

Use the GET_MODEM_RESPONSE subroutine to check for a result code.

ERRORS:

Carry-bit (*Clear* = OK / *Set* = Command did not execute properly.)

MODEM_REDIAL

DESCRIPTION:

This subroutine forces the external modem to go off-hook, wait for a dial tone, and then redial the last telephone number stored in the modem's memory. It assumes that the attached modem is "Hayes-compatible" which will accept commonly used "AT" command sequences.

VECTOR:

$00:811C MODEM_REDIAL

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

If the modem has been configured to automatically return result codes, a response may have been received.

Use the GET_MODEM_RESPONSE subroutine to check for a result code.

ERRORS:

Carry-bit (*Clear* = OK / *Set* = Command did not execute properly.)

MOVE_BUFFER_TO_LCD

DESCRIPTION:

This subroutine will move a 600-character LCD image from BUFFER1+40 (doesn't move line 0). Lines 1 through 15 are moved. A null is automatically inserted into buffer position 641 to be used as a string terminator.

VECTOR:

$00:8101 MOVE_BUFFER_TO_LCD

EXPECTS:

No input arguments.

RETURNS:

No arguments returned. All other registers are saved upon entry and restored before returning.

The global variable: **BUFFER1** is located @ $00:2800 in the Mensch Computer OS configuration.

ERRORS:

No errors detected or reported.

MOVE_PAGE_TO_BUF

DESCRIPTION:

This subroutine will move a 600-character LCD image to BUFFER1+40 (doesn't move line 0). Lines 1 though 15 are moved. A null is automatically inserted into buffer position 641 to be used as a string terminator.

VECTOR:

$00:80FE MOVE_PAGE_TO_BUF

EXPECTS:

No input arguments.

RETURNS:

No arguments returned. All other registers are saved upon entry and restored before returning.

The global variable: **BUFFER1** is located @ $00:2800 in the Mensch Computer OS configuration.

ERRORS:

No errors detected or reported.

POSITION_PIXEL (@ Coordinates: H,V)

DESCRIPTION:

This subroutine will position the next graphics operation at the specified coordinates on the LCD screen. The upper left corner of the display is: H=0, V=0. The lower right corner corresponds to coordinates: H=239, V=127.

VECTOR:

$00:81B6 POSITION_PIXEL

EXPECTS:

Horizontal coordinate (0-239) in 16-bit register-X.

Vertical coordinate (0-127) in 8-bit register-Y.

RETURNS:

No arguments returned.

ERRORS:

The carry-bit normally will return *clear*, but it will be *set* if the specified coordinates were unacceptable. (Clear = OK / Set = Unacceptable)

POSITION_TEXT_CURSOR (@ Row & Column)

DESCRIPTION:
This subroutine will position the text cursor at the specified coordinates on the LCD screen.

VECTOR:
$00:802F POSITION_TEXT_CURSOR

EXPECTS:
Row coordinate (0-15) 8-bit in register-A.

Column coordinate (0-39) in 16-bit register-X.

RETURNS:
No arguments returned.

ERRORS:
The carry-bit normally will return *clear*, but it will be *set* if the specified coordinates were unacceptable. (Clear = OK / Set = Unacceptable)

PUT_CHR

DESCRIPTION:
This subroutine will output a character to each of the selected output streams.

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **PUT_CHR** subroutine.

VECTOR:
$00:E04B PUT_CHR

EXPECTS:
Character to be output in 8-bit register-A.

RETURNS:
All registers are saved upon entry and restored before returning.

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:
The carry-bit will be *set* if no output ports are enabled.

PUT_STR

DESCRIPTION:

This subroutine will output a string to each of the selected output streams.

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **PUT_STR** subroutine.

VECTOR:

$00:E04E PUT_STR

EXPECTS:

Long pointer to the string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set.

The maximum string input size is limited to 640 characters.

RETURNS:

No arguments returned. All registers are saved upon entry and restored before returning.

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if no output ports are enabled.

RD_LCD_STRNG

DESCRIPTION:
This subroutine will read a text string from the LCD memory.

VECTOR:
$00:80DA RD_LCD_STRNG

EXPECTS:
Long pointer to the string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

Character count to be read in 16-bit register-Y.

The data will be read from the LCD memory starting at the current text cursor position.

RETURNS:
The carry-bit will be *clear* if the operation was successfully performed.

The string will be read into the specified buffer and terminated with a **NUL** ($00) character at the [Register-Y + 1] location in the buffer.

ERRORS:
The carry-bit will be *set* if the current text cursor position plus the character count exceeds the screen size, generating an invalid cursor position.

NOTE:
The string buffer should be at least one byte larger than the longest string to be read into it.

READ_ALARM

DESCRIPTION:
This subroutine will read the current system alarm setting as a null-terminated text string into a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **READ_ALARM** subroutine.

VECTOR:
$00:E051 READ_ALARM

EXPECTS:
A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

RETURNS:
Null terminated time string in specified buffer.

The string format is: HH:MM:SS <*null*>

wherein: HH = Hours code (0-23 possible) in ASCII digits.

"00" = Midnight
"12" = Noon
"23" = 11 PM

MM = Minutes code (0-59 possible) in ASCII digits.

SS = Seconds code (0-59 possible) in ASCII digits.

ERRORS:
No meaningful errors.

READ_DATE

DESCRIPTION:

This subroutine will read the current system date as a null-terminated text string into a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **READ_DATE** subroutine.

VECTOR:

$00:E054 READ_DATE

EXPECTS:

A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

RETURNS:

Null terminated date string in specified buffer.

The string format is: MM-DD-YY <*null*>

wherein: MM = Month code (1-12 possible) in ASCII digits.

"01" = January
"02" = February
"03" = March
"04" = April
"05" = May
"06" = June
"07" = July
"08" = August
"09" = September
"10" = October
"11" = November
"12" = December

DD = Day of month code (1-31 possible) in ASCII digits.

YY = Year code (last two digits: "94" = 1994) in ASCII digits.

ERRORS:

No meaningful errors.

READ_TIME

DESCRIPTION:

This subroutine will read the current system time as a null-terminated text string into a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **READ_TIME** subroutine.

VECTOR:

$00:E057 READ_TIME

EXPECTS:

A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

RETURNS:

Null terminated time string in specified buffer.

The string format is: HH:MM:SS <*null*>

wherein: HH = Hours code (0-23 possible) in ASCII digits.

“00” = Midnight
“12” = Noon
“23” = 11 PM

MM = Minutes code (0-59 possible) in ASCII digits.

SS = Seconds code (0-59 possible) in ASCII digits.

ERRORS:

No meaningful errors.

REMOVE_DIRECTORY (Reserved!)

DESCRIPTION:

This vector currently invokes a non-functional “dummy” subroutine. It is reserved for subdirectory operations in future versions of the Mensch Operating System.

VECTOR:

$00:80CB REMOVE_DIRECTORY

EXPECTS:

Not Applicable.

RETURNS:

Not Applicable.

ERRORS:

Not Applicable.

RESET

DESCRIPTION:

This subroutine will invoke the master start-up vector in ROM, effectively resetting the entire W65C265 system. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the processing and internal operations associates with the **RESET** library vector.

NOTE:	This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:

$00:E084 RESET

EXPECTS:

No input arguments.

RETURNS:

This vector does not return.

ERRORS:

No errors reported.

RESET_ALARM

DESCRIPTION:

This subroutine will reset all alarm flags to a "don't care" condition, effectively canceling any alarm setup or active alarm.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **RESET_ALARM** subroutine.

VECTOR:

$00:E05A RESET_ALARM

EXPECTS:

No input arguments.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors are detected.

RETRIEVE_DISPLAY_STATUS

DESCRIPTION:
This subroutine will get the status byte for the LCD display.

VECTOR:
$00:8029 RETRIEVE_DISPLAY_STATUS

EXPECTS:
No input arguments.

RETURNS:
Display status byte in 8-bit register-A:

Bit #	Meaning
0-LSB	LCD Power: 0 = OFF 1 = ON
1	
2	
3	
4	
5	
6	
7-MSB	

ERRORS:
No meaningful errors.

NOTE:
If the controller has been turned: **OFF**, via the *CONTROL_DISPLAY_PORT* subroutine, then the returned status bits may be misleading.

SBREAK

DESCRIPTION:
This subroutine will call the "software break" routine in ROM. It will save and print the processor's registers, and transfer control to the Mensch ROM Monitor's command processor. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the processing and internal operations of the **SBREAK** subroutine.

NOTE: This is an entry vector, not a subroutine. **IT WILL NOT RETURN!**

VECTOR:
$00:E05D SBREAK

EXPECTS:
No input arguments.

RETURNS:
This vector does not return.

ERRORS:
No errors reported.

SELECTED_COMMON_BAUD_RATE (for all ports except modem)

DESCRIPTION:

This subroutine allows the program to reconfigure the common baud rate generator which drives the serial ports for the keyboard, printer, and PC link. Changing this baud rate will affect all three ports. If used incorrectly, it can disable the keyboard.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SELECT_COMMON_BAUD_RATE** subroutine.

VECTOR:

$00:E060 SELECT_COMMON_BAUD_RATE

EXPECTS:

Baud rate selection code in 8-bit register-A:

0 = 110 Baud
1 = 150 Baud
2 = 300 Baud
3 = 600 Baud
4 = 1200 Baud
5 = 1800 Baud
6 = 2400 Baud
7 = 4800 Baud
8 = 9600 Baud
9 = 14400 Baud
A = 19200 Baud
B = 38400 Baud
C = 57600 Baud
D = 115000 Baud

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Unacceptable selection code.)

SELECT_DISK

DESCRIPTION:

This routine selects a PCMCIA Slot to be the Default Drive.

VECTOR:

$00:80D1 SELECT_DISK

EXPECTS:

Register-A: Device code to be the Default Drive
$00 = Internal (No Card Selected)
$01 = Low Card
$02 = High Card

RETURNS:

No arguments returned.

ERRORS:

No errors returned.

SEND_BYTE_TO_PC (Send via PC link port)

DESCRIPTION:

This subroutine will queue one byte from 8-bit register-A to be sent to the serial PC link port. The carry-bit will be set if the serial PC link port cannot accept data, otherwise it will be clear upon return.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SEND_BYTE_TO_PC** subroutine.

VECTOR:

$00:E063 SEND_BYTE_TO_PC

EXPECTS:

Output byte in 8-bit register-A.

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Serial PC link port cannot accept data.)

SEND_CR

DESCRIPTION:

This subroutine will output a **CR** (Carriage-Return/Enter = $0D) character to each of the selected output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SEND_CR** subroutine.

VECTOR:

$00:E066 SEND_CR

EXPECTS:

No input arguments.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if no output ports are enabled.

SEND_HEX_OUT

DESCRIPTION:

This subroutine will output an 8-bit value as two ASCII HEX digits to each of the selected output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SEND_HEX_OUT** subroutine.

VECTOR:

$00:E06C SEND_HEX_OUT

EXPECTS:

Value to be output in 8-bit register-A.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if no output ports are enabled.

SEND_SPACE

DESCRIPTION:

This subroutine will output a **SPACE** ($20) character to each of the selected output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SEND_SPACE** subroutine.

VECTOR:

$00:E069 SEND_SPACE

EXPECTS:

No input arguments.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if no output ports are enabled.

SET_ALARM

DESCRIPTION:

This subroutine will set the system alarm time from a null-terminated text string in a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SET_ALARM** subroutine.

VECTOR:

$00:E06F SET_ALARM

EXPECTS:

A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

The string format is: HH:MM:SS <*null*>

wherein: HH = Hours code (0-23 possible) in ASCII digits.

"00" = Midnight
"12" = Noon
"23" = 11 PM

MM = Minutes code (0-59 possible) in ASCII digits.

SS = Seconds code (0-59 possible) in ASCII digits.

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Error detected)

NOTE:

This subroutine requires a fixed-size, fixed-format input string. Therefore, the <*null*> terminator character is acceptable, but not really necessary.

RELATED:

The "User Alarm Wakeup Subroutine Vector" (UALRMIRQ = $00:0134) allows application programs to associate a subroutine with the alarm timeout condition. This vector should be initialized prior to setting the alarm. When the alarm times out, all essential overhead will be handled by the Mensch ROM Monitor before transferring control to the vector. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the operation of the feature.

SET_Breakpoint

DESCRIPTION:

This is the subroutine invoked by typing the '**B**' command at the monitor prompt. The '**B**' monitor command allows the user to set a breakpoint at a specific location. **SET_Breakpoint** will request an address and accept input via the each of the selected output streams.

The user must enter six ASCII-Hex digits to form a 24-bit address. The input format is:

BB:AAAA

Wherein: "BB" is the bank address. This subroutine will echo a ':' after the 2-digit bank address. "AAAA" is the offset address within the bank.

After a valid address has been entered, **SET_Breakpoint** will store a **BRK** instruction ($00) at the target location. When program execution reaches that address, the **BRK** instruction will transfer control to the Mensch Monitor in ROM.

NOTE:	This subroutine would not normally be used by application software on the W65C265. It has been included in this vector table to support anticipated needs of the extended Mensch Computer Operating System in that specific configuration. Developers should consult **Mensch Monitor Assembly Listing** for specific details regarding internal operation of this subroutine in order to determine suitability for other applications and configurations.

VECTOR:

$00:E072 SET_Breakpoint

EXPECTS:

No input arguments. This is an interactive subroutine which requests 'parameters from the user as needed.

RETURNS:

The carry-bit will be *clear* if the operation was successfully performed.

ERRORS:

The carry-bit will be *set* if invalid address data was entered.

SET_DATE

DESCRIPTION:

This subroutine will set the current system date from a null-terminated text string provided in a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SET_DATE** subroutine.

VECTOR:

$00:E075 SET_DATE

EXPECTS:

A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

The string format is: MM-DD-YY <*null*> or MM/DD.YY <*null*>

wherein: MM = Month code (1-12 possible) in ASCII digits.

"01" = January
"02" = February
"03" = March
"04" = April
"05" = May
"06" = June
"07" = July
"08" = August
"09" = September
"10" = October
"11" = November
"12" = December

DD = Day of month code (1-31 possible) in ASCII digits.

YY = Year code (last two digits: "94" = 1994) in ASCII.

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Error detected)

NOTE:

This subroutine requires a fixed-size, fixed-format input string. Therefore, the <*null*> terminator character is acceptable, but not really necessary.

SET_TIME

DESCRIPTION:

This subroutine will set the system time from a null-terminated text string in a user-specified buffer.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **SET_TIME** subroutine.

VECTOR:

$00:E078 SET_TIME

EXPECTS:

A pointer in 16-bit register-X to a nine (9) character buffer, located in memory bank #0.

The string format is: HH:MM:SS <*null*>

wherein: HH = Hours code (0-23 possible) in ASCII digits.

"00" = Midnight
"12" = Noon
"23" = 11 PM

MM = Minutes code (0-59 possible) in ASCII digits.

SS = Seconds code (0-59 possible) in ASCII digits.

RETURNS:

No arguments returned.

ERRORS:

Carry-bit (Clear = OK / Set = Error detected)

NOTE:

This subroutine requires a fixed-size, fixed-format input string. Therefore, the <*null*> terminator character is acceptable, but not really necessary.

STRCMP

DESCRIPTION:

This routine will compare 2 strings and test if the same.

VECTOR:

$00:80C5 STRCMP

EXPECTS:

TMP_PTR: Address of String Number 1

Register-X: High Address of String Number 2

Register-A: Bank of String Number 2

RETURNS:

If Strings are Equal:
Carry Bit = Clear

ERRORS:

If Strings are NOT Equal:
Carry Bit = Set

No other meaningful errors detected or reported.

UPPER_CASE

DESCRIPTION:

This subroutine will convert 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 they byte is not a lower case ASCII character, it will return unchanged. Refer to the **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **UPPER_CASE** subroutine.

Example:

```
LDA #'b'
JSL UPPER_CASE
CMP #'B'
BNE FAILED  ;Should never  branch
…
```

VECTOR:

$00:E0A3 UPPER_CASE

EXPECTS:

Character to be converted in register-A.

RETURNS:

Converted upper-case character is returned in register-A. If the byte was not a lower case ASCII character, it will be returned unchanged.

ERRORS:

No errors reported.

VERSION

DESCRIPTION:

This subroutine will return info on the current ROM version. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **VERSION** subroutine.

VECTOR:

$00:E07B VERSION

EXPECTS:

No input arguments.

RETURNS:

ROM version information:

Register-X = Pointer to 4-character string representing the version.
(Example: "2.01")

Register-Y = Pointer to formatted ASCII string representing the last assembly date.
(Example: "SAT DEC 3 12:16:05 1994")

Register-A = 0 (No particular significance.)

ERRORS:

No errors reported.

WR_3_ADDRESS

DESCRIPTION:

This subroutine will write a 3-byte address to the selected outputs in ASCII-Hex characters. The 3-byte address to be sent must be loaded into global variable: **TMP0** ($00:005D).

Refer to the description of CONTROL_OUTPUT for details about configuring the output streams.

Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **WR_3_ADDRESS** subroutine.

VECTOR:

$00:E07E WR_3_ADDRESS

EXPECTS:

No input arguments in registers. The 3-byte address to be sent must be loaded into global variable: **TMP0** ($00:005D), **TMP0+1** and **TMP0+2**. The least significant byte (LSB) of the 3-byte address must reside in **TMP0** and the MSB must be in **TMP0+2**. The address to be sent must be loaded into TMP0.

RETURNS:

No arguments returned.

ERRORS:

No errors detected or reported.

WR_LCD_STRNG

DESCRIPTION:

This subroutine will write a string of data to the LCD screen at the current text cursor position. This routine does not process non-displayable codes such as **BS** (Backspace = $08) or **CR** ($0D).

VECTOR:

$00:80D7 WR_LCD_STRNG

EXPECTS:

Long pointer to the string as follows:

Bank address of string in 8-bit register-A.

Offset address of string in 16-bit register-X.

The string must be terminated with either (1) a null character, or (2) the most significant bit of the last character set.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

WRITE_LCD_CHARACTER (@ Text Cursor Position)

DESCRIPTION:

This subroutine will write one character to the current text cursor position in the text display, and then advance the cursor to the next position.

Programmers should note that this subroutine should not be used unless they know where the cursor is positioned. If the cursor coordinates are outside the display area, the string will not appear.

VECTOR:

$00:8035 WRITE_LCD_CHARACTER

EXPECTS:

Output byte in register-A.

RETURNS:

No arguments returned.

ERRORS:

The carry-bit normally will return *clear*, but will be *set* if this subroutine cannot write the character as expected.

WRITE_PIXEL (@ Graphics Cursor Position)

DESCRIPTION:

This subroutine will write one pixel at the current graphics cursor position in the graphics display. (Optional: ON or OFF)

VECTOR:

$00:81B9 WRITE_PIXEL

EXPECTS:

Z/NZ pixel code in 8-bit register-A. If zero, the pixel will be cleared. If non-zero, the pixel will be written.

RETURNS:

No arguments returned.

ERRORS:

No meaningful errors.

XS28IN

DESCRIPTION:

This subroutine will read S28 records from any selected input stream, and place them into memory. It will wait (i.e. *not return*) until input is provided.

An **ESC** (Escape) character from any selected input stream will cancel the load operation and cause the subroutine to return with the carry-bit = *set*. Other characters from multiple inputs may cause load errors. This subroutine will begin computing a checksum when the start of an S28 record is detected.

Each time a record is processed, this subroutine outputs a period ('.') and 4-digit record number to all of the selected output streams. A '?' is returned if the checksum does not agree. This is for user feedback only. After receiving the final record a carry-bit = *clear* will be returned if no errors had been encountered. Likewise, if errors occurred, a carry-bit = *set* will be returned.

This operation cycle will continue until an error occurs or the "S8" end record is received. Refer to **Mensch Monitor Assembly Listing** for specific details regarding the internal operation of the **XS28IN** subroutine.

VECTOR:

$00:E081 XS28IN

EXPECTS:

No input arguments. The initial load address for each block of data is the first part of each S28 record.

RETURNS:

The carry-bit normally will return *clear* if the load operation completes without incident. No register arguments are returned, and no registers have been saved.

ERRORS:

The carry-bit will be *set* if **ESC** (Escape) was detected or if checksum errors occurred. Each S28 record is loaded into memory as it is processed. A checksum can only reflect the integrity of the data. When a checksum error is detected, it means that the memory has already been loaded with contaminated data. This subroutine is used by the 'S' command. Refer to that description for comments regarding error detection.

WARNING:

The address fields of the S28 records are not filtered in any way. ALL POSSIBLE ADDRESSES ARE ACCEPTABLE! The user is responsible for assuring that records do not overwrite critical locations which may disrupt proper operation of the firmware.

Appendix C – Glossary

ADSI
Abbreviation for: Analog Display Services Interface. This is a new series of telephone standards from BELLCORE relating to menus and responses for *smart* telephones.

Checksum
Value computed summing the contents of a file or block of memory prior to storage or transfer. Sometimes the checksum may be manipulated (negated, inverted, modulus-256, ect.) for specific algorithms. This value is then stored or transferred with the data. The computation may be repeated later and compared to the previous value to verify the integrity of the data.

CLOCK (i.e. Fast Clock / Slow Clock / Default Clock)
The W65C265 may operate from either of two clock sources. These can be dynamically selected. The slower clock is typically used to support *low-power mode*.

DTMF
Abbreviation for: Dual-Tone Mutli-Frequency

Emulation Mode
The W65C816 and therefore the W65C265 has the capability of operating in a mode wherein it's registers and instruction set *emulate* those of the earlier W65C02 design. This means that programs written for the W65C02 may execute without modification under emulation mode on the W65C265.

EPROM
Abbreviation for: Erasable Programmable Read-Only Memory

Interrupt
A break in the normal sequence of instruction execution.

I/O Library
This is a group of subroutines located in the W65C265 internal ROM and external EPROM which is accessible to user application programs. These subroutines support the standard I/O operations available in the Mensch Computer configuration.

LCD
Abbreviation for: Liquid Crystal Display

PCMCIA
Abbreviation for: Personal Computer Memory Card Interface Association. This is an international organization of interested parties attempting to standardize the interface for plug-in IC memory cards.

NOTE: Then Mensch Computer configuration supports a subset of the PCMCIA Type I standard.

PWM (as input)
Abbreviation for: Pulse Width Measurement. This is a hardware feature supported by the W65C265 chip.

PWM (as output)
Abbreviation for: Pulse Width Modulation. This is a technique for manipulating binary output levels to control the duty cycle when used in analog applications. Examples include digital sound reproduction and DC motor speed control.

RAM

Abbreviation for: Random Access Memory

ROM

Abbreviation for: Read-Only Memory

Shell

A *shell* is usually a command interpreter intended to simplify user interaction with more complex subsystem. The shell on the Mensch, in the PCMCIA DOS-compatible file emulation, allows the user to initiate complicated operations via single keystrokes or simple keyword commands.

W65C02 (Microprocessor)

The W65C02 has 66 instructions, 180 operational codes, and 15 addressing modes with one 8-bit accumulator, two 8-bit index registers, and an 8-bit stack pointer (256 byte stack). The standard part has an 8-bit data bus and a 16-bit address bus (64K byte address space). The clock input is not divided internally; the memory bus runs at one clock cycle per memory cycle. The W65C02S is compatible with the NMOS 6502 microprocessor used in early Apple, Atari, Nintendo, and Commodore computers.

W65C134 (Microprocessor)

The W65C134S has a W65C02S core microprocessor which is a fully static version of the 65C02. This part can operate at very low clock frequencies for low power consumption. The W65C134S chip also includes: 192 bytes of static RAM, 4K bytes of mask ROM, seven 8-bit bi-directional I/O ports, and four 16-bit timers. The 56 I/O pins may be reconfigured in software to provide additional functions. These include: external memory bus (8-bit data, 16-bit address), hardware interrupts (1 NMI, 2 IRQ, 7 positive edge, 5 negative edge), eight chip select outputs (for external static memories and I/O), a UART, and a serial interface bus (SIB). In addition, the W65C134S has the following features: vector interrupt system (22 priority encoded interrupts), two clock inputs (software switchable between low frequency and high frequency), wide power supply range (1.8 volts to 6.0 volts), and a debug monitor on ROM.

W65C265 (Microprocessor)

The W65C265S has a W65C816S core microprocessor which is a fully static version of the W65C816S. This part can operate at very low clock frequencies for low power consumption. The W65C265S chip also includes: 576 bytes of static RAM, 8K bytes of mask ROM, eight 8-bit bi-directional I/O ports, and eight 16-bit timers. The 65I/O pins may be reconfigured in software to provide additional functions. These include: external memory bus (8-bit data, 24-bit non-multiplexed address), hardware interrupts (1 NMI or ABORT, 1 IRQ, 1 positive edge, 3 negative edges), eight chip select outputs (for external static memories and I/O), four UARTs, and a parallel interface bus (PIB). In addition, the W65C265S has the following features: vectored interrupt system (29 priority encoded interrupts), two clock inputs (software switchable between low frequency and high frequency), wide power supply range (2.8 volts to 6.0 volts), and a debug monitor in ROM.

W65C816 (Microprocessor)

The W65C816S has 91 instructions, 255 operational codes, and 24 addressing modes with one 16-bit accumulator, two 16-bit index registers, and a 16-bit stack pointer (64K byte stack). The standard part has an 8-bit data bus and a 24-bit address bus (16M byte address space). The clock input is not divided internally; the memory bus runs at one clock cycle per memory cycle. The W65C816 has a software switchable emulation mode which allows it to run all 6502 and 65C02 software without modification.

WDC

Abbreviation for: The Western Design Center, Inc. of Mesa, Arizona. The company was founded in 1978 by William D. Mensch, Jr., who remains president and CEO. WDC designs, licenses, and sells CMOS microprocessors and microcomputers.

Appendix D – Connector Pinouts

Appendix D.1 – Internal Battery Connector (Pinouts)

Connector Pin #	Signal Name	Comments
1	Unregulated Charging Voltage	
2	Ground	
3	Battery Positive Terminal	

Appendix D.2 – Controller Connector (Pinouts)

There are only nine pins available on the game controller connector. One is used to supply the unit with +5 volts, and another is ground. This leaves only seven pins for everything else.

Pin #	Port Pin Identifier
1	PB0
2	PB1
3	PB2
4	PB3
5	+5 Volts
6	PB4
7	PB5
8	Ground
9	PB6

The most significant bit of port (PB7) is used as an output to switch the supply voltage to the controller connector. There is a jumper (**JMP4**) which may be used to change this feature and allow the user to define the entire 8-bit port. (Refer to the Mensch Computer schematics for the exact location of this jumper.)

Switch encoding on the SEGA Controller may be interpreted from the following table:

PB6	PB5	PB4	PB3	PB2	PB1	PB0	Notes
Start	0	A	-	-	Down	Up	
C	1	B	Right	Left	Down	Up	

Appendix D.3 – Serial Connectors (Pinouts))

Connector Pin #	Signal Name	Meaning	Comments
1	GND	Ground	
2	TXD	Transmitted Data	
3	+5V	+5 volts	
4	RXD	Received Data	
5	DSR	Data Set Ready	
6	DTR	Data Terminal Ready	

W65C265 Port #	Mensch Usage	Panel Label	Comments
S0	Keyboard	KBD	
S1	Printer	PTR	
S2	Modem	MDM	
S3	PC	PC	

Appendix D.4 – Display Cable Connector (Pinouts)

Mensch Connector Pin #	Mensch Signal Name	Contrast Pot	T6963C Module Pin #	T6963C Module Signal Name	Comments
1,2,7,8,23,24	GND		1,18	GND	
3,4,9,21,22	+5V	P4	2,19	+5V	
		P3	3	V0	
5	SEL		5	SEL	
6	MOD		16	MOD	
10	DISPEN		15	DISPEN	
11	DISPRS		4	DISPRS	
12	DISPR/We		6	DISPR/We	
13	DO0		7	DO0	
14	DO1		8	DO1	
15	DO2		9	DO2	
16	DO3		10	DO3	
17	DO4		11	DO4	
18	DO5		12	DO5	
19	DO6		13	DO6	
20	DO7		14	DO7	
		P2	17	VEE	

Appendix D.5 – IC Card Connectors (Pinouts)

Connector Pin #	Signal Name	PCMCIA Usage	Comments
1	GND	GND	
2	D3	DATA 3	
3	D4	DATA 4	
4	D5	DATA 5	
5	D6	DATA 6	
6	D7	DATA 7	
7	HighICCard = CS6* LowICCard = CS5*	Card Enable 1-	Memory Mapping: HighICCard = $40:0000 LowICCard = $01:0000
8	A10	ADDR 10	
9	OE*	Output Enable -	
10	A11	ADDR 11	
11	A9	ADDR 9	
12	A8	ADDR 8	
13	A13	ADDR 13	
14	A14	ADDR 14	
15	WE*	Write Enable -	
16	(Not Used)	Ready/Busy +/-	*Note: PCMCIA signal is ignored.
17	+5V	Vcc	
18	+5V	Prog. Voltage 1	*Note: PCMCIA Prog. Voltage 1 is usually +12 volts.
19	A16	ADDR 16	
20	A15	ADDR 15	
21	A12	ADDR 12	
22	A7	ADDR 7	
23	A6	ADDR 6	
24	A5	ADDR 5	
25	A4	ADDR 4	
26	A3	ADDR 3	
27	A2	ADDR 2	
28	A1	ADDR 1	
29	A0	ADDR 0	
30	D0	DATA 0	
31	D1	DATA 1	
32	D2	DATA 2	
33	(Not Used)	Write Protect +	*Note: PCMCIA signal is not used.
34	GND	GND	
35	GND	GROUND	
36	HighICCard = P44R LowICCard = P42R	Card Detect 1 -	
37	(Not Used)	DATA 11	
38	(Not Used)	DATA 12	
39	(Not Used)	DATA 13	
40	(Not Used)	DATA 14	
41	(Not Used)	DATA 15	
42	+5V	Card Enable 2 -	
43	(Not Used)	REFRESH	* Note: PCMCIA signal is not used.
44	(Not Used)	RESERVED	
45	(Not Used)	RESERVED	
46	A17	ADDR 17	
47	A18	ADDR 18	

Connector Pin #	Signal Name	PCMCIA Usage	Comments
48	A19	ADDR 19	
49	A20	ADDR 20	
50	A21	ADDR 21	
51	+5V	Vcc	
52	+5V	Prog Voltage 2	*Note: PCMCIA Prog. Voltage 2 is usually +12 volts.
53	A22	ADDR 22	
54	A23	ADDR 23	
55	GND	ADDR 24	
56	GND	ADDR 25	
57	(Not Used)	RESERVED	
58	P47/RES	CARD RESET +	
59	(Not Used)	WAIT -	*Note: PCMCIA signal is not used.
60	(Not User)	RESERVED	
61	HighICCard = RS2* LowICCard = RS1*	REGISTER SEL -	
62	(Not Used)	BATT VOLT DET2	
63	(Not Used)	BATT VOLT DET1	
64	(Not Used)	DATA 8	
65	RW*	DATA 9	
66	(Not Used)	DATA 10	
67	HighICCard = P45R LowICCard = P43R	CARD DETECT 2 -	
68	GND	GROUND	

Appendix D.6 – Expansion Connector (Pinouts)

Connector Pin #	Signal Name	Comments and Notes
1	GND	
2	GND	
3	D3	
4	(Not Used)	
5	D4	
6	BA/DOD*	
7	D5	
8	RES*	
9	D6	
10	IRQ*	
11	D7	
12	NMI*/AEO	
13	CS7*	
14	FCLK*	
15	A10	
16	+5V	
17	OE*	
18	(Not Used)	
19	A11	
20	(Not Used)	
21	A9	
22	(Not Used)	
23	A8	
24	A17	
25	A13	
26	A18	
27	A14	
28	A19	
29	WE*	
30	A20	
31	(Not Used)	
32	A21	
33	+5V	
34	+5V	
35	+5V	
36	+5V	
37	A16	
38	A22	
39	A15	
40	A23	
41	A12	
42	P72/CS2*	
43	A7	
44	P71/CS1*	
45	A6	
46	P47/RES	
47	A5	
48	P48/RES*	
49	A4	

Connector Pin #	Signal Name	Comments and Notes
50	BE/RDY	
51	A3	
52	PA4	
53	A2	
54	PA5	
55	A1	
56	PA6	
57	A0	
58	PA7	
59	D0	
60	PH12	
61	D1	
62	RW*	
63	D2	
64	RUN/SYNC	
65	(Not Used)	
66	(Not Used)	
67	GND	
68	GND	

Appendix E – Keycode To ASCII Conversion Tables

Key	SCAN CODE	ASCII	SHIFT	CTRL	ALT	ALT + CTRL	FUNC
ESC	01	1B	--	--	--	--	--
F1	02	A5	B1	BD	C9	--	--
F2	03	A6	B2	BE	CA	--	--
F3	04	A7	B3	BF	CB	--	--
F4	05	A8	B4	C0	CC	--	--
F5	06	A9	B5	C1	CD	--	--
F6	07	AA	B6	C2	CE	--	--
F7	08	AB	B7	C3	CF	--	--
F8	09	AC	B8	C4	D0	--	--
F9	10	AD	B9	C5	D1	--	--
F10	11	AE	BA	C6	D2	--	--
F11	12	AF	BB	C7	D3	--	--
F12	13	B0	BC	C8	D4	--	--
NumLk	14	--	--	--	--	--	--
PrtSc/SysRq	15	F0	F1	--	--	--	--
Insert/ScrLock	16	9B	--	--	--	--	F2/FE
Delete	17	9C	--	--	--	FF	--
Pause/Break	18	F3	F3	F4	F3	--	--
`~ key	19	60	7E	--	--	--	--
1! key	20	31	21	--	--	--	--
2@ key	21	32	40	--	80	--	--
3# key	22	33	23	--	--	--	--
4$ key	23	34	24	--	--	--	--
5% key	24	35	25	--	--	--	--
6^ key	25	36	5E	--	--	--	--
7& key	26	37	26	--	--	--	--
8* key	27	38	2A	--	--	--	--
9(key	28	39	28	--	--	--	--
0) key	29	30	29	--	--	--	--
-_ key	30	2D	5F	--	--	--	--
=+ key	31	3D	2B	--	--	--	--
Backspace	32	08	08	08	08	08	--
Home	33	9D	D5	DD	E5	--	--
Tab	34	09	ED	--	--	--	--
qQ	35	71	51	11	91	--	--
wW	36	77	57	17	97	--	--
eE	37	65	45	05	85	--	--
rR	38	72	52	12	92	--	--
tT	39	74	54	14	94	--	--
yY	40	79	59	19	99	--	--
uU	41	75	55	15	95	--	--
iI	42	69	49	09	89	--	--
oO	43	6F	4F	0F	8F	--	--

Key	SCAN CODE	ASCII	SHIFT	CTRL	ALT	ALT + CTRL	FUNC
pP	44	70	50	10	90	--	--
[{	45	5B	7B	--	--	--	--
]}	46	5D	7D	--	--	--	--
\ |	47	5C	7C	--	--	--	--
PageUp	48	9F	D7	DF	E7	--	--
CapsLock	50	--	--	--	--	--	--
aA	51	61	41	01	81	--	--
sS	52	73	53	13	93	--	--
dD	53	64	44	04	84	--	--
fF	54	66	46	06	86	--	--
gG	55	67	47	07	87	--	--
hH	56	68	48	08	88	--	--
jJ	57	6A	4A	0A	8A	--	--
kK	58	6B	4B	0B	8B	--	--
lL	59	6C	4C	0C	8C	--	--
;:	60	3B	3A	--	--	--	--
‘”	61	27	22	--	--	--	--
Enter	63	0D	--	--	--	--	--
PageDn	64	A0	D8	E0	E8	--	--
Left Shift	66	--	--	--	--	--	--
zZ	68	7A	5A	1A	9A	--	--
xX	69	78	58	18	98	--	--
cC	70	63	43	03	83	--	--
vV	71	76	56	16	96	--	--
bB	72	62	42	02	82	--	--
nN	73	6E	4E	0E	8E	--	--
mM	74	6D	4D	0D	8D	--	--
,<	75	2C	3C	--	--	--	--
. >	76	2E	3E	--	--	--	--
/?	77	2F	3F	--	--	--	--
Right Shift	78	--	--	--	--	--	--
Up Arrow	79	A1	D9	E1	E9	--	--
End	80	9E	D6	DE	E6	--	--
Function	81	--	--	--	--	--	--
Left Ctrl	82	--	--	--	--	--	--
Left Alt	83	--	--	--	--	--	--
Space Bar	84	20	--	--	--	--	--
Right Alt	86	--	--	--	--	--	--
Right Ctrl	87	--	--	--	--	--	--
Left Arrow	88	A3	DB	E3	EB	--	--
Down Arrow	89	A2	DA	E2	EA	--	--
Right Arrow	90	A4	DC	E4	EC	--	--

CODE	KEY/KEYS
00	Not Used.
01	**CTRL** + **aA** key
02	**CTRL** + **bB** key
03	**CTRL** + **cC** key
04	**CTRL** + **dD** key
05	**CTRL** + **eE** key
06	**CTRL** + **fF** key
07	**CTRL** + **gG** key
08	**Ctrl** + **hH** key *or* **BACKSPACE** *key only or with any of:* **CTRL** or **SHIFT** or **ALT** modifiers
09	**CTRL** + **iI** key *or* **TAB** key
0A	**CTRL** + **jJ** key
0B	**CTRL** + **kK** key
0C	**CTRL** + **lL** key
0D	**CTRL** + **mM** key *or* **ENTER** key
0E	**CTRL** + **nN** key
0F	**CTRL** + **oO** key
10	**CTRL** + **pP** key
11	**CTRL** + **qQ** key
12	**CTRL** + **rR** key
13	**CTRL** + **sS** key
14	**CTRL** + **tT** key
15	**CTRL** + **uU** key
16	**CTRL** + **vV** key
17	**CTRL** + **wW** key
18	**CTRL** + **xX** key
19	**CTRL** + **yY** key
1A	**CTRL** + **zZ** key
1B	**ESC** key
1C, 1D, 1E, and 1F	Not Used.
20	**Space** bar
21	**SHIFT** + **1!** key
22	**SHIFT** + **‘ “** key
23	**SHIFT** + **3#** key
24	**SHIFT** + **4$** key
25	**SHIFT** + **5%** key
26	**SHIFT** + **7&** key
27	**‘ “** key
28	**SHIFT** + **9(** key
29	**SHIFT** + **0)** key
2A	**SHIFT** + **8*** key
2B	**SHIFT** + =+ key
2C	, < key
2D	- _ key
2E	. > key
2F	/ ? key

CODE	KEY/KEYS
30	**0)** key
31	**1!** key
32	**2@** key
33	**3#** key
34	**4$** key
35	**5%** key
36	**6^** key
37	**7&** key
38	**8*** key
39	**9(** key
3A	**SHIFT** + ;: key
3B	;: key
3C	**SHIFT** + ,< key
3D	=+ key
3E	**SHIFT** + . > key
3F	**SHIFT** + **. >** key
40	**SHIFT** + **2@** key
41	**SHIFT** + **aA** key
42	**SHIFT** + **bB** key
43	**SHIFT** + **cC** key
44	**SHIFT** + **dD** key
45	**SHIFT** + **eE** key
46	**SHIFT** + **fF** key
47	**SHIFT** + **gG** key
48	**SHIFT** + **hH** key
49	**SHIFT** + **iI** key
4A	**SHIFT** + **jJ** key
4B	**SHIFT** + **kK** key
4C	**SHIFT** + **lL** key
4D	**SHIFT** + **mM** key
4E	**SHIFT** + **nN** key
4F	**SHIFT** + **oO** key
50	**SHIFT** + **pP** key
51	**SHIFT** + **qQ** key
52	**SHIFT** + **rR** key
53	**SHIFT** + **sS** key
54	**SHIFT** + **tT** key
55	**SHIFT** + **uU** key
56	**SHIFT** + **vV** key
57	**SHIFT** + **wW** key
58	**SHIFT** + **xX** key
59	**SHIFT** + **yY** key
5A	**SHIFT** + **zZ** key
5B	**[{** key
5C	\ | key
5D	**]}** key
5E	**SHIFT** + **6^** key
5F	**SHIFT** + **- _** key
60	` ~ key
61	**aA** key

CODE	KEY/KEYS
62	**bB** key
63	**cC** key
64	**dD** key
65	**eE** key
66	**fF** key
67	**gG** key
68	**hH** key
69	**iI** key
6A	**jJ** key
6B	**kK** key
6C	**lL** key
6D	**mM** key
6E	**nN** key
6F	**oO** key
70	**pP** key
71	**qQ** key
72	**rR** key
73	**sS** key
74	**tT** key
75	**uU** key
76	**vV** key
77	**wW** key
78	**xX** key
79	**yY** key
7A	**zZ** key
7B	**SHIFT** + **[{** key
7C	**SHIFT** + \ | key
7D	**SHIFT** + **]}** key
7E	**SHIFT** + `~ key
7F	
80	**ALT** + **2@** key
81	**ALT** + **aA** key
82	**ALT** + **bB** key
83	**ALT** + **cC** key
84	**ALT** + **dD** key
85	**ALT** + **eE** key
86	**ALT** + **fF** key
87	**ALT** + **gG** key
88	**ALT** + **hH** key
89	**ALT** + **iI** key
8A	**ALT** + **jJ** key
8B	**ALT** + **kK** key
8C	**ALT** + **lL** key
8D	**ALT** + **mM** key
8E	**ALT** + **nN** key
8F	**ALT** + **oO** key
90	**ALT** + **pP** key
91	**ALT** + **qQ** key
92	**ALT** + **rR** key
93	**ALT** + **sS** key

CODE	KEY/KEYS
94	**ALT** + **tT** key
95	**ALT** + **uU** key
96	**ALT** + **vV** key
97	**ALT** + **wW** key
98	**ALT** + **xX** key
99	**ALT** + **yY** key
9A	**ALT** + **zZ** key
9B	**Insert/ScrLock** key
9C	**Delete** key
9D	**Home** key
9E	**End** key
9F	**PageUp** key
A0	**PageDn** key
A1	**Up Arrow** key
A2	**Down Arrow** key
A3	**Left Arrow** key
A4	**Right Arrow** key
A5	**F1** key
A6	**F2** key
A7	**F3** key
A8	**F4** key
A9	**F5** key
AA	**F6** key
AB	**F7** key
AC	**F8** key
AD	**F9** key
AE	**F10** key
AF	**F11** key
B0	**F12** key
B1	**SHIFT** + **F1** key
B2	**SHIFT** + **F2** key
B3	**SHIFT** + **F3** key
B4	**SHIFT** + **F4** key
B5	**SHIFT** + **F5** key
B6	**SHIFT** + **F6** key
B7	**SHIFT** + **F7** key
B8	**SHIFT** + **F8** key
B9	**SHIFT** + **F9** key
BA	**SHIFT** + **F10** key
BB	**SHIFT** + **F11** key
BC	**SHIFT** + **F12** key
BD	**CTRL** + **F1** key
BE	**CTRL** + **F2** key
BF	**CTRL** + **F3** key
C0	**CTRL** + **F4** key
C1	**CTRL** + **F5** key
C2	**CTRL** + **F6** key
C3	**CTRL** + **F7** key
C4	**CTRL** + **F8** key

CODE	KEY/KEYS
C5	**CTRL** + **F9** key
C6	**CTRL** + **F10** key
C7	**CTRL** + **F11** key
C8	**CTRL** + **F12** key
C9	**ALT** + **F1** key
CA	**ALT** + **F2** key
CB	**ALT** + **F3** key
CC	**ALT** + **F4** key
CD	**ALT** + **F5** key
CE	**ALT** + **F6** key
CF	**ALT** + **F7** key
D0	**ALT** + **F8** key
D1	**ALT** + **F9** key
D2	**ALT** + **F10** key
D3	**ALT** + **F11** key
D4	**ALT** + **F12** key
D5	**SHIFT** + **Home** key
D6	**SHIFT** + **End** key
D7	**SHIFT** + **PageUp** key
D8	**SHIFT** + **PageDn** key
D9	**SHIFT** + **Up Arrow** key
DA	**SHIFT** + **Down Arrow** key
DB	**SHIFT** + **Left Arrow** key
DC	**SHIFT** + **Right Arrow** key
DD	**CTRL** + **Home** key
DE	**CTRL** + **End** key
DF	**CTRL** + **PageUp** key
E0	**CTRL** + **PageDn** key
E1	**CTRL** + **Up Arrow** key
E2	**CTRL** + **Down Arrow** key
E3	**CTRL** + **Left Arrow** key
E4	**CTRL** + **Right Arrow** key
E5	**ALT** + **Home** key
E6	**ALT** + **End** key
E7	**ALT** + **PageUp** key
E8	**ALT** + **PageDn** key
E9	**ALT** + **Up Arrow** key
EA	**ALT** + **Down Arrow** key
EB	**ALT** + **Left Arrow** key
EC	**ALT** + **Right Arrow** key
ED	**SHIFT** + **Tab** key
EE &EF	Not Used.
F0	**PrtSc/SysRq** key
F1	**SHIFT** + **PrtSc/SysRq** key
F2	**Fn** + **Insert/ScrLock** key
F3	**Pause/Break** key *or* **SHIFT** + **Pause/Break** key *or* **ALT** + **Pause/Break** key
F4	**CTRL** + **Pause/Break** key

CODE	KEY/KEYS
F5	
F6	
F7	
F8	
F9	
FA	
FB	
FC	
FD	
FE	**Fn** + **Insert/ScrLock** key
FF	**ALT** + **CTRL** + **Delete** key

Appendix F – Bibliography / Recommended References

Additional information on WDC products may be obtained by contacting:

The Western Design Center, Inc.
2166 East Brown Road
Mesa, Arizona 85213 USA

TEL: (480) 962-4545
FAX: (480) 835-6442

Some useful materials from WDC are listed below:

Publication	Notes
Programming the 65816 Including the 6502, 65C02, and 65802 by David Eyes and Ron Lichty	Excellent programming reference and tutorial for 65816 & W65C265S.
W65C265S INFORMATION, SPECIFICATION, AND DATA SHEET	Very detailed hardware data on the W65C265S micro-controller chip.
MENSCH MONITOR ROM REFERENCE MANUAL	Describes the internal ROM monitor program of the W65C265S chip.
Mensch Computer User Guide	Non-technical user description of the Mensch Computer.

Information about the *Analog Display Services Interface (ADSI)* standards for telephones may be obtained from:

BELLCORE ADSI PROJECT OFFICE

TEL: (908) 758-2257

Detailed information on IC memory cards and applicable standards are available from:

Source	Publication	Notes
Personal Computer Memory Card International Association (PCMCIA) 1030 G East Duane Avenue Sunnyvale, CA 94086 TEL: (408) 720-0107 Fax: (408) 720-9416	PC Card Standard Specification 2.01 11/92 Socket Services Specification 2.00 11/92 Card Services Specification 2.00 11/92 PC Card ATA Mass Storage Specification 1.01 11/92 AIMS Specification 1.01 11/92 Recommended Extensions 1.00 11/92	
Sycard Technology Sunnyvale, CA 94086 TEL: (408) 247-0703	The PCMCIA Developer's Guide by Michael T. Mori	

Specific information on non-WDC products mentioned in this manual may be obtained by contacting the supplier directly. The following table identifies some of these sources:

Source	Publication	Notes
Citizen American Corporation 2450 Broadway Ave. Suite 600 Santa Monica, California 90404 TEL: (310) 453-0614 FAX: (310)453-2814	CITIZEN™ GSX-190 User's Manual	Describes the GSX-190 serial printer.
SEGA 3335 Arden Road Hayward, CA 94545	SEGA™ 6-Button Arcade Pad	Describes the SEGA controller.
DENSITRON CORPORATION 3425 W. Lomita Boulevard Torrance, CA 90505 TEL: (213) 530-3530 FAX: (213) 325-8958	Application Notes for the T6963C LCD Graphics Controller	

Appendix G – Detailed Memory Map

Address Range	Function
$00:0000-$00:01FF	**W65C265S INTERNAL RAM**
$00:0000-$00:00FF	W65C265S internal RAM. (Page #0)
$00:0100-$00:0138	RAM IRQ Vectors
$00:0139-$00:01FF	W65C265S internal RAM. (Page #1)
$00:0200-$00:7FFF	**EXTERNAL 32K RAM IN MENSCH COMPUTER**
$00:0140-$00:02FF	Mensch Computer Stack
$00:0300-$00:0380	Variables
$00:0381-$00:023FF	Used by MENSCHWORKS
$00:0400-$00:05FF	Graphics Variables & Buffering
$00:0600-$00:06FF	Available
$00:0700-$00:07FF	Keyboard (Input From) Buffer
$00:0780-$00:07BF	Keyboard (Output To) Buffer
$00:07C0-$00:07FF	Printer (Output From) Buffer
$00:0800-$00:0BFF	Modem Input Buffer
$00:0C00-$00:0FFF	PC Link Input Buffer
$00:1000-$00:17FF	Printer Output Buffer
$00:1800-$00:1FFF	Modem Output Buffer
$00:2000-$00:27FF	PC Link Output Buffer
$00:2800-$00:2A89	Screen Buffer #1 (Global variable: BUFFER1)
$00:2A8A-$00:2D14	Screen Buffer #2
$00:2D15-$00:2FFF	?
$00:3000-$00:4FFF	Application Buffers
$00:5000-$00:5CFF	OSSHELL BUFFERS
$00:5D00-$00:77FF	?
$00:7800-$00:7A8F	PCMCIA Variables & Buffers
$00:7A90-$00:7FFF	?
$00:8000-$00:FFFF	**TOTAL EXTERNAL EPROM IN MENSCH COMPUTER**
$00:8000-$00:DEFF	**USABLE EXTERNAL EPROM IN MENSCH COMPUTER**
$00:8000-$00:8004	“WDC” semaphore & startup ENTRY POINT in external EPROM memory for Firmware.
$00:8005-$00:DEFF	Mensch Operating System
$00:DF00-$00:DF07	Not Usable.
$00:DF08-$00:DF1F	External I/O (LCD)
$00:DF20-$00:DF27	Internal I/O
$00:DF28-$00:DF3F	Reserved.
$00:DF40-$00:DF49	Register Storage
$00:DF4A-$00:DF4F	Reserved.
$00:DF50-$00:DF6F	Int. Timers
$00:DF70-$00:DF77	UARTs
$00:DF78-$00:DF7F	Unused Parallel Port
$00:DF80-$00:DF8F	W65C265S internal SRAM (Reserved by Monitor)
$00:DFC0-$00:DFFF	External I/O
$00:E000-$00:FFFF	W65C265S Internal Mensch ROM Monitor firmware.
$00:FF00-$00:FFFF	Interrupt Vectors
$01:0000-$3F:FFFF	Low IC Card Memory
$40:0000-$BF:FFFF	High IC Card Memory
$C0:0000-$FF:FFFF	Available to custom applications via the expansion connector.

Appendix H – S28 Record Transfer Format

FEILD	# Bytes	DESCRIPTION
Prefix	1	All records in the S28 style begin with the letter: 'S' ($53).
Record Type	1	Two types: '2' ($32) = Data record '8' ($38) = EOF record
Record Length	2	Record length (Address, Data, and Checksum fields) formatted as two ASCII hexadecimal digits, MSB first/LSB last. The record length of an EOF record will always be: "08" ($30, $38), because the other fields are fixed.
Load Address	6	Load address (24-bit) formatted as six ASCII hexadecimal digits, MSB first/LSB last. The address of an EOF record will always be zero, "000000" ($30, $30, $30, $30, $30, $30).
Data	Variable (Typically: 32, 48, or 64 characters)	Actual data bytes formatted as two ASCII hexadecimal digits each. The data field of an EOF record will be empty.
Checksum	2	Modulo-256 sum of all characters in previous three fields complemented (1's complement) and formatted as two ASCII hexadecimal digits. The checksum field of an EOF record will always be: "77" ($37, $37).

Figure 106
S28 Memory Transfer Format

INDEX

_

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V

W

X