AN1723

Interfacing MC68HC05 Microcontrollers to the IBM AT Keyboard Interface

By Derrick B. Forte
Networking and Communications Operation
Austin, Texas

Introduction

Since the inception of the IBM PC platform, the keyboard has served as its primary input device and, along with the PC's keyboard interface, now serves as part of the PC architecture standard. However, in recent years, PC hardware engineers have designed peripheral devices that can be used in place of or in conjunction with the keyboard.

This application note discusses the hardware and software issues involved in designing applications based on Motorola's M68HC05 Family of microcontrollers that can interact with an IBM AT computer at its keyboard interface. It explores using the interface as a power supply and a low-speed serial data link between an MC68HC05-based application and an IBM AT-compatible host computer. The major focus is on applications that are capable of operating while the keyboard is connected to the host PC.

The topics covered in this note are:

- An overview of the PC's keyboard subsystem's operation
- An examination of the subsystem's hardware design through an explanation of the keyboard-to-keyboard interface connection and the signals and protocols used in their communications
- A discussion of the interface's programming model and a method for using the interface as a power supply and a communications link with the PC
- An example of a digital thermometer design that is powered and controlled by an IBM AT computer's keyboard interface

IBM Keyboard Subsystem Overview

During its lifetime, the IBM PC platform has been supported by three types of keyboards: the XT keyboard, the AT keyboard, and the Multifunction II keyboard. Early PC platforms, such as the IBM XT, were supported by the XT keyboard. These early keyboards are not compatible with later PC platforms such as the AT and PS/2. Since the XT platform is now obselete, the XT keyboard will not be discussed in detail here.

With the advent of the AT platform, a new type of keyboard, the AT keyboard, was developed to support it. Its design gave the host computer more control over the keyboard's operation than was previously available on the XT. The AT keyboard's design also is used for the PS/2 platform. The two keyboards, however, use different cable connectors and scan code sets.

The third type of keyboard, the Multifunction II (MF II), evolved from the AT keyboard. The MF II's enhanced feature set has made it the standard in most systems today. The MF II keyboard uses the same keyboard interface as the AT, but it has a number of enhancements such as status LEDs and 18 to 19 additional keys. The MF II keyboard is available in a 101-key U.S. version and a 102-key European version.

Despite differences in their design and feature sets, all IBM PC keyboard subsystems consist of two parts:

- The keyboard with its cable
- A keyboard interface that links the keyboard to a host computer

During normal operation, the keyboard continually scans its key matrix for a keyboard event, either the pressing or releasing of a key by the user. When an event occurs, the keyboard assigns a unique byte or sequence of bytes called scan codes to the keystroke. The keyboard then attempts to transmit the scan code(s) to the PC over its cable. The PC's keyboard interface receives each scan code and, after performing a parity check on the transmission, either requests a retransmission of the code from the keyboard, if an error occurred, or passes it on to the PC's microprocessor. If the microprocessor is occupied at the time that a scan code is generated, the keyboard interface will signal the keyboard that the processor is busy. The keyboard will then hold off the transmission of any more scan codes until the interface signals that the processor can handle them. While the processor is busy, additional scan codes that may be generated by user keystrokes are stored in an internal keyboard buffer.

IBM Keyboard Subsystem Design

As mentioned earlier, the keyboard subsystem can be divided into two subsystems: the keyboard and the keyboard interface or port.

The keyboard performs these functions:

- Acquires user keystrokes from its key matrix
- Encodes them into scan codes; consult Appendix F for the AT keyboard scan codes of common alphanumeric characters
- Transmits the codes through its cable to the keyboard interface on the host.

To implement these functions, the IBM keyboard's design has always been centered on a single-chip microcontroller (MCU). Keyboards that supported the IBM XT were designed around the Intel 8048 microcontroller.

AT and MF II keyboards, on the other hand, are designed around a variety of microcontrollers. The microcontroller scans the key matrix for a key being pressed or released by the user. On detecting a keyboard event, the MCU debounces the key and determines its position within the key matrix. The MCU then encodes the keystroke by assigning it a scan code from an internal table. Keyboard scan codes are not to be confused with ASCII codes or any other character code sets that may be used internally by the host computer. Scan codes are converted to internal host computer codes after being passed to the host computer's main processor by the keyboard interface. On receiving a valid scan code, the keyboard interface circuitry generates an interrupt to the host's processor. If the processor is able to service the interrupt, the host computer enters its keyboard interrupt routine, which resides in the host system's BIOS. It is in the keyboard interrupt routine that scan codes are mapped to the host's internal character set.

The keyboard interface, the second component in the keyboard subsystem, is the keyboard's link to the host computer. The keyboard interface serves five functions:

- Supplies power to the keyboard
- Transmits host commands to the keyboard
- Receives the keyboard's responses to host commands
- Receives scan codes from the keyboard
- Provides an interface to the host computer's system bus

The keyboard interface's design integrates all these functions into a single microcontroller that serves as the interface's controller. The first AT keyboard interfaces were designed around the Intel 8042 microcontroller. In newer ATs and PS/2s, the Intel 8741 and Intel 8742 are used.

The keyboard interface's design, illustrated in **Figure 1**, can be divided into two parts:

- Keyboard communication link
- PC system bus interface

The keyboard interface communication link not only transmits to and receives data from the keyboard, but it also checks incoming keyboard data for transmission errors and controls the flow of data from the keyboard to the host.

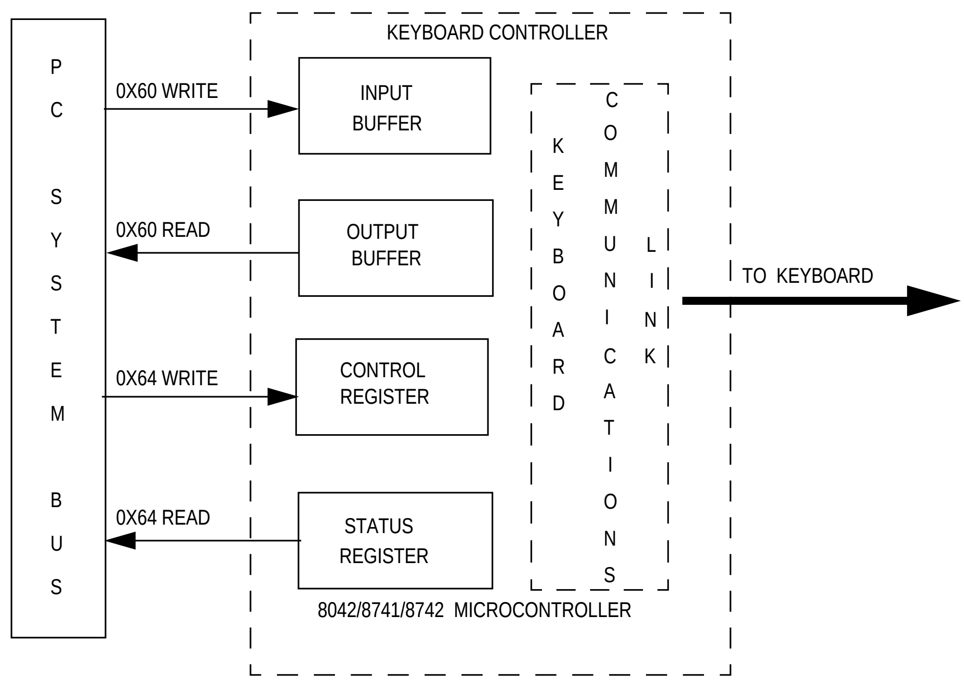


Figure 1. Keyboard Interface Design

The PC system bus interface is the point at which the PC's microprocessor interacts with the keyboard. The host configures and monitors the keyboard through the interface by sending keyboard commands directly to the keyboard or by writing keyboard controller commands to the interface's controller.

The keyboard interface consists of an input buffer, an output buffer, and the keyboard controller's control and status registers. The input and output buffers are mapped at address 0x60 in the PC's input/output (I/O) space. The input buffer is accessed on writes to address 0x60 while reads to address 0x60 access the output buffer. The host reads the keyboard's responses to host commands and scan codes from the output buffer. The keyboard controller's control and status registers are mapped at address 0x64 in the PC's I/O space. The keyboard status register is accessed on reads of address 0x64, while the control register is accessed on writes. The host issues commands to the keyboard controller by writing to the control register. For controller commands that require data in addition to the command byte, the host writes the required data to the input buffer. The host monitors the keyboard interface's transmission and reception of data by reading the keyboard controller status register.

The keyboard and the keyboard interface are physically connected through the keyboard's cable. This cable is a 5-wire shielded cable that has a male 5-pin or 6-pin circular DIN connector at one end. The other end of the cable is directly attached to the keyboard's internal circuitry. There are currently two types of keyboard connectors in use, the circular 5-pin DIN that is used with the AT platform and the 6-pin mini-DIN that is the PS/2's standard.

Figure 2 and Figure 3 show the pinouts of the two types of connectors.

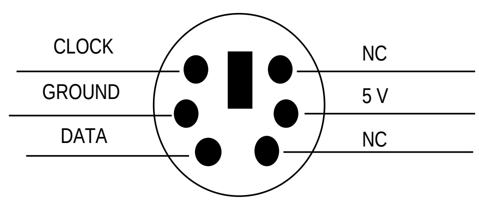


Figure 2. PS/2 Connector

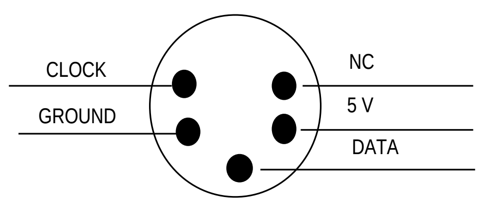


Figure 3. AT Connector

As shown in the figures, each connector has a 5-V pin and a ground pin. The keyboard interface powers the keyboard through these two pins. All keyboard interfaces are capable of supplying a keyboard with 5 V and a ground. The amount of power that the interface is capable of delivering can vary from one PC vendor to another. In addition, some PC motherboard designs fuse the interface's power signal to prevent a keyboard malfunction from affecting the host's power supply. The connector's shield ground pin along with a high-frequency filter in the connector limit the amount of EMI (electromagnetic interference) from the host that is permitted to travel along the keyboard's cable.

The keyboard and keyboard interface communicate over the connector's two remaining pins, clock, and data using a synchronous serial data link. The clock and data pins are bidirectional, open-collector signals that are pulled to 5 V by pullup resistors in the keyboard. This allows these lines to be pulled low by either the keyboard or the interface.

The first keyboards designed for the IBM PC/XT allowed only for the unidirectional transmission of scan codes from the keyboard to the host. The host exerted a minimal amount of control over the keyboard by means of a reset signal that was part of the keyboard interface. The enhanced features of the AT and MF II keyboards, however, required that the host exercise a greater measure of control over the configuration and operation of the keyboard. This led to a re-design of the keyboard, the keyboard interface, and the development of a protocol to govern the keyboard-to-host data link. The protocol defines a format and one set of timing specifications for the clock and data signals for keyboard-to-host data transfers and another for host-to-keyboard transfers.

In addition to these functions, the protocol also defines a set of commands that the host may send to the keyboard to monitor its status or change its configuration. The command set provides the host with commands to reset the keyboard, enable or disable the keyboard, and in the case of some keyboards change the keyboard's scan code set. (Consult the reference *PC Keyboard Design* for a complete list of the host-to-keyboard command set.)

The protocol also defines a set of codes that the keyboard should transmit back to the host after receiving a command from it. The protocol gives host computer-to-keyboard transfers priority over keyboard-to-host transfers. Therefore, if the keyboard is in the process of transmitting a scan code or a response to the host and the host wishes to send a command to the keyboard, the keyboard will relinquish control of the clock and data lines and allow its internal pullup resistors to pull them high. Then the host will transmit the command to the keyboard. After the keyboard has responded to the command, it will re-transmit the data whose transmission was interrupted. Keyboard-to-host and host-to-keyboard transfers share the same data format. The format consists of a start bit, eight data bits, one odd parity bit, and one stop bit. Also, in both protocols the keyboard generates the rising and falling edges of the clock signal. **Figure 4** illustrates the host-to-keyboard protocol, which is used by the host to send commands to the keyboard.

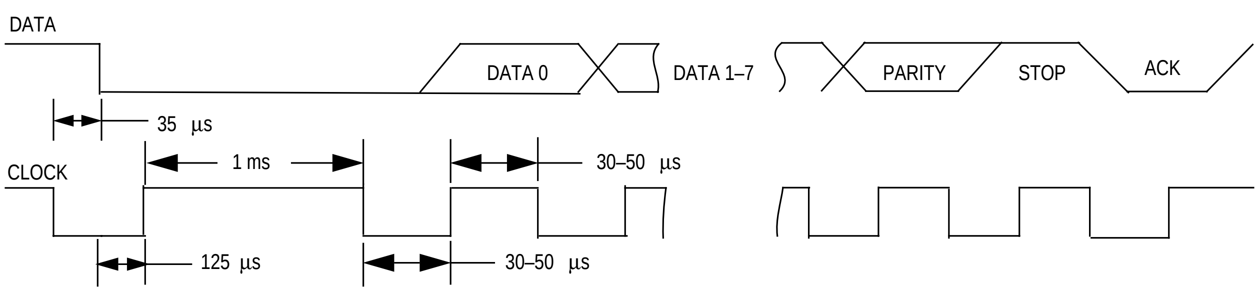


Figure 4. Host-to-Keyboard Data Transfer

The host-to-keyboard data transfer is accomplished by using these steps:

1. The host initiates a host-to-keyboard data transfer by pulling the clock line low. Approximately 35 microseconds later, the host pulls the data line low. This sequence of events signals the keyboard that the host is about to transfer a command. The clock signal is released and pulled high by the keyboard's pullup resistor approximately 125 microseconds after the falling edge of the data signal.

2. The transfer of data starts approximately 1 millisecond after the rising edge of the clock signal. During this time, the data line is held low. The transfer starts by the keyboard pulling the clock line low and clocking in the low data line. This serves as the transfer's start bit.
3. The keyboard then clocks in eight data bits from the host. The clock has a 50 percent duty cycle and has a high and low time of between 30 and 50 microseconds. The host changes the data during the low period of each cycle. Data from the host is sampled by the keyboard 5 to 25 microseconds after the rising edge of each clock.
4. The data bits are followed by a parity bit. The protocol uses odd parity.
5. The keyboard then clocks in a stop bit, ending the transfer.
6. If the keyboard reads a high stop bit, the keyboard pulls the data line low in the low period following the falling edge of the clock that is used to sample the stop bit. This serves as the keyboard's acknowledgement signal to the host. The keyboard pulls the data line high after pulling the clock high.
7. After receiving a byte, the keyboard performs a parity check on the received data. If a parity error is detected or the data received is not recognized as a valid command, the keyboard will request a retransmission of the byte by transmitting a $FE back to the host.

The keyboard-to-host protocol is used by the keyboard to send responses to host commands and scan codes to the host, as illustrated in **Figure 5**.

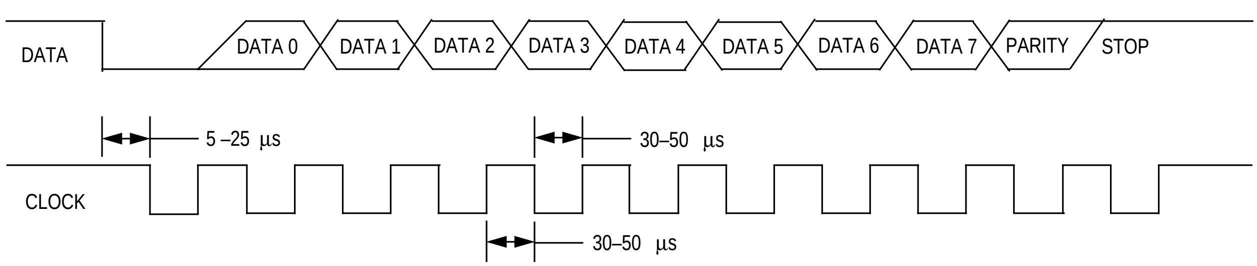


Figure 5. Keyboard-to-Host Transfer

The keyboard-to-host data transfer is accomplished by using these steps:

1. The keyboard initiates a keyboard-to-host data transfer by first allowing both the data and clock lines to be pulled high by its internal pullup resistors. The keyboard then pulls the data line low. Five to 25 microseconds later the keyboard pulls the clock line low. The falling edge of the clock line clocks in the transfer's start bit.
2. The keyboard then clocks in eight data bits to the host. The clock has a 50 percent duty cycle and high and low times of between 30 to 50 microseconds. The keyboard changes the data during the high period of each clock cycle. The change can occur between 5 microseconds after the rising edge of the clock and 5 microseconds before the falling edge. The keyboard's data is latched into the host by the falling edge of the clock.
3. The data bits are followed by an odd parity bit.
4. The keyboard then clocks in a stop bit ending the transfer. The host will pull the clock signal low from 0 to 50 microseconds after the falling clock edge that latches in the stop bit. This is a signal to the keyboard that the host is busy and is not capable of accepting another keyboard transfer. The host will release the clock line after it has processed the transfer and is ready to accept another transmission.
5. At any point during a keyboard-to-host transfer, the host can interrupt the transfer and transmit a command to the keyboard. The host signals that it wants to transmit a command by pulling the data line low while a high is being driven on the data line or pulling the clock line low during the high period of the clock. Therefore, the keyboard must sample the data line during the clock's low period whenever it outputs a high data bit. Since the keyboard is the master of the clock, the keyboard must also read the clock line whenever it outputs a rising edge on the clock line. If the data or clock signals are low under any one of these two conditions, the keyboard must relinquish control of the data and clock lines. It does this by allowing both lines to be pulled high. It then reverts to the host-to-keyboard transfer mode.

The Keyboard Interface Programming Model

The IBM personal computer architecture offers three ways to access the keyboard interface and through it, the keyboard:

- Operating system calls
- Keyboard access routines
- Reading and writing to the keyboard interface's input buffer, output buffer, and status and control registers

The first method involves the use of operating system calls, the highest level from which the keyboard can be accessed. The DOS operating system, for example, provides seven functions — 01h, 06h, 07h, 08h, 0Ah, 0Bh, and 3Fh — of DOS interrupt 21h for this purpose. Consult a good DOS reference for information on the calling parameters and return values for these functions. The disadvantage with using these functions is that they do not provide direct access to the keyboard or the keyboard interface.

At the level below the operating system calls are the keyboard access routines found in the BIOS. Among these are functions 4Fh and 85h of BIOS interrupt 15h, which are used by the keyboard hardware interrupt handler, which also resides in the BIOS, to process scan codes. In addition to the functions used by the keyboard interrupt handler, the BIOS interrupt 16h provides eight standalone keyboard access functions. Since they are close to the keyboard hardware, the functions provide better keyboard control than do the DOS functions. Consult a system BIOS reference guide for more information on these functions.

The third method is the one that perhaps the majority of hardware engineers are most comfortable with. It involves reading from and writing to the keyboard interface's input buffer, output buffer, and status and control registers to provide direct access to the keyboard and the keyboard interface. Commands can be issued directly to the keyboard by writing a command byte to the keyboard interface's input buffer. As mentioned earlier, this buffer is accessed by writing to address 0x60 in the host's I/O memory map. **Figure 6** illustrates both the keyboard interface's input and output buffers, which can be accessed by reading

I/O address 0x60. On completion of a write to address 0x60, the keyboard controller will take the data from the input buffer and transmit it to the keyboard using the host-to-keyboard serial protocol. The keyboard controller sets the input buffer status flag of the keyboard interface's status register when the transmission is complete. (See **Figure 6**.) If the command requires a response from the keyboard, the keyboard will transmit the appropriate response code(s) back to the host using the keyboard-to-host protocol. On receiving a response from the the keyboard, the keyboard interface places it into its output buffer and sets the output buffer status flag in the status register. The flag is also set on receiving a scan code from the keyboard. Therefore, by polling this flag, it can be determined when a byte has been received from the keyboard. The output buffer also is the location where scan codes are deposited when they are received from the keyboard.

D7	D6	D5	D4	D3	D2	D1	D0

Figure 6. Input and Output Buffer, Address 0x60

PARITY	TIMEOUT	AUXILIARY DEVICE	KEYBOARD LOCK	COMMAND DATA	SYSTEM FLAG	INPUT BUFFER STATUS	OUTPUT BUFFER STATUS

Figure 7. Keyboard Interface Status Register, Address 0x64

In addition to the input and output buffer status flags, the status register contains six error and status flags:

- Parity flag — Set if the last byte received from the keyboard or the mouse (PS/2 only) generated a parity error; it is clear otherwise
- Timeout flag — Set if a timeout occurs before the keyboard interface receives an expected response from the keyboard
- Auxiliary device flag — Set if the output buffer holds data from the mouse and it is cleared if the data is from the keyboard. This flag is relevant only in PS/2 models
- Keyboard local flag — Set if the keyboard is locked and clear if the keyboard is free

- Command/data flag — Set if a byte is written to the input buffer at address 0x60. The flag is cleared if the byte was written to the control reigster at address 0x64
- System flag — Set after the keyboard has passed its reset self test successfully

Using the Keyboard Interface as a Resource

The keyboard interface's design allows it to be used by embedded applications as a power supply and a low-speed bidirectional serial data link with the PC. The keyboard and an application can be easily supplied with power from the interface's 5-V pin.

Interfacing an application to the data link, on the other hand, requires addressing a number of issues.

The first of these involves the interface's clock and data pins. Since the keyboard interface's clock and data lines are open-drain signals, the possibility of contention exists on these two signals if both the keyboard and another device are capable of driving them at the same time. For example, if the host attempts to transmit data to a device other than the keyboard, the keyboard will see the activity on the clock and data lines and will try to respond to it. If the byte sent by the host is a valid keyboard command, the keyboard will attempt to respond to it with an appropriate response code. This could lead to a collision on the clock and data lines if another device connected to the interface attempts to transmit data at the same time. If the data sent by the host is interpreted by the keyboard as an invalid command, the keyboard will transmit a resend response code (0xFE) back to the host. This is another point at which contention could occur. Given these conditions, the keyboard's clock and data lines must be disconnected from those of the host whenever data is being transferred between another device and the host. Since neither the keyboard nor the keyboard interface are capable of disconnecting these two signals, this task must be performed by the other device. Since in most instances the keyboard will have priority over any other device connected to the interface, any device used in conjunction with the keyboard usually will operate as a pass-through device for the keyboard.

This will allow for normal keyboard operation until the device is activated by a signal sent by a program running on the host. This requires that the host send an activation signal that conforms to the data link protocol and that can be easily detected by other devices connected to the interface. Ideally, such a signal would have a minimal effect on the keyboard's present state and configuration.

The host-to-keyboard protocol's echo command is an ideal candidate for implementing such a signalling mechanism. The echo command (0xEE) is a part of the host-to-keyboard command set that can be used to test the integrity of the host-to-keyboard serial link. An echo command is initiated by the host sending an echo command (0xEE) to the keyboard through the keyboard interface. The keyboard responds by transmitting a 0xEE back to the host. This command and response sequence does not change the configuration of the keyboard in any way and thus fulfills the requirements for an activation signal. Since one echo command-response sequence may be sent in the normal course of host-to-keyboard transfers, a more distinctive signal must be devised to prevent the device from being inadvertently activated. Toward that end, the activation signal developed for this application consists of two echo command-response sequences sent between the keyboard and the host in rapid sucession. The fact that the activation signal can be sent by the host at any time requires that the receiving device constantly monitor the traffic on the data and clock lines. On detecting the signal, the device disconnects the keyboard's data and clock signals from those of the host and assumes sole possession of the keyboard's end of the data link. The keyboard's clock line must then be pulled low. While its clock line is low, the keyboard will store any scan codes that may be generated while it is disconnected from the host. Data can be transferred between the device and the keyboard interface at this point.

In addition to the issues involved with the clock and data signals, the host-to-keyboard interface protocol also imposes some restrictions on the data that can be exchanged between a device and the interface. If the host sends a byte that is a host-to-keyboard command, the host will expect an appropriate response code from the device. Any transmission from the device that is not the appropriate response will be regarded as an invalid response. Therefore, the program running on the host should only transmit data bytes that are not host-to-keyboard commands.

The protocol also restricts the bytes that can be sent from a device to the host. The recommended character set for device-to-host transmissions is the AT scan code set. By transmitting scan codes, the host will view the data as user keystrokes which can be parsed and processed by software running on the host.

Keyboard Thermometer System Design

The example application developed for this note is a digital thermometer that interfaces with an IBM AT-compatible host computer at its keyboard interface. The thermometer consists of two components:

- Keyboard thermometer device
- THERMO.EXE, a DOS application program that resides on the host computer.

The first component of the keyboard thermometer is the thermometer device itself. The thermometer has two connectors, one with which it interfaces to a host computer and the other with an AT keyboard. The thermometer is powered and controlled by the host at the interface. When deactivated, the thermometer serves as a passthrough device between the host and the keyboard, allowing normal keyboard operation to take place. While operating in this mode, the thermometer passively monitors the data traffic between the host and its keyboard for a predetermined activation sequence. On detecting an activation sequence, the thermometer disconnects the keyboard from the host and becomes the only device on the interface. The thermometer then takes a temperature reading, converts the reading into a series of scan codes, which the host will interpret as keystrokes, and transmits the codes to the host through its keyboard interface. After transmitting the scan codes, the thermometer re-connects the keyboard's clock and data signals to the host and the keyboard resumes normal operation. The thermometer then returns to monitoring the clock and data lines for an activation sequence.

The second component of the thermometer is THERMO.EXE, a DOS application program resident on the host computer. On being invoked, THERMO.EXE directs the thermometer device to take a temperature reading. If the attempt was successful, THERMO.EXE displays the data in a dialog box on the host computer's screen. From then on, THERMO.EXE waits for a keystroke from the user. If the user types in a "q" or "Q" character, the program exits to DOS. Otherwise, a keyboard thermometer activation sequence is sent through the host's keyboard interface. The activation sequence used for this application consists of two consecutive host-to-keyboard echo command-response sequences. On sucessfully completing the activation sequence, THERMO.EXE waits a maximum of two seconds for a response from the thermometer. The thermometer sends the reading as a string in the form of scan codes through the keyboard interface. The end of the string is delimited by the carriage return character. If the string is sucessfully received, THERMO.EXE displays it in a dialog box on the host's monitor. THERMO.EXE was compiled and linked with Borland's C++ compiler version 3.1. See **Appendix B. THERMO.EXE Flowchart** for a complete flowchart of THERMO.EXE 's design.

Keyboard Thermometer Hardware Design

The hardware design of the keyboard thermometer can be divided into two functional blocks:

- Temperature acquisition/conversion circuitry
- Keyboard interface circuitry

Each of these blocks is partially implemented by a Motorola MC68HC(7)05J1A microcontroller serving as the application's processor. Due to the limited amount of on-chip resources available on the MC68HC(7)05J1A, the Dallas Semiconductor DS1820 One-Wire Digital Thermometer was selected to implement the temperature acquisition and conversion block. The DS1820 integrates a temperature sensor, signal conditioning circuitry, and an A/D converter (analog-to-digital) into a 3-pin device. The device is capable of sensing its ambient temperature and converting the analog measurement into a 9-bit digital word every second. The 9-bit word is a representation of a temperature between –55 and +125 degrees Celcius in 0.5 degree Celcius increments. After the conversion process, the 9-bit word is stored, least significant byte first, in scratchpad RAM on the DS1820. A microcontroller can then read the word from the DS1820 using a serial protocol over the DS1820's DS pin.

Since the main focus of this note is a discussion of the keyboard interface circuitry, interfacing the DS1820 to the MC68HC(7)05J1A will not be examined in detail. For more information on interfacing the DS1820 to a 68HC05 MCU, consult *Adding a Voice User Interface to M68HC05 Applications*, Motorola order number AN1292/D.

The second functional block of the keyboard thermometer consists of circuitry that interfaces the application to the host PC's keyboard interface. In this application, this block is implemented with four of the MC68HC(7)05J1A's I/O pins, which are used to emulate the keyboard's clock and data signals. To comply with the keyboard-to-host transfer protocol, the AT keyboard must both drive and read the data and clock lines while transmitting data to the host. Therefore, the data and clock signals require two I/O pins each, one configured as an input and the other an output. The AT keyboard specification also calls for both the data and clock lines to be open-collector signals so that the host can interrupt a keyboard-to-host data transfer. Since the MC68HC(7)05J1A's I/O pins are actively driven when configured as outputs, they cannot be directly connected to a host's keyboard interface. Therefore, an open-collector buffer device along with an accompanying pullup resistor must be used as an interface between any one of the MC68HC(7)05J1A's I/O pins that is configured as an output and the host keyboard interface. The device selected to perform this function is a 7407 hex open-collector buffer. **Figure 8** illustrates a generic circuit that can be used to interface any member of the MC68HC05 Family of microcontrollers that does not have I/O pins with open-drain capabilities to an AT keyboard interface. Some members of the MC68HC05 Family, however, have I/O pins that can be configured as open-drain outputs. For devices with this feature, only a single pullup resistor is needed.

As explained earlier, the thermometer must disconnect the keyboard's clock and data signals from those of the host before transmitting data back to the host. If this is not done, the keyboard will detect the activity caused by the thermometer on the common data and clock lines and attempt to respond to it. This will create contention on both the data and clock lines.

Therefore, the keyboard's clock and data lines must be removed from those of the thermometer and the keyboard interface whenever the thermometer is transmitting. Since the the data line is bidirectional, a 4066 analog switch was selected to accomplish this task. After being disconnected from the common data and clock lines, the keyboard's clock and data lines are pulled up by its internal resistors.

For more information on the thermometer's hardware design, consult the schematic in **Appendix A. Keyboard Thermometer Schematics**.

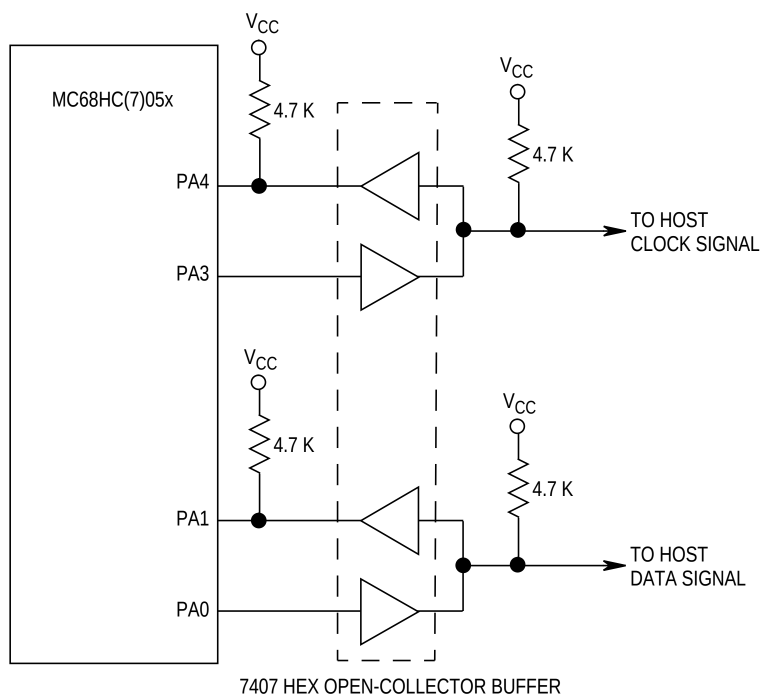


Figure 8. Generic MC68HC(7)05 to AT Keyboard Port Interface

Keyboard Thermometer Firmware Design

The keyboard thermometer's firmware has three main modules:

- Activation signal acquisition module
- Temperature acquisition and conversion module
- Keyboard interface module

The activation signal acquisition module includes routines that monitor the keyboard's clock and data lines for the activation sequence from the host. The main function within this module, **contact,** searches transactions between the host and the keyboard for two echo command-response sequences. Since the protocols for a host-to-keyboard transfer and a keyboard-to-host transfer are different, **contact** uses two routines, **read_command** and **read_response,** to detect a host-to-keyboard and keyboard-to-host transfer respectively.

As can be inferred from its name, the **read_command** routine monitors the keyboard lines for a valid host-to-keyboard transfer. As stated earlier, a host-to-keyboard transfer starts with the clock line being pulled followed by the data line being pulled low approximately 35 microseconds later. If the routine detects this sequence of events, a host-to-keyboard transfer is about to occur and the routine proceeds to read the command being sent to the keyboard. The routine reads a command by monitoring the rising and falling edges of the clock. The routine shifts in a bit on the data line 10 microseconds after detecting a rising edge on the clock line. Since the routine must wait for clock edges that are produced by the keyboard's clock signal, it is possible for the code to hang if an expected edge does not occur. To avoid this problem, good software design practice dictates that a software timeout loop be implemented for every instance where the routine waits on an edge. The routine checks a transfer for a start bit, eight data bits, a parity bit, a stop bit, and the keyboard's acknowledgement. Though all the elements of a transfer are checked, only the data and parity bits are stored. If a timeout error occurs or a parity error is detected, the routine's global error flag is set and exited.

The **read_response** routine is similar in implementation to the **read_command,** but is capable of detecting a keyboard-to-host transfer.

The second module handles all transactions with the DS1820 One-Wire Digital Thermometer. This module consists of all those functions that configure, and read data from the DS1820. Included among these are those functions that convert the 9-bit word received from the DS1820 into a sequence of scan codes for transmission to the host. A full discussion of these functions is found in *Adding a Voice User Interface to M68HC05 Applications*, Motorola order number AN1292/D.

The last module consists of those routines that allow the thermometer to transmit and receive data from the host's keyboard interface. After acquiring a temperature reading, the thermometer converts the 9-bit word read from the DS1820 into an array of scan codes to be transmitted to the PC through the keyboard interface. The transmission of the scan codes is interpreted as a series of user keystrokes by the host. To support the transmission of scan codes, the thermometer follows the timing specifications and protocol for keyboard-to-keyboard interface data transfers. This requires that the thermometer be capable of transmitting and receiving to and from the keyboard interface. Though the main function of this block is to transmit data to the PC, the module must be capable of receiving data from the host in the event that a parity error occurs during a keyboard-to-host transfer. So in addition to having a routine to transmit data to the host, the module also contains a routine to receive data from the keyboard interface. The transmission of data to the host is accomplished by toggling or "bit banging" two of the MC68HC(7)05J1A's I/O pins which have been configured as outputs, in accordance with the timing specifications for the data and clock lines. Data is read from the host by toggling the clock line and reading in the level of the data line 5 to 25 μs after each rising edge of the clock line. See **Appendix C. Keyboard Thermometer Firmware Flowchart** for a complete flowchart of the themometer's firmware design.

Keyboard Thermometer Operating Instructions

Follow these steps to operate the keyboard thermometer:

1. Copy THERMO.EXE to a directory on an IBM AT compatible host computer.
2. Disconnect the keyboard from the host computer.
3. Connect the keyboard connector to the appropriate connector on the thermometer.
4. Connect a keyboard extension cable between the keyboard interface of the host computer and the appropriate connector on the thermometer.
5. Start THERMO.EXE by typing **thermo** on the DOS commandline.
6. Follow the instructions given in the dialog box that is displayed.

Summary

The IBM AT platform's keyboard interface is a resource that can be used to power and control small MC68HC(7)05-based applications. By observing the constraints imposed by the PC keyboard's hardware design and keyboard-to-host and host-to-keyboard protocols, M68HC05-based applications can be developed that can operate in conjunction with the keyboard. A host computer can exert control over an application by using the host-to-keyboard data transfer protocol and the host-to-keyboard command set. The application can relay data back to the PC by sending scan codes which will be interpreted as user keystrokes. Programs resident on the host can then process the input as required.

Bibliography

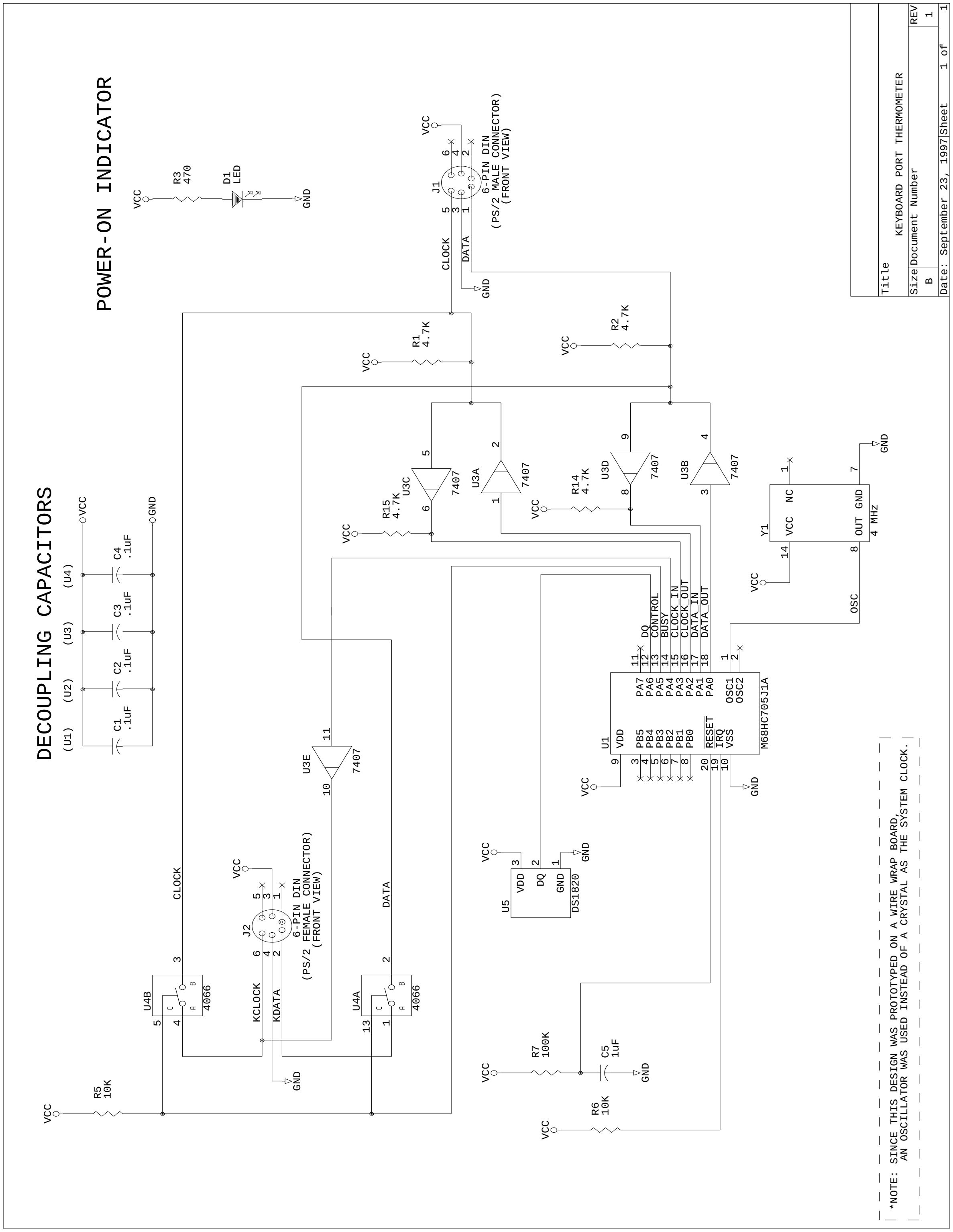
MC68HC705J1A Technical Data, Motorola order number MC68HC708J1A/D

Dallas Semiconductor DS1820 One-Wire Digital Thermometer Data Sheet

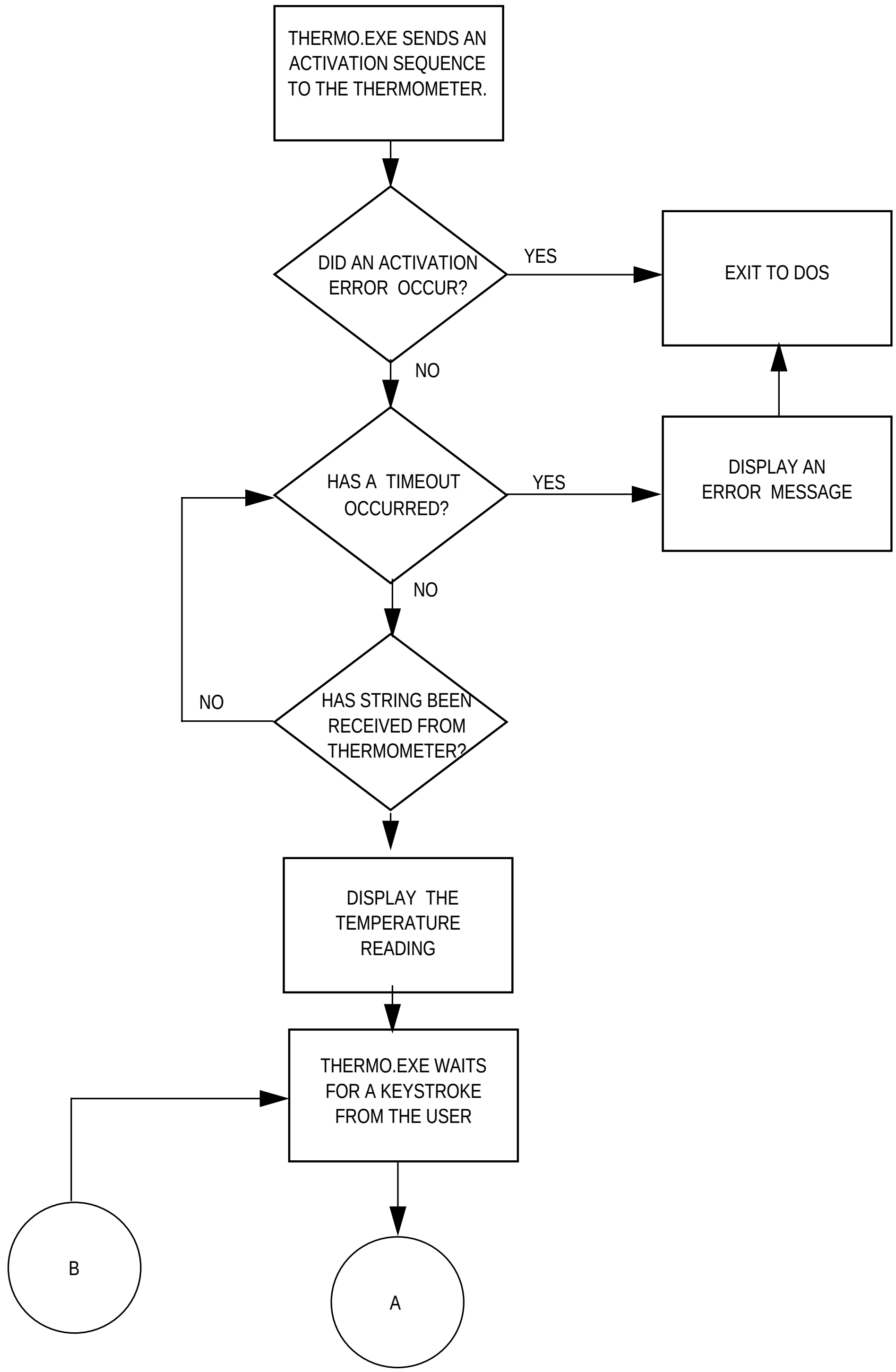
Konzak, Gary J.: *PC Keyboard Design,* 2nd. ed., Annabooks, San Diego, CA, 1993

Messmer, Hans-Peter: *The Indispensable PC Hardware Book – Your Questions Answered*; 1st. ed., Addison-Wesley Publishing Company, Reading, MA, 1994

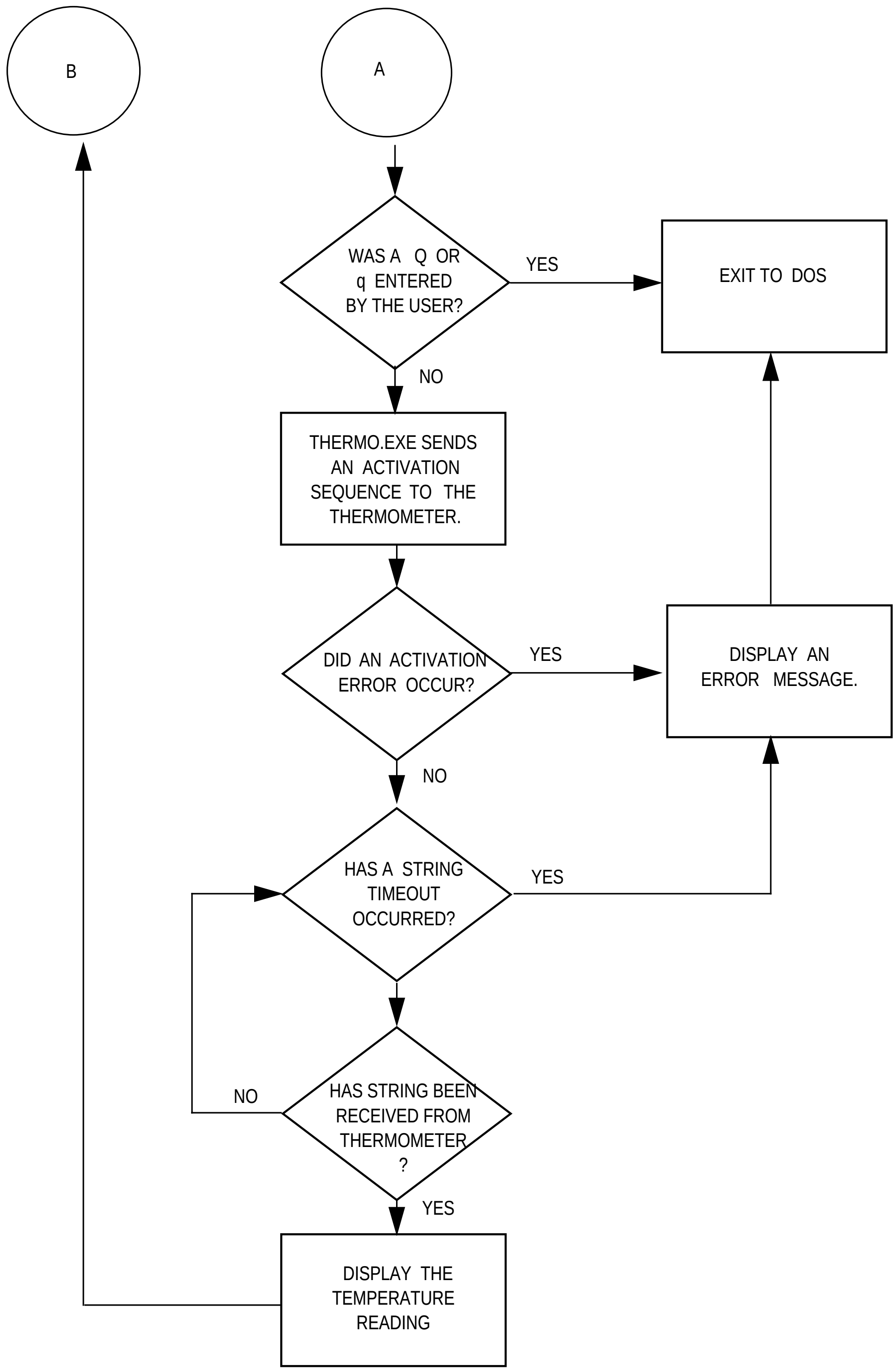
Appendix A. Keyboard Thermometer Schematics



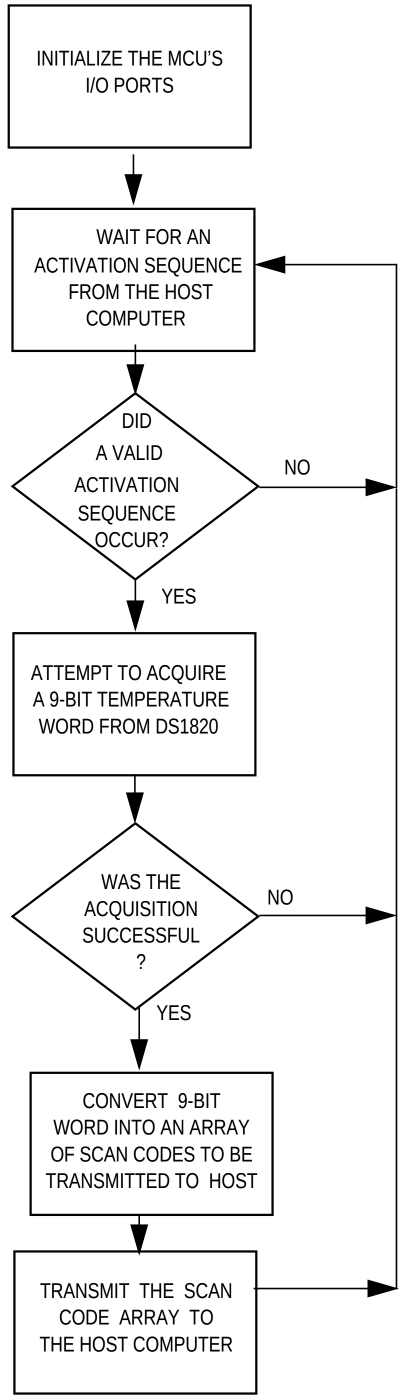
Appendix B. THERMO.EXE Flowchart



Appendix B. THERMO.EXE Flowchart (Continued)



Appendix C. Keyboard Thermometer Firmware Flowchart



Appendix D. Keyboard Thermometer Firmware Source Code

```
DATA            RMB   1            ;Storage space holding data that is transmitted or
                                   ;received
FLAG            RMB   1            ;Function return flag
TX_BUFFER       RMB   9            ;Data transmission buffer
TX_BUFFER_PTR   RMB   1            ;Transmission buffer pointer
TX_RESEND       RMB   1            ;Re-transmission storage space
TEMP            RMB   1            ;Temporary storage space
TEMPA           RMB   1            ;Temporary storage space for the A register
TEMPX           RMB   1            ;Temporary storage space for the X register
TEMP_HI         RMB   1            ;Temperature reading high byte
TEMP_LO         RMB   1            ;Temperature reading low byte
ODD_MULTIPLE    RMB   1            ;Flag indicating that a temperature reading that
                                   ;is an odd multiple of .5
QUOTIENT        RMB   1            ;Storage space for the result of division

PORTA           EQU   $00          ;PORT A data register
PORTB           EQU   $01          ;PORT B data register
DDRA            EQU   $04          ;PORT A data direction register
DDRB            EQU   $05          ;PORT B data direction register
TSCR            EQU   $08          ;Timer status/control register
COMMAND         EQU   DATA         ;Command byte read from the PC
RESPONSE        EQU   DATA         ;Response byte read from the keyboard
RX_BUFFER       EQU   DATA         ;Data receiver buffer
RAW_TEMP        EQU   TEMP_HI      ;Start of buffer holding an acquired
                                   ;temperature reading
ECHO            EQU   $EE          ;PC keyboard ECHO command
RESPONSE_BYTE   EQU   ECHO         ;Keyboard's response to an ECHO command
RESEND          EQU   $FE          ;PC keyboard resend command
CLOCK_OUT       EQU   2            ;Device keyboard clock output signal
CLOCK_IN        EQU   3            ;Device keyboard clock input signal
DATA_OUT        EQU   0            ;Device keyboard data output signal
DATA_IN         EQU   1            ;Device keyboard data input signal
BUSY            EQU   4            ;Keyboard busy
CONTROL         EQU   5            ;Keyboard enable/disable control signal
ONE_SECOND      EQU   $3D          ;One second RTI timeout value
RTIFR           EQU   2            ;Real-time interrupt flag mask
RTIF            EQU   6            ;Real-time interrupt flag mask
SIXTEENMS       EQU   1            ;16.4 mS timer delay mask
QUARTERSECOND   EQU   $0F          ;1/4 second timer delay mask
RX_PARITY       EQU   0            ;Parity bit in the FLAG variable
PARITY          EQU   7            ;Received parity bit in the FLAG variable
DQ              EQU   6            ;1820 data signal
DQ_CTRL         EQU   6            ;MCU 1820 data signal control pin
SKIPROM         EQU   $CC          ;1820 SKIP ROM COMMAND
CONVERT         EQU   $44          ;1820 temperature CONVERT command byte
READRAM         EQU   $BE          ;1820 READ RAM command byte
DDRAMASK        EQU   $F5          ;PORT A data direction register mask
```

```
DDRBMASK        EQU  $FF           ;PORT B data direction register mask
PORTAMASK       EQU  DDRAMASK      ;PORT A data mask
PORTBMASK       EQU  DDRBMASK      ;PORT B data mask
POSITIVE_SIGN   EQU  $00           ;MSB of a positive temperature reading
NEGATIVE_SIGN   EQU  $FF           ;MSB of a negative temperature reading
POSITIVE_LIMIT  EQU  $FA           ;The highest valid LSB for a positive
                                   ;temperature reading
NEGATIVE_LIMIT  EQU  $92           ;The lowest valid LSB for a negative
                                   ;temperature reading
ERROR           EQU  0             ;Error bit in return flag variable
MINUS           EQU  $4E           ;Scan code for the "-" character
ONE             EQU  $16           ;Scan code for the "1" character
POINT           EQU  $49           ;Scan code for the "." character
FIVE            EQU  $2E           ;Scan code for the "5" character
ZERO            EQU  $45           ;Scan code for the "0" character
END             EQU  $5A           ;Delimiter for the end of the TX table


                ORG  $300

START           BSR  INITIALIZE    ;Initialize MCU I/O ports.
WAIT_4_COMMAND  JSR  CONTACT       ;Wait for the PC to contact the device.
                JSR  ACQUIRE_TEMP  ;If contact is established with the PC
                TST  FLAG          ;acquire a temperature reading from the 1820,
                BNE  WAIT_4_COMMAND;convert it to a series of PC keyboard scan codes,
                JSR  FORMAT_TEMP   ;and send them to the PC.
                JSR  SEND_TEMP
                BRA  WAIT_4_COMMAND

*******************************************************************************
* Function Name: INITIALIZE                                                   *
* Function Inputs: None                                                       *
* Function Outputs: None                                                      *
*                                                                             *
* Purpose: This function initializes the MC68HC705J1A's I/O ports.            *
*                                                                             *
*******************************************************************************

INITIALIZE      LDA  #PORTAMASK    ;Set bits 1 & 3 of PORT A low
                STA  PORTA         ;Set all other bits high
                LDA  #DDRAMASK     ;Set bits 1 & 3 of PORT A as inputs
                STA  DDRA          ;Set all other bits as outputs
                LDA  #PORTBMASK    ;Set all PORT B bits high
                STA  PORTB
                LDA  #DDRBMASK     ;Set all PORT B bits as outputs
                STA  DDRB
                RTS
```

```
*******************************************************************************
*                                                                             *
* Function Name: CONTACT                                                      *
* Function Inputs: None                                                       *
* Function Outputs: None                                                      *
*                                                                             *
* Purpose: This function monitors the data traffic on the PC-to-keyboard data *
*          and clock signals for two ECHO command-response sequences. If the  *
*          sequences are found, this is interpreted as an activation signal   *
*          from the PC.                                                       *
*                                                                             *
*******************************************************************************

CONTACT        JSR   READ_COMMAND    ;Monitor the PC-to-keyboard traffic for a
                                     ;PC to keyboard command.
               TST   FLAG            ;Check the received byte for transmission
               BEQ   FIRST_ECHO      ;errors or for an $EE byte. If an error
               JSR   BYTE_DELAY      ;has occurred or an $EE was not received,
               BRA   CONTACT         ;delay one character time then branch back.
FIRST_ECHO     LDA   #ECHO           ;Otherwise continue
               CMPA  COMMAND
               BNE   CONTACT
               JSR   READ_RESPONSE   ;Read and check the response from
               TST   FLAG            ;the keyboard. If an error occurs
               BNE   CONTACT         ;or an $EE was not sent, search for
               LDA   RESPONSE        ;a new sequence.
               CMPA  #RESPONSE_BYTE
               BNE   CONTACT
               JSR   READ_COMMAND    ;Search for another keyboard
               TST   FLAG            ;ECHO command-response sequence.
               BNE   CONTACT         ;If one is sucessfully detected,
               LDA   #ECHO           ;continue. Otherwise branch back and
               CMPA  COMMAND         ;start searching for a new sequence.
               BNE   CONTACT
               JSR   READ_RESPONSE
               TST   FLAG
               BNE   CONTACT
               LDA   RESPONSE
               CMPA  #RESPONSE_BYTE
               BNE   CONTACT
               JSR   RESPONSE_DELAY  ;Allow time for the data and clock
               JSR   RESPONSE_DELAY  ;lines to be pulled high.
               RTS
```

```
*****************************************************************************
*                                                                           *
* Function Name: READ_COMMAND                                               *
* Function Inputs: None                                                     *
* Function Outputs: 0-If the data transmitted on the PC-keyboard data link is *
*                     a valid PC-to-keyboard command that does not have any   *
*                     transmission errors.                                  *
*                                                                           *
*                   1-If the data transmitted on the PC-keyboard data link is *
*                     an invalid PC keyboard command or if a transmission     *
*                     error occurred.                                       *
*                                                                           *
* Purpose: This function monitors the PC-to-keyboard data link signals for the*
*          transmission of a valid host-to-keyboard command. If the trans-    *
*          mitted data is not a command or if a transmission error occurred,  *
*          return a zero in the FLAG variable otherwise return a one.         *
*                                                                           *
*****************************************************************************

READ_COMMAND  CLR    FLAG                           ;Clear the return flag
                                                    ;variable.
              STA    TEMPA                          ;Store the accumulator
              STX    TEMPX                          ;Store the index register
              CLR    TEMP                           ;Clear temporary storage
                                                    ;space.
              CLR    DATA                           ;Clear the space that will
                                                    ;receive the data.
              CLC                                   ;Clear the carry bit
              LDX    #$9
WAIT4COMMAND  BRSET  DATA_IN,PORTA,WAIT4COMMAND     ;Wait for the falling edge
              BRSET  CLOCK_IN,PORTA,READ_CMD_ERROR;of the data line. If the
              LDA    #$48                           ;clock line is low continue.
WAIT4CLOCKHI  BRSET  CLOCK_IN,PORTA,STARTBITCLOCK ;Wait for a maximum of 504 µS
              DECA                                  ;for the clock line to
              BEQ    READ_CMD_ERROR                 ;rise.
              BRA    WAIT4CLOCKHI
STARTBITCLOCK LDA    #$D7                           ;Wait a maximum of 1.5 mS
WAIT4STARTING BRCLR  CLOCK_IN,PORTA,RISINGCLOCK     ;to clock in the start
              DECA                                  ;bit.
              BEQ    READ_CMD_ERROR
              BRA    WAIT4STARTING
FALLINGCLOCK  LDA    #$0A
WAIT4FALLING  BRCLR  CLOCK_IN,PORTA,RISINGCLOCK
              DECA
              BEQ    READ_CMD_ERROR
              BRA    WAIT4FALLING
RISINGCLOCK   LDA    #$0A                           ;Wait a maximum of 70 mS
WAIT4RISING   BRSET  CLOCK_IN,PORTA,GET_BIT         ;for the rising edge of
              DECA                                  ;the clock.
              BEQ    READ_CMD_ERROR
              BRA    WAIT4RISING
```

```
GET_BIT        LDA    #$3                           ;Wait 10US and read a data
GET_BIT_DELAY  DECA                                 ;bit from the data line.
               BNE    GET_BIT_DELAY
               BRCLR  DATA_IN,PORTA,GET_LOW_BIT     ;If the data bit is high
               CPX    #$1                           ;use the bit to calculate
               BEQ    SET_BIT                       ;the parity.
               INC    TEMP
SET_BIT        SEC                                  ;Set the carry bit
               BRA    STORE_BIT
GET_LOW_BIT    CLC                                  ;If the data is low clear
                                                    ;the carry bit.
STORE_BIT      ROR    DATA                          ;Roll the carry bit into
               DECX                                 ;the DATA variable.
               BNE    FALLINGCLOCK
               ROL    DATA                          ;Adjust the data and parity bits.
               BCS    PARITY_HI                     ;Check the parity.
               BRCLR  0,TEMP,READ_CMD_ERROR
               BRA    PARITYLO
PARITY_HI      BRSET  0,TEMP,READ_CMD_ERROR
PARITYLO       LDA    #$0A
WAIT4PARITYLO  BRCLR  CLOCK_IN,PORTA,STOPHI
               DECA
               BEQ    READ_CMD_ERROR
               BRA    WAIT4PARITYLO
STOPHI         LDA    #$0A
WAIT4STOPHI    BRSET  CLOCK_IN,PORTA,STOP_BIT
               DECA
               BEQ    READ_CMD_ERROR
               BRA    WAIT4STOPHI
STOP_BIT       LDA    #$2
STOP_BIT_DELAYDECA
               BNE    STOP_BIT_DELAY
               BRCLR  DATA_IN,PORTA,READ_CMD_ERROR ;Check for a stop bit.
                                                    ;If one is not found, exit
                                                    ;the function and signal an
                                                    ;error.
               LDA    #$0A
ACKNOWLEDGE_LOBRCLR   DATA_IN,PORTA,ACKNOWLEDGE_HI ;Check for an acknowledgement
                                                    ;from the keyboard. If one is
                                                    ;not found exit the function
                                                    ;and signal an error.
               DECA
               BEQ    READ_CMD_ERROR
               BRA    ACKNOWLEDGE_LO
ACKNOWLEDGE_HILDA     #$0E
HANDLE_ACK     BRSET  DATA_IN,PORTA,READ_CMD_EXIT
               DECA
               BEQ    READ_CMD_ERROR
               BRA    HANDLE_ACK
READ_CMD_ERRORINC     FLAG
READ_CMD_EXIT  LDA    TEMPA                         ;Restore the accumulator
               LDX    TEMPX                         ;Return the index register
               RTS                                  ;Return
```

```
*******************************************************************************
*                                                                             *
* Function Name: READ_RESPONSE                                                *
* Function Inputs: None                                                       *
* Function Outputs: 0 — If the data transmitted on the PC-keyboard data link is *
*                     a valid response to a host-to-keyboard command and has  *
*                     no transmission errors.                                 *
*                                                                             *
*                   1 — If the data transmitted on the PC-keyboard data link is *
*                     not a valid PC-to-keyboard command or if a transmission *
*                     error occurred.                                         *
*                                                                             *
* Purpose: This function monitors the PC-to-keyboard data link signals for a  *
*          response to the previously sent ECHO host-to-keyboard command. If  *
*          the transmitted data is not a command or if a transmission error   *
*          occurred, return a one in the FLAG variable, otherwise return a zero. *
*                                                                             *
*******************************************************************************

READ_RESPONSE CLR    FLAG                         ;Clear function return flag.
              STA    TEMPA                        ;Save the accumulator
              STX    TEMPX                        ;Save the index register
              CLR    TEMP                         ;Clear the temporary variable
              CLR    DATA                         ;Clear the DATA variable
              CLC                                 ;Clear the carry bit
                                                  ;Get the data and parity bits.
              LDX    #$09                         ;Initialize the index register
              LDA    #$90                         ;Wait for a maximum of 504 µS
                                                  ;for a response from the keyboard.
START_LOOP    BRCLR  DATA_IN,PORTA,CHECK_CLOCK    ;Wait for the falling the edge of
              DECA                                ;the data line. If the clock is
              BEQ    READ_ERR                     ;line is low, continue. If a
              BRA    START_LOOP                   ;response is not received,
                                                  ;exit the routine.
CHECK_CLOCK   BRCLR  CLOCK_IN,PORTA,READ_ERR
              LDA    #$0A
STARTING_EDGE BRCLR  CLOCK_IN,PORTA,RISINGEDGE
              DECA
              BEQ    READ_ERR
              BRA    STARTING_EDGE
RISINGEDGE    LDA    #$0A
RISING_EDGE   BRSET  CLOCK_IN,PORTA,FALLING_EDGE ;Wait for a maximum of 70 µS
              DECA                                ;for a rising edge of the clock.
              BEQ    READ_ERR
              BRA    RISING_EDGE
FALLING_EDGE  LDA    #$0A
WAIT4_FALLING BRCLR  CLOCK_IN,PORTA,GETBIT        ;Wait for a maximum of 70 µS
              DECA                                ;for a falling edge of the clock.
              BEQ    READ_ERR
              BRA    WAIT4_FALLING
```

```
GETBIT          BRCLR  DATA_IN,PORTA,GET_LO_BIT    ;If the data bit is low branch.
                CMPX   #$1                         ;Otherwise, use the bit
                BEQ    GET_HI_BIT                  ;to calculate the parity.
                INC    TEMP
GET_HI_BIT      SEC                                ;Set the carry bit
                BRA    STORE_DATA
GET_LO_BIT      CLC                                ;Clear the carry bit.
STORE_DATA      ROR    DATA                        ;Store the data bit.
                DECX
                BNE    RISINGEDGE
                ROL    DATA                        ;Adjust the data and parity bits.
                BCS    HI_PARITY_BIT               ;Check for a parity error, if one
                BRCLR  0,TEMP,READ_ERR             ;occurred, exit the function and set
                BRA    STOPBIT                     ;the function return FLAG variable.
HI_PARITY_BIT   BRSET  0,TEMP,READ_ERR
STOPBIT         LDA    #$20
WAIT_4_STOP_HIBRSET    CLOCK_IN,PORTA,STOP_LO_CLOCK;Wait for the stop bit to be
                                                   ;clocked in.
                DECA
                BEQ    READ_ERR
                BRA    WAIT_4_STOP_HI
STOP_LO_CLOCK   LDA    #$20
WAIT_4_STOP_LOBRCLR    CLOCK_IN,PORTA,CHECK_STOP_BIT
                DECA
                BEQ    READ_ERR
                BRA    WAIT_4_STOP_LO
CHECK_STOP_BITBRSET    DATA_IN,PORTA,RESPONSE_EXIT ;Check for the stop bit
READ_ERR        INC    FLAG
RESPONSE_EXIT   LDA    TEMPA                       ;Restore the accumulator
                LDX    TEMPX                       ;Restore the index register
                RTS                                ;Return
```

```
***********************************************************************************
*                                                                                 *
* Function Name: SEND_BYTE                                                        *
* Function Inputs: None                                                           *
* Function Outputs: 0 — If a data byte is successfully transmitted to the PC.     *
*                                                                                 *
*                   1 - If a data byte failed to be transmitted to the PC.        *
*                                                                                 *
* Purpose: This function transmits a data byte to the PC. The function            *
*          contact with the PC by transmitting a scan code to the PC. The         *
*          function then waits for a response from the PC. If the PC detects a    *
*          an error in the transmission, it will send a keyboard resend com-      *
*          mand (0xFE) back to the thermometer. On receiving a resend command,    *
*          the thermometer will re-transmit the data byte to the PC. If the       *
*          re-transmission fails, the function is exited and the function         *
*          return variable, FLAG, is set. If no transmission error occurs in      *
*          this function, the function is exited with the return flag cleared.    *
*                                                                                 *
***********************************************************************************

SEND_BYTE       BCLR   CONTROL,PORTA              ;Disconnect the keyboard from
                BCLR   BUSY,PORTA                 ;the PC.
                LDA    DATA                       ;Save the data to be transmitted
                STA    TX_RESEND                  ;in case a transmission error occurs.
                JSR    SEND                        ;Transmit the byte to the PC.
                BRCLR  ERROR,FLAG,EXIT_SEND_BYTE  ;If a transmission error did not
                JSR    ERROR_DELAY                ;occur, exit the function.
                JSR    RECEIVE                    ;Otherwise prepare to receive the
                BRCLR  ERROR,FLAG,CHECK_FOR_$FE   ;resend command (0xFE) from the PC.
                BRA    EXIT_SEND_BYTE
CHECK_FOR_$FE   LDA    #RESEND                    ;If a 0xFE is not received, set the
                CMP    RX_BUFFER                  ;return flag and exit the function.
                BEQ    RESEND_BYTE
                BRA    EXIT_SEND_BYTE             ;Otherwise resend the original data.
RESEND_BYTE     LDA    TX_RESEND                  ;If the re-transmission failed, set
                STA    DATA                       ;the function return flag and exit
                JSR    ERROR_DELAY                ;the function.
                JSR    SEND
                BRCLR  ERROR,FLAG,EXIT_SEND_BYTE
SEND_BYTE_ERROR BSET   0,FLAG                     ;If an error occurred, set the FLAG
                                                  ;variable to a non zero value.
EXIT_SEND_BYTE  BSET   BUSY,PORTA                 ;Reconnect the keyboard to the PC.
                BSET   CONTROL,PORTA
                RTS
```

```
*****************************************************************************
*                                                                           *
* Function Name: SEND                                                       *
* Function Inputs: None                                                     *
* Function Outputs: 0 - If a data byte is sucessfully transmitted to the PC.    *
*                                                                           *
*                   1 - If a data byte failed to be transmitted to the PC.     *
*                                                                           *
* Purpose: This function performs the low level I/O pin manipulations needed  *
*          to transmit a byte to the PC. This involves "bit banging" two I/O  *
*          pins to generate the clock and data signals. The function will    *
*          return a zero if the transmission was sucessful. A one will be    *
*          returned if an error occurred or if the PC wants to transmit a    *
*          a command while the data was being transmitted.                  *
*                                                                           *
*****************************************************************************

SEND        CLR     TEMP                        ;Clear space to calculate the
                                                ;parity.
            CLR     FLAG                        ;Clear the return flag.
            BSET    CLOCK_OUT,PORTA             ;Set the clock signal high.
            BSET    DATA_OUT,PORTA              ;Set the data signal high.
            LDX     #8
            BCLR    DATA_OUT,PORTA              ;Set up and clock in the start bit.
            JSR     HALF_CLOCK
            BCLR    CLOCK_OUT,PORTA
            JSR     FULL_CLOCK                  ;Clock in eight data bits.
            BSET    CLOCK_OUT,PORTA             ;If the PC pulls the clock line low,
            JSR     HALF_CLOCK                  ;while the I/O pin is driven high,
            BRCLR   CLOCK_IN,PORTA,SEND_ERROR   ;set the return flag and exit the
                                                ;function.
SEND_BIT    ROR     DATA
            BCS     SEND_ONE
            BCLR    DATA_OUT,PORTA
            BRA     SEND_DATA
SEND_ONE    BSET    DATA_OUT,PORTA              ;If the data bit being transmitted
            BRCLR   DATA_IN,PORTA,SEND_ERROR    ;is a one and the PC pulls it low,
            INC     TEMP                        ;set the return flag and exit
SEND_DATA   JSR     HALF_CLOCK                  ;the function.
            BCLR    CLOCK_OUT,PORTA
            JSR     FULL_CLOCK
            BSET    CLOCK_OUT,PORTA
            JSR     HALF_CLOCK
            BRCLR   CLOCK_IN,PORTA,SEND_ERROR
            DECX
            BNE     SEND_BIT
            ROR     TEMP                        ;Calculate the parity and send
            BCC     PARITY_ONE                  ;the parity bit.
            BCLR    DATA_OUT,PORTA
            BRA     SEND_PARITY
```

```
PARITY_ONE   BSET    DATA_OUT,PORTA
             BRCLR   DATA_IN,PORTA,SEND_ERROR
SEND_PARITY  JSR     HALF_CLOCK
             BCLR    CLOCK_OUT,PORTA
             JSR     FULL_CLOCK
             BSET    CLOCK_OUT,PORTA
             JSR     HALF_CLOCK
             BRCLR   CLOCK_IN,PORTA,SEND_ERROR
             BSET    DATA_OUT,PORTA
             BRCLR   DATA_IN,PORTA,SEND_ERROR
             JSR     HALF_CLOCK
             BCLR    CLOCK_OUT,PORTA
             JSR     FULL_CLOCK
             BSET    CLOCK_OUT,PORTA
             LDX     #2
PC_BUSY      BRCLR   CLOCK_IN,PORTA,STILL_BUSY  ;The PC will pull the clock
             JSR     FULL_CLOCK                 ;low while it processes the
             DECX                               ;transmitted data.
             BEQ     SEND_ERROR
             BRA     PC_BUSY
STILL_BUSY   LDX     #QUARTERSECOND             ;Wait a maximum of 1/4 second
             LDA     #SIXTEENMS                 ;for the PC to process the
             STA     TSCR                       ;transmitted data. If the PC
RST_TIMEOUT  BSET    RTIFR,TSCR                 ;does not release the clock
PC_TIMEOUT   BRSET   CLOCK_IN,PORTA,CHECK_DATA  ;line set the function return
             BRCLR   RTIF,TSCR,PC_TIMEOUT       ;flag and exit.
             DECX
             BNE     RST_TIMEOUT
             BRA     SEND_ERROR
CHECK_DATA   BRSET   DATA_IN,PORTA,SEND_EXIT    ;The PC will pull the data
SEND_ERROR   INC     FLAG                       ;low if a transmission error
SEND_EXIT    BSET    CLOCK_OUT,PORTA            ;set the return flag.
             BSET    DATA_OUT,PORTA             ;Reconnect the keyboard to the PC.
             RTS
```

```
***************************************************************************************
*                                                                                     *
* Function Name: RECEIVE                                                              *
* Function Inputs: None                                                               *
* Function Outputs: 0 - If a data byte was successfully received from the PC.         *
*                                                                                     *
*                   1 - If a data byte was unsuccessfully received from the PC.       *
*                                                                                     *
* Purpose: This function performs the low level I/O pin manipulations needed          *
*          to receive a data byte from the PC.                                        *
*                                                                                     *
***************************************************************************************
RECEIVE        CLR      DATA
               CLR      FLAG
               CLR      TEMP
               BSET     DATA_OUT,PORTA            ;Pull the clock and data lines
               BSET     CLOCK_OUT,PORTA           ;high.
               LDX      #$9
               BCLR     CLOCK_OUT,PORTA           ;Clock in the start bit.
               JSR      FULL_CLOCK
GET_BITS       BSET     CLOCK_OUT,PORTA
               JSR      HALF_CLOCK                ;Read in 8 data bits and the
               BRCLR    DATA_IN,PORTA,DATA_LO     ;parity bit.
               CPX      #$01
               BEQ      HIGH_BIT
               INC      TEMP
HIGH_BIT       SEC
               BRA      STORE
DATA_LO        CLC
STORE          ROR      DATA
               JSR      HALF_CLOCK
               BCLR     CLOCK_OUT,PORTA
               JSR      FULL_CLOCK
               DECX
               BNE      GET_BITS
               ROL      DATA
               BSET     CLOCK_OUT,PORTA
               BCC      CLR_PARITY
               BSET     PARITY,TEMP
               BRA      STOP
CLR_PARIT      BCLR     PARITY,TEMP
STOP           JSR      HALF_CLOCK                ;Clock in the stop bit.
               BRCLR    DATA_IN,PORTA,RCV_ERROR
               BCLR     DATA_OUT,PORTA
               JSR      HALF_CLOCK
               BCLR     CLOCK_OUT,PORTA
               JSR      FULL_CLOCK
               BRCLR    PARITY,TEMP,TST_PARITY    ;Check the parity of the
               BRSET    RX_PARITY,TEMP,RCV_ERROR  ;received data.
               BRA      RCV_EXIT
TST_PARITY     BRSET    RX_PARITY,TEMP,RCV_EXIT
RCV_ERROR      INC      FLAG
RCV_EXIT       BSET     CLOCK_OUT,PORTA           ;Reconnect the keyboard to
               BSET     DATA_OUT,PORTA            ;the PC.
               RTS
```

```
*************************************************************************************
*                                                                                   *
* Function Name: ACQUIRE_TEMP                                                       *
* Function Inputs: None                                                             *
* Function Outputs: 0 - If a temperature reading was sucessfully acquired from      *
*                       the 1820.                                                   *
*                                                                                   *
*                     1 - If a temperature reading was not acquired from the 1820.  *
*                                                                                   *
* Purpose: This function calls the sequence of low level routines that acquire      *
*          a temperature reading from the 1820. If the acquisition is               *
*          sucessful, the reading is returned in the TEMP_HI and TEMP_LO            *
*          variables and function return flag is cleared. If an error occurs        *
*          while acquiring a reading, the function return flag is set and the       *
*          function is exited.                                                      *
*                                                                                   *
*************************************************************************************

ACQUIRE_TEMP     JSR        RESET_1820           ;Reset the 1820. If the
                 TST        FLAG                 ;1820 did not reset, set
                 BNE        GET_ERROR            ;the function return flag
                                                 ;and exit the function.
                 LDA        #SKIPROM             ;Send the 1820 SKIP PROM
                 STA        TEMP                 ;command.
                 JSR        WRITE_1820
                 LDA        #CONVERT             ;Send the 1820 CONVERT T
                 STA        TEMP                 ;command.
                 JSR        WRITE_1820
READ_LOOP        JSR        READ_1820            ;Wait for the 1820 to
                 LDA        TEMP                 ;execute the CONVERT
                 CMP        #$FF                 ;command.
                 BNE        READ_LOOP
                 JSR        RESET_1820           ;Reset the 1820. If the
                 TST        FLAG                 ;1820 did not reset, set
                 BNE        GET_ERROR            ;the function return flag
                                                 ;and exit the function.
                 LDA        #SKIPROM             ;Send the 1820 SKIP PROM
                 STA        TEMP                 ;command.
                 JSR        WRITE_1820
                 LDA        #READRAM             ;Send the 1820 READ RAM
                 STA        TEMP                 ;command.
                 JSR        WRITE_1820
                 JSR        READ_1820            ;Read the temperature from the
                 LDA        TEMP                 ;1820.
                 STA        TEMP_LO
                 JSR        READ_1820
                 LDA        TEMP
                 STA        TEMP_HI
                 CMP        #POSITIVE_SIGN       ;Check for a positive
                 BEQ        CHECK_POSITIVE       ;temperature.
                 CMP        #NEGATIVE_SIGN       ;Check for a negative
                 BNE        GET_ERROR            ;temperature.
                 LDA        TEMP_LO
                 CMP        #NEGATIVE_LIMIT      ;Check a negative reading
                 BLO        GET_ERROR            ;to see if it is within
                 BRA        GET_EXIT             ;proper limits.
CHECK_POSITIVE   LDA        TEMP_LO              ;Check a positive reading
                 CMP        #POSITIVE_LIMIT      ;to see if it is within
                 BLS        GET_EXIT             ;proper limits.
GET_ERROR        INC        FLAG
GET_EXIT         JSR        RESET_1820           ;Reset the 1820.
                 RTS
```

```
************************************************************************************
*                                                                                  *
* Function Name: RESET_1820                                                        *
* Function Inputs: None                                                            *
* Function Outputs: 0 - If the 1820 resets                                         *
*                                                                                  *
*                   1 - If the 1820 fails to reset.                                *
*                                                                                  *
* Purpose: This function resets the 1820. After a reset the 1820 should send       *
*          back an acknowledgement. If an acknowledgement is not sent back set     *
*          the function return flag and exit the function. Otherwise return a      *
*          cleared function return flag.                                           *
*                                                                                  *
************************************************************************************

RESET_1820     STA      TEMPA                     ;Save the CPU registers
               STX      TEMPX
               CLR      FLAG                      ;Clear the function return flag.
               BSET     DQ,PORTA                  ;Send a reset pulse to the 1820.
               BSET     DQ_CTRL,DDRA
               BCLR     DQ,PORTA
               JSR      DELAY_500µS
               BSET     DQ,PORTA
               BCLR     DQ_CTRL,DDRA              ;Wait for a response from the
               JSR      DELAY_100uS               ;1820. If a response is not received
               BRSET    DQ,PORTA,RESET_ERR        ;set the function return flag and
               JSR      DELAY_500µS               ;exit the function.
               BRSET    DQ,PORTA,RESET_EXIT
RESET_ERR      INC      FLAG
RESET_EXIT     BSET     DQ,PORTA                  ;Set the J1A up for the next
               BSET     DQ_CTRL,DDRA              ;transmission.
               LDA      TEMPA                     ;Restore the CPU registers.
               LDX      TEMPX
               RTS
```

```
******************************************************************************
*                                                                            *
* Function Name: WRITE_1820                                                  *
* Function Inputs: None                                                      *
* Function Outputs: None                                                     *
*                                                                            *
* Purpose: This function writes the data stored in the TEMP variable to the  *
*          1820.                                                             *
*                                                                            *
******************************************************************************

WRITE_1820      STA     TEMPA           ;Save the CPU registers.
                STX     TEMPX
                LDX     #8
WRITE_SHIFT     LSR     TEMP            ;Shift out the next data bit.
                BCS     WRITE_ONE
WRITE_ZERO      BCLR    DQ,PORTA        ;Send a zero to the 1820.
                JSR     DELAY_80µS
                BSET    DQ,PORTA
                BRA     DEC_WRITE
WRITE_ONE       BCLR    DQ,PORTA        ;Send a one to the 1820.
                NOP
                NOP
                NOP
                BSET    DQ,PORTA
                JSR     DELAY_80µS
DEC_WRITE       DECX
                BNE     WRITE_SHIFT
                LDA     TEMPA           ;Restore the CPU registers.
                LDX     TEMPX
                RTS
```

```
*****************************************************************************************
*                                                                                       *
* Function Name: READ_1820                                                              *
* Function Inputs: None                                                                 *
* Function Outputs: None                                                                *
*                                                                                       *
* Purpose: This function reads the data from the 1820 and stores it in the              *
*          TEMP variable.                                                               *
*                                                                                       *
*****************************************************************************************

READ_1820       STA     TEMPA               ;Save the CPU registers
                STX     TEMPX
                LDX     #8
READ_BIT        BSET    DQ,PORTA            ;Set up the DQ line for a read
                BSET    DQ_CTRL,DDRA
                BCLR    DQ,PORTA
                NOP
                NOP
                NOP
                NOP
                NOP
                BCLR    DQ_CTRL,DDRA        ;Set the DQ line to receive data
                BRSET   DQ,PORTA,READ_ONE   ;read the data bit.
                CLC
                BRA     READ_SHIFT
READ_ONE        SEC
READ_SHIFT      ROR     TEMP                ;Rotate the bit into the TEMP
                JSR     DELAY_80µS          ;variable
                DECX
                BNE     READ_BIT
                BSET    DQ,PORTA
                BSET    DQ_CTRL,DDRA
                LDA     TEMPA               ;Restore the CPU registers
                LDX     TEMPX
                RTS
```

```
*****************************************************************************
*                                                                           *
* Function Name: FORMAT_TEMP                                                *
* Function Inputs: None                                                     *
* Function Outputs: None                                                    *
*                                                                           *
* Purpose: This function calls the sequence of low level routines that acquire *
*          a temperature reading from the 1820. If the acquisition is       *
*          sucessful, the reading is returned in the TEMP_HI and TEMP_LO    *
*          variables and function return flag is cleared. If an error occurs *
*          while acquiring a reading, the function return flag is set and the *
*          function is exited.                                              *
*                                                                           *
*****************************************************************************

FORMAT_TEMP   CLR   ODD_MULTIPLE  ;Check to see if the temperature reading is an
                                  ;odd multiple .5. If it is set the POINT_FLAG
                                  ;variable
              BRCLR 0,(RAW_TEMP+1) ;NOT_POINT
              INC   ODD_MULTIPLE
NOT_POINT     LDX   #TX_BUFFER    ;Check to see if the temperature is negative.
              LDA   RAW_TEMP      ;If it is place the scan code for "-" into
              BEQ   NOT_NEG       ;the transmission buffer and convert the
              LDA   #MINUS        ;temperature into its positive equivalent.
              STA   ,X
              INCX
              COM   (RAW_TEMP+1)
              INC   (RAW_TEMP+1)
NOT_NEG       LSR   (RAW_TEMP+1)  ;Remove the .5 component of the temperature
                                  ;from the temperature reading.
              LDA   (RAW_TEMP+1)  ;Check for the temperature being greater than
              CMP   #$64          ;100 degrees Celsius.
              BLO   BELOW_100     ;If the value is greater than 100 degrees
              SUB   #$64          ;subtract the value for 100 degrees Celsius
              STA   (RAW_TEMP+1)  ;and store the result.
              LDA   #ONE          ;Store the scan code for a "1" in the
              STA   ,X            ;transmission buffer.
              INCX
BELOW_100     LDA   (RAW_TEMP+1)  ;Divide the reading into its tens and ones
              CLR   QUOTIENT      ;components.
DIV10         CMP   #$0A
              BLO   DIV_DONE
              INC   QUOTIENT
              SUB   #$0A
              BRA   DIV10
DIV_DONE      STA   (RAW_TEMP+1)  ;Find the scan code for the multiple of ten
              STX   TEMP          ;and store it in the transmission buffer.
              TST   QUOTIENT
              BEQ   NO_TENS
              LDX   QUOTIENT
              LDA   SCAN_TABLE,X
```

```
              LDX    TEMP
              STA    ,X
              INCX
              STX    TEMP            ;Find the scan code for the ones component
NO_TENS       LDX    (RAW_TEMP+1)    ;in the scan code table and store it in the
              LDA    SCAN_TABLE,X    ;transmission buffer.
              LDX    TEMP
              STA    ,X
              INCX
              TST    ODD_MULTIPLE    ;If the temperature reading is an odd multiple
              BEQ    WHOLE_NUMBER    ;of .5 degrees Celsius, store the scan codes for
                                     ;the characters ".5" in the transmission buffer.
                                     ;Otherwise store the scan codes for the characters
                                     ;".0" in the transmission buffer.
              LDA    #POINT
              STA    ,X
              INCX
              LDA    #FIVE
              STA    ,X
              BRA    FORMAT_END
WHOLE_NUMBER  LDA    #POINT
              STA    ,X
              INCX
              LDA    #ZERO
              STA    ,X
FORMAT_END    INCX                   ;Store the transmission delimiter character in the
              LDA    #END            ;transmission buffer.
              STA    ,X
              INCX
              LDA    #$FF            ;Store the stop transmission character in the
              STA    ,X              ;transmission buffer.
              RTS
```

```
*******************************************************************************
*                                                                             *
* Function Name: SEND_TEMP                                                    *
* Function Inputs: None                                                       *
* Function Outputs: None                                                      *
*                                                                             *
* Purpose: This function transmits the contents of the transmission buffer to *
*          the PC.                                                            *
*                                                                             *
*******************************************************************************

SEND_TEMP     LDX   #TX_BUFFER     ;Transmit the contents of the transmission buffer.
SEND_LOOP     LDA   ,X             ;If an error occurs, exit the function.
              STX   TX_BUFFER_PTR
              STA   DATA
              JSR   SEND_BYTE
              TST   FLAG
              BNE   SEND_END
              LDX   TX_BUFFER_PTR
              INCX
              LDA   ,X
              CMP   #$FF
              BEQ   SEND_END
              LDA   #2
              STA   TEMP
              JSR   DELAY_500µS
TX_DELAY      DEC   TEMP
              BNE   TX_DELAY
              BRA   SEND_LOOP
SEND_END      RTS

*******************************************************************************
*                                                                             *
*                              SCAN_TABLE                                     *
*                                                                             *
*******************************************************************************

SCAN_TABLE    FCB   $45            ;SCAN CODE FOR "0"
              FCB   $16            ;SCAN CODE FOR "1"
              FCB   $1E            ;SCAN CODE FOR "2"
              FCB   $26            ;SCAN CODE FOR "3"
              FCB   $25            ;SCAN CODE FOR "4"
              FCB   $2E            ;SCAN CODE FOR "5"
              FCB   $36            ;SCAN CODE FOR "6"
              FCB   $3D            ;SCAN CODE FOR "7"
              FCB   $3E            ;SCAN CODE FOR "8"
              FCB   $46            ;SCAN CODE FOR "9"
```

```
*************************************************************************************
*                                                                                   *
*                              TIME DELAY ROUTINES                                  *
*                                                                                   *
*************************************************************************************

ERROR_DELAY       LDA       #$40
                  BRA       CLOCK_LOOP
FULL_CLOCK        LDA       #7
                  BRA       CLOCK_LOOP
HALF_CLOCK        LDA       #3
CLOCK_LOOP        DECA
                  BNE       CLOCK_LOOP
                  RTS

CMD_DELAY         LDA       #$D4
                  BRA       CMD_LOOP
CMD_LOOP          DECA
                  NOP
                  BNE       CMD_LOOP
                  RTS


AFTER_BYTE        LDX      #$2
AFTER_LOOP        JSR      FULL_CLOCK
                  DECX
                  BNE      AFTER_LOOP
                  RTS

DELAY_80µS        LDA      #$0C
                  BRA      DELAY_LOOP
DELAY_100µS       LDA      #$0F
                  BRA      DELAY_LOOP
DELAY_500µS       LDA      #$52
                  BRA      DELAY_LOOP
DELAY_LOOP        NOP
                  NOP
                  NOP
                  DECA
                  BNE      DELAY_LOOP
                  RTS

BYTE_DELAY        LDX      #$18
                  JSR      FULL_CLOCK
DELAY_BYTE_LOOP   DECX
                  BNE      DELAY_BYTE_LOOP
                  RTS

RESPONSE_DELAY    LDA      #$7
                  STA      TSCR
RESPONSE_LOOP     BRSET    6,TSCR,DELAY_EXIT
                  BRA      RESPONSE_LOOP
DELAY_EXIT        RTS
                  ORG      $07FE
                  FDB      START
```

Appendix E. THERMO.EXE Source Code

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>

#define INTR 0x1C                           //Timer interrupt vector

// Function prototypes
void draw_dialog_box(void);                 // displays a dialog box
int acquire_temperature(void);              // acquires a temperature reading
void print_center(int y, char string[]);    // display a string in the center of
                                            // the screen
void interrupt far (*oldhandler) (...);     // original PC timer handler
void interrupt far handler(...);            // replacement PC timer handler

// Global variables
int  counter = 0;                           // timer counter variable
int  error_flag = 0;                        // global error flag
char buffer[80];                            // keystroke buffer


void main(void)
       {
       int c;


       // Turn the cursor off.
       _setcursortype(_NOCURSOR);

       // Acquire and display the temperature until a "q" or
       // "Q" is pressed or an error occurs.
       do
         {
         // Attempt to contact the device and acquire a temperature reading.
         // If the attempt failed, display an error message and exit the program.
         // Otherwise display the temperature in a dialog box.
         if(!acquire_temperature())
          error_flag = 1;
         else
          {
           // Display the temperature in a dialog box.
           draw_dialog_box();

           // Wait for the user to enter a key.
           // If the user presses a 'q' or 'Q' quit.
           while(!kbhit())
```

```
                ;
           c = getch();
          }
       }while((!error_flag) && (c != 'q') && (c != 'Q'));

     // If an error occurred, display an error message.
     if(error_flag)
       printf("Error - Contact was lost with the thermometer.");
     exit(0);
    }

/*                        draw_dialog_box function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function draws a dialog box displaying the temperature.
*
*/
void draw_dialog_box(void)
    {
     // Top of message box display character array
     char top_text[2][80] ={

     "*******************************************************************\n",
     "*                                                                 *\n"};

     // Bottom of message box display character array
     char bottom_text[3][80] ={

     "*                                                                 *\n",
     "*******************************************************************\n",
     "Press Q to quit or any other key to measure the ambient temperature\n"};

     int i; // generic counter variable
     char temp[80];  // temporary string

     // Clear the screen.
     clrscr();

     // Display the message box.
     for(i=0;i<2;i++)
         print_center(i+9,top_text[i]);
```

```
    // Size the message string according to the size of the temperature string.
    if((strlen(buffer)) == 5)
     sprintf(temp,"*        The current temperature is: %s degrees Celcius     *\n",
           buffer);
    else if((strlen(buffer)) == 4)
          sprintf(temp,"*   The current temperature is: %s degrees Celcius      *\n",
             buffer);
    else
       sprintf(temp,"*     The current temperature is: %s degrees Celcius       *\n",
             buffer);

    print_center(11,temp);

    for(i=0;i<3;i++)
       print_center(i+12,bottom_text[i]);

    return;
  }

/*                      acquire_temperature function
*
* Function input variables: None
*
* Function outputs: an integer;
*                    0: If the device failed to respond to the PC.
*                    1: If the device responded to the PC.
*
* This function attempts to contact the device.
*
*/
int acquire_temperature(void)
    {
     int i;  // generic counter variable
     unsigned char c;  // generic character variable

     // Send the keyboard echo command ($EE) twice to the device to signal that
     // PC wishes to contact it.

     counter = 0;

     // Replace the default timer handler routine with the one designed for
     // this program.
     oldhandler = getvect(INTR);
     setvect(INTR,handler);

     for(i = 0;i<2;i++)
       {
        // Send a $EE to the keyboard.
        outportb(0x60,0xEE);

        // Check to see if a response was received to the echo command.
```

```
         // If one was not, clear the function's flag and exit.
         while((!(inportb(0x64) & 0x01)) && (counter < 18))
                 ;

         // If a response is not received within one second, re-install the
         // default timer handler routine, exit this function, and return a zero.
         if(counter > 18)
           {
            setvect(INTR,oldhandler);
            return(0);
           }
        }

        // Initialize the buffer that will hold the temperature reading.
        i = 0;
        memset(buffer,'\0',79);

        // Wait a maximum of two seconds for the temperature string from the
        // device. If a timeout occurs, exit the routine and return a zero.
        // Otherwise return a one.
        do{
           if(kbhit())
             {
              c = getch();
              if(c != '\r')
                {
                 buffer[i] = c;
                 i++;
                }
             }
          }while((c != '\r') && (counter < 36));

        setvect(INTR,oldhandler);

        if(counter < 32)
           return(1);
        else
           return(0);
     }

/*                             print_center function
*
* Function input variables: int y;
*                            vertical position at the string will be printed.
*                            char string[];
*                            string to be centered and printed on the screen.
*
```

```
* Function outputs: None.
*
* This function prints the character string passed to it in the center of the
* screen.
*
*/
void print_center(int y, char string[])
     {
      // Position the string in the center of the string.
      gotoxy (40 - (strlen(string)/2), y);

      // Print the string to the string.
      printf("%s",string);
     }


void interrupt far handler(...)
     {
      counter++;
      oldhandler();
     }
```

Appendix F. AT Keyboard Scan Codes of Common Alphanumeric Characters

Table 1. AT Keyboard Scan Codes

Scan	Character	ASCII Code
045F	0	030H
016H	1	031H
01EH	2	032H
026H	3	033H
025H	4	034H
02EH	5	035H
036H	6	036H
03DH	7	037H
03EH	8	038H
046H	9	039H
01CH	a	061H
032H	b	062H
021H	c	063H
023H	d	064H
024H	e	065H
02BH	f	066H
034H	g	067H
033H	h	068H

Scan	Character	ASCII Code
043H	i	069H
03BH	j	06AH
042H	k	06BH
04BH	l	06CH
03AH	m	06DH
031H	n	06EH
044H	o	06FH
04DH	p	070H
015H	q	071H
02DH	r	072H
01BH	s	073H
02CH	t	074H
03CH	u	075H
02AH	v	076H
01DH	w	077H
022H	x	078H
035H	y	079H
01AH	z	07AH

Application Note

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™ semiconductor