MICROPROCESSOR CONTROLLED EQUIPMENT MONITORING SYSTEM

BY

BRADLEY SCOTT RUBIN

B.S., University of Illinois, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

University of Illinois at Urbana-Champaign

DEPARTMENTAL FORMAT APPROVAL

THIS IS TO CERTIFY THAT THE FORMAT AND QUALITY OF PRESENTATION OF THE THESIS SUBMITTED BY BRADLEY SCOTT RUBIN **AS ONE OF THE REQUIREMENTS FOR THE DEGREE OF** MASTER OF SCIENCE **IS ACCEPTABLE TO THE** DEPARTMENT OF ELECTRICAL ENGINEERING.

(Full Name of Department, Division or Unit)

7 May 1985

Date of Approval

William D. Beck

Departmental Representative

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

MAY 1985

WE HEREBY RECOMMEND THAT THE THESIS BY

BRADLEY SCOTT RUBIN

ENTITLED MICROPROCESSOR CONTROLLED EQUIPMENT MONITORING SYSTEM

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

R. H. Brown

Director of Thesis Research

Ed Ernst

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

ACKNOWLEDGEMENTS

Several key people have contributed to this thesis. I would like to thank my unofficial advisor, Robert Dobinson, for his help, enthusiasm, expertise, resources, discussions, and for the other opportunities he has provided me. I would also like to thank my official advisor, Richard Brown, who has been my professor, advisor, boss, and friend for the past two years. Richard Brown and Lillian Beck are responsible for the very careful proofreading of this thesis. This work was done at the High Energy Physics group facilities in Loomis Laboratory of the Department of Physics in Urbana, Illinois. Robert Downing, formerly of this group, is responsible for the original idea of a monitoring system using high-integration components. Finally, I would like to thank my family, who for the past 22 years have always encouraged my interests in electronics and computers.

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER 1

INTRODUCTION

The tremendous impact and influence of the microprocessor in many areas over the past decade are well-known. Microprocessors have entered our homes, through sophisticated microwave ovens, dishwashers, refrigerators, and thermostats. Microprocessors have entered our cars, controlling engine functions and reminding us to buckle our seatbelts. Microprocessors have entered medicine, where they perform vital monitoring and diagnostic functions. Microprocessors are also at the center of one of the most interesting products to enter our homes, schools, and businesses- the personal computer.

Although the personal computer and the microprocessor are often associated with each other, the most common application of the microprocessor is not computing, but control and monitoring. Microprocessors are the ideal choice for industrial process control and monitoring because of their easy adaptability and reconfiguration via software to meet new requirements, as well as for their tremendous cost advantages over their mechanical or discrete electrical component counterparts. Microprocessors are chosen because they are universal logic units that are alterable via software, reliable because they offer high function density without moving parts, and are inexpensive because they are mass

produced. These are only some of the reasons microprocessors are replacing traditional hardwired, discrete electronic, and other electrical-mechanical devices.

Microprocessors and their associated family members are undergoing two seemingly opposite trends simultaneously. Functionality per integrated ciruit is increasing and cost per function is decreasing. Integrated circuits are one of the few items where in time one can truly obtain more for less. As costs decrease and functionality increases, microprocessors are entering new areas where the use of a microprocessor had previously been considered too extravagant. Today, microprocessors can justifiably be used in a wide variety of control and monitoring applications to provide some tremendous capabilities.

As an example of the application of some of the above considerations, let us examine the topic of this thesis, which is a monitoring system for a FASTBUS equipment rack. FASTBUS is a very high-speed bus system for data acquisition. It is used in many nuclear particle accelerators, where the data obtained from a particle collision experiment are collected at very high speeds and stored for later analysis by a main-frame computer. Because of the high-speed requirements, expensive ECL circuitry is used to implement this bus system. This ECL circuitry requires sophisticated air and water cooling. The printed circuit boards for the FASTBUS system are mounted in units called crates, and three crates are placed in a standard instrumentation rack. The total value of the electronics inside one FASTBUS rack can approach $200,000. With this much money at stake, not to mention

the cost of interrupting a particle collision experiment for equipment repair, it makes sense to invest $600 to add a monitoring system to monitor various environmental conditions inside this rack.

This monitoring system can monitor power supply voltages and currents, air temperature, water temperature, air flow rates, humidity (which in conjunction with air temperature can be used to warn against condensation on components), as well as detect the presence of smoke, fire, open rack doors, etc. In addition, some control can be provided to trigger a main power circuit breaker if a severe problem is detected.

In addition to the monitoring function, the system could be used for information collection. For example, the air temperature characteristics over time can be obtained by automatically reading the temperature value every hour. In this way, studies of system environmental characteristics can be used to further improve instrumentation cooling system design.

Since a typical large FASTBUS setup may contain about 30 of these instrumentation racks, separate monitoring units in each rack are desirable. Also desirable is a way of centralizing the information gathering, so information from a roomful of instrumentation racks can be coordinated. This implies the need for a network to connect these various monitoring units to a central information gathering system. This would allow decentralized monitoring and centralized information gathering.

The above monitoring system goals coupled with current microprocessor-based technology resulted in a prototype design for a $600 microprocessor-based equipment monitoring system (not

including the cost of the sensors), which is the subject of this thesis. Contained herein is a description of the monitoring system, a description of the monitoring unit (both the hardware and software), and operational instructions for the prototype. As with any project, several interesting subtopics were explored and are also discussed here. These include "overkill" hardware design with very high-integration components, some interesting hardware and software design methodologies, local area network selection decisions, and the use of personal computers in a wide variety of areas, including word-processing of these words. All in all, this thesis has provided me with an opportunity to enhance some previous skills as well as develop some new ones. I hope this thesis will not only provide information for the original problem of the FASTBUS monitoring system, but will be useful for other monitoring applications and microprocessor-based system development in general.

CHAPTER 2

SYSTEM OVERVIEW

The block diagram for the monitoring system is shown in Figure 2-1. The monitoring system can consist of from 1 to 254 monitoring units connected on an ARCNET coaxial cable (RG-62U). There is one monitoring unit per FASTBUS instrumentation rack, and each unit monitors both global rack conditions and individual FASTBUS crate conditions. There are three FASTBUS crates per rack. Also connected on the cable is an IBM Personal Computer. The monitoring units have three functions. First, the units continually poll 64 sensors, which sense voltages, currents, air temperature, water temperature, air flow, water flow, humidity, smoke, etc. The value of a given sensor is compared to internally stored threshold tables. If the sensor value exceeds the upper threshold value or falls below the lower threshold value, the sensor is added to a list of problem values. A message is also sent to the IBM PC over the ARCNET local area network to alert it of the problem. A dual threshold technique can be used, whereby the system can distinguish between sensor values that are slightly out of limits and those that are severely out of limits. While the software was written to support this dual threshold scheme, only the single threshold is active for prototype development simplicity.

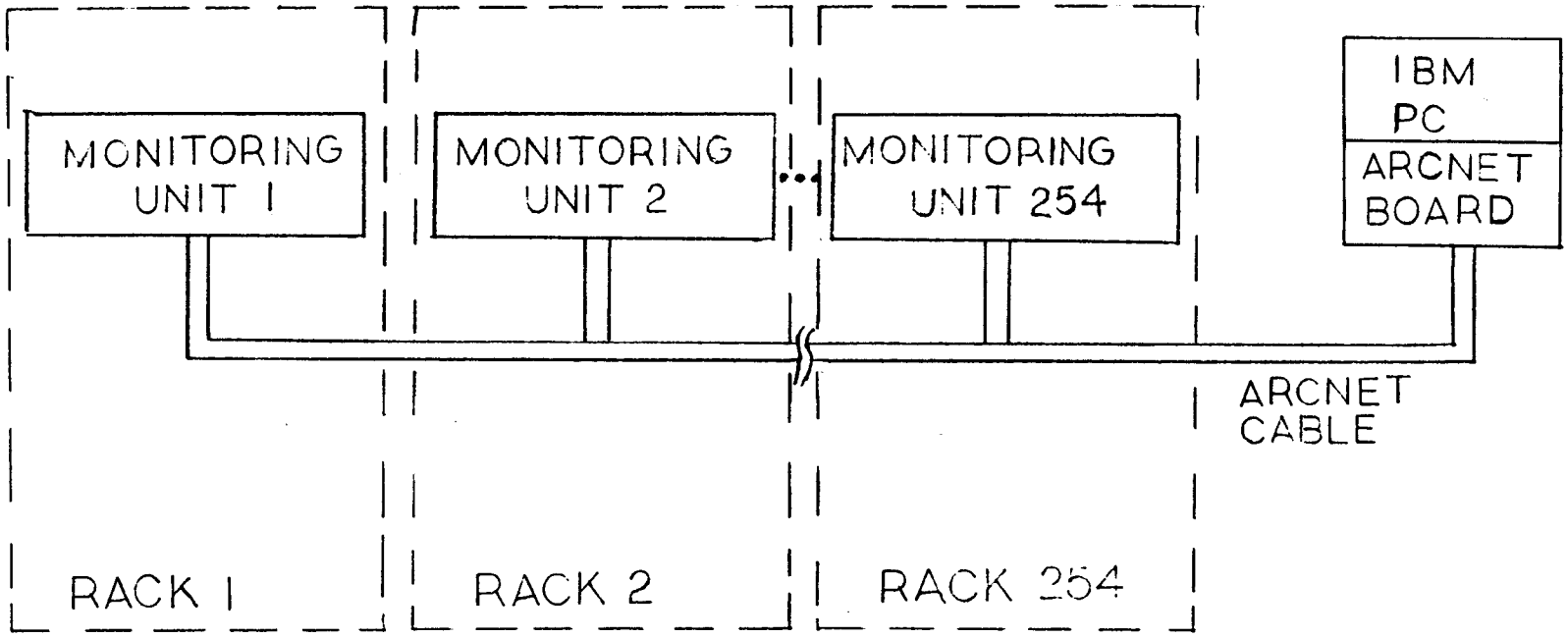


Figure 2-1. Monitoring system block diagram.

A second function of the monitoring unit is to allow a user to interrogate the unit for various information via a 16-key keyboard and a two-line, 20 character-per-line LCD display. A user can display the current value of any sensor, the threshold values for any sensor, and the calibration points for any sensor. The upper line of the LCD display shows the number of sensors that currently are out of the threshold limits. The user can choose to display a list of these sensors. Also, a user can set the threshold values and perform calibrations. A sensor is calibrated by measuring a sensor value with an external device (voltmeter, thermometer, etc.) at two points and setting the monitoring unit to display the desired values at these two points.

The third function of the monitoring unit is to allow the same functions that can be performed at the keyboard and display to be performed remotely over ARCNET, with the IBM PC requesting and receiving information from all the monitoring units. This allows a centralized data base and control system for the decentralized monitoring units. All monitoring units can function alone, or as part of a complete monitoring system.

Throughout the following description, it is assumed that the external sensors with circuitry are available to provide the monitoring unit sensor inputs with scaled voltage values. The sensor considerations are outside the scope of this discussion.

CHAPTER 3

HARDWARE

3.1 Hardware Overview

The following sections discuss various aspects of the hardware design goals, component selection, design considerations and constraints, design methodology, and description of the major hardware components as well as an analysis of their costs. The basic block diagram for the monitoring unit hardware is shown in Figure 3-1. A memory space map and I/O space map are shown in Figure 3-2. The blueprint schematics of the prototype monitoring unit are contained in Appendix D.

The heart of the monitoring unit consists of an Intel 80188 microprocessor. This high-integration processor contains many system support functions. The software for the unit resides in an EPROM, and an IRAM is used for scratchpad and other support purposes. The thresholds and calibration points are stored in a EEPROM because this non-volatile storage can be altered electrically. A parallel port is used, for output, to control the LCD display and, for input, to accept data from the keyboard via the keyboard encoder. The sensors are multiplexed and fed to an analog-to digital converter, which is also connected to the processor bus. The ARCNET local area network support consists of

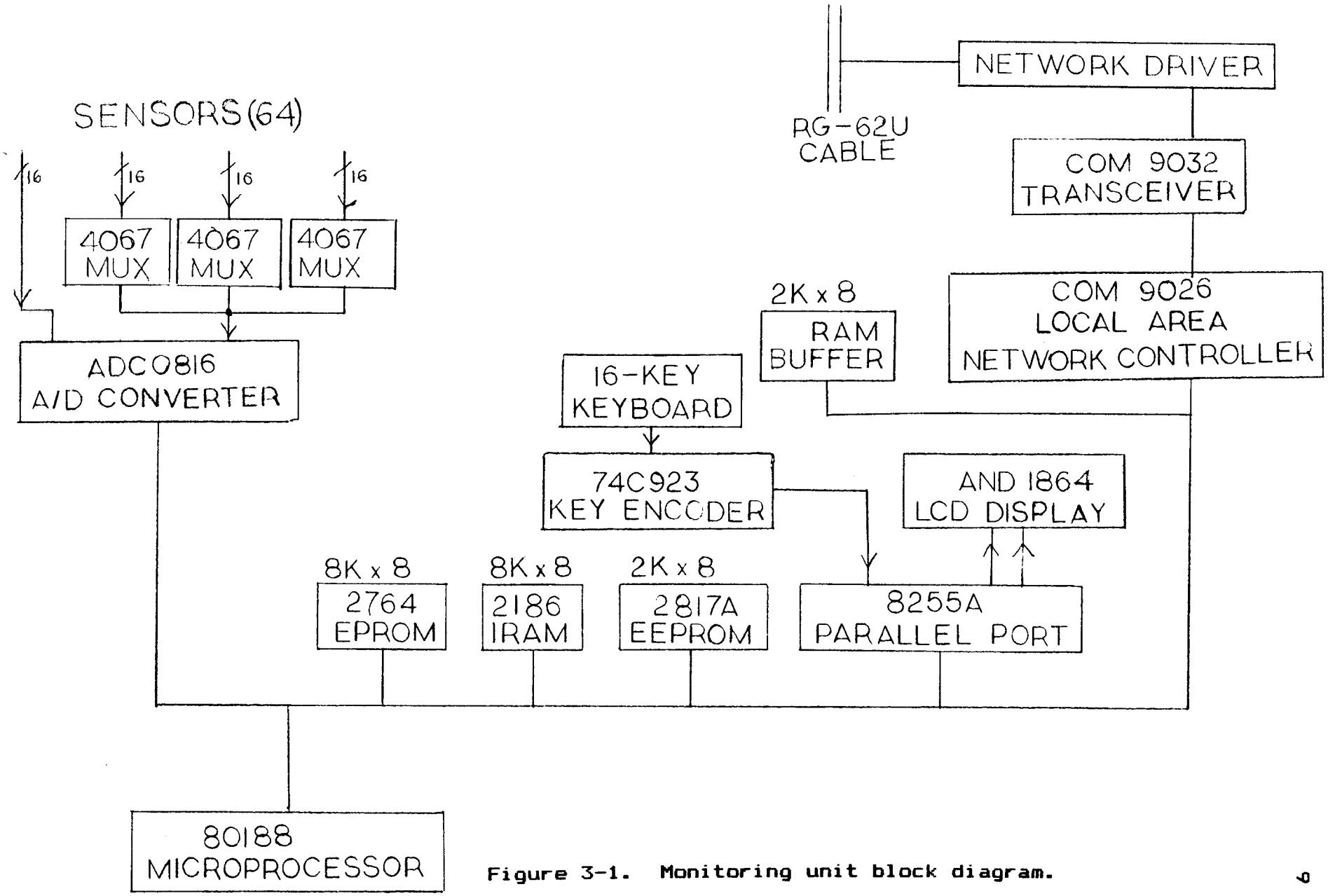


Figure 3-1. Monitoring unit block diagram.

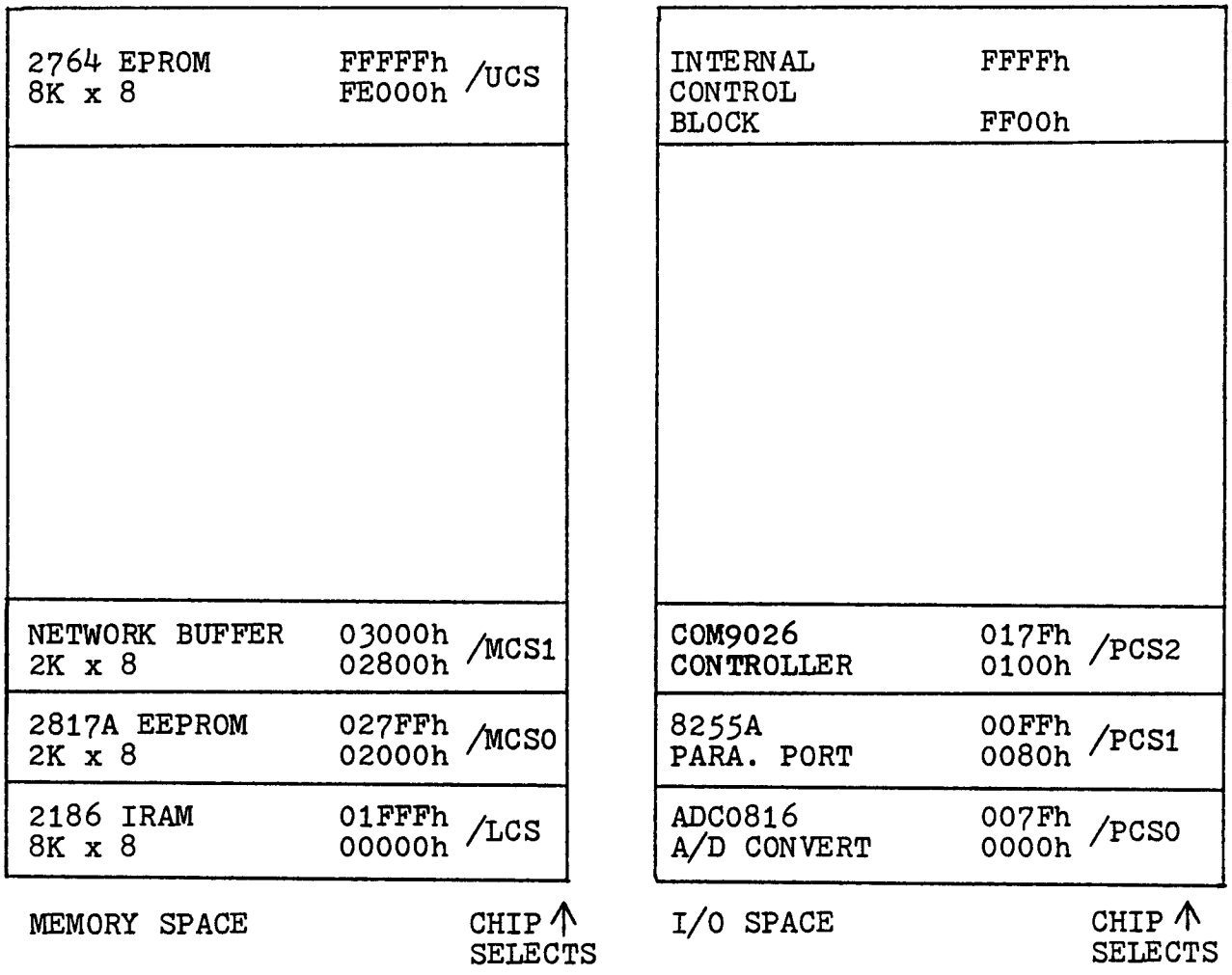


Figure 3-2. Memory and I/O space maps.

a buffer RAM used to hold messages, a controller for protocol support, a transceiver, and a driver that interfaces to the coaxial cable.

3.2 Component Selection

There were several unique design goals for the hardware. A major goal was to take advantage of new, high-integration components whenever possible, depending on availability, to realize several benefits over using more, less expensive, components. First, design time is reduced because design is at a higher level of abstraction. For example, it is easier to design with a processor that has on-chip timers, chip select logic, and interrupt controllers than to design with several chips that implement the various functions separately. Printed circuit board space required is reduced, the number of wiring errors is reduced, power supply requirements are reduced, circuit speed is increased (although this was not a factor in this design), and hardware debugging is easier because of fewer connections, as well as fewer functional units that can become defective. Finally, most of the high-integration components are controlled and configured via software, allowing design changes to be carried out more easily than hardware changes allow.

There are also some disadvantages to using high-integration components. First, using the highest density components available means using new components, which means using more expensive and less available components. Availability was, of

course, a major consideration in component selection, but the cost disadvantage was far outweighed by the above advantages. The final hardware design used the highest integration components available for each function required. These components will be discussed individually in later sections.

One interesting result of using high-integration components is that the resulting system has much excess capacity. All of the capabilities of a given component are usually not needed. This extra capacity may, however, be needed to handle new, unexpected design requirements. It is better to overdesign than to underdesign, and the cost of overdesigning over that of designing close to the original specification capacity is not significant when using high-integration components. For example, if 4K-bytes of EPROM are needed, an 8K-byte EPROM can be used with negligible increase in cost and potentially better availability. The system can then handle a possible increase in EPROM requirements for a future design addition or change. In this case, chip size will not be increased with the 8K-byte EPROM, making it a good component choice.

Another hardware component selection consideration was power supply voltage. To keep power supply costs at a minimum (because the monitoring system must use separate power supplies than the crates use), +5 VDC only components were selected wherever possible. The only component that could not be obtained with a +5 VDC supply was the local area network driver hybrid circuit. This circuit requires +5 VDC and -5 VDC supplies; however, the power supply current requirements for the -5 VDC supply are low enough that a DC to DC converter could be used.

3.3 Component Description

In this section, each major hardware component is described. Selection criteria for each component as well as special design considerations that pertain to its use in this project are given. No attempt is made to comprehensively describe the components. Since data sheets and applications notes for the components are readily available (see references), only those aspects and peculiarities that pertain to this specific monitoring system design are explored.

3.3.1 Intel 80188 Microprocessor [1],[2],[3]

The heart of the system is the Intel 80188-3. This component is a good example of using a high-integration component to replace a large number of other chips. The 80188 contains an enhanced 8088-2 CPU, a clock generator, two independent, high-speed DMA channels, a programmable interrupt controller, three programmable 16-bit timers, programmable memory and peripheral chip-select logic, a programmable wait state generator, and some bus control. All of these functions are implemented on one 68-pin leadless JEDEC type A hermetic chip carrier. The 80188 effectively combines 15-20 of the most common iAPX 88 system components into one.

The 8-bit 80188 processor has a 16-bit internal architecture

and an 8-bit data bus interface with a maximum bandwidth of 2 Mbytes/second. The processor can run at 8 MHz (although a 4-MHz rate was chosen for this design), and provides two times greater throughput than the standard 5-MHz 8088. The 80188 is upward compatible with the 8086 and 8088 software and adds ten new instruction types to the 8088/86 instruction set. The processor has a direct addressing capability of 1 Mbyte of memory, and can support an optional numeric processor extension, the Intel 8087 high-performance 80-bit numeric data processor.

The 80188 requires only a +5 VDC power supply, but it draws 550 mA of current and dissipates 3 watts of power. A small heatsink was glued to the top of the chip, and the system has been run without forced air cooling without problems, but the heatsink still gets quite hot to the touch. There should, however, be sufficient airflow at the top of the rack where the system will eventually be mounted to keep the entire board safely cooled.

The lower eight address lines and the eight data lines share the same 80188 pins (AD0-AD7). For this application, the ALE signal from the 80188, which indicates when the address is stable and on the lines, is used to latch the lower eight address lines. In this way, the multiplexed lines are demultiplexed for distribution to the rest of the components.

Clock speed, as previously mentioned, was chosen to be half of the maximum value (this decision is explained later). Since a clock speed of 4 MHz is needed and the crystal frequency must be twice the clock speed, an 8-MHz crystal was used.

The 80188 reset input (/RES) is driven by a power-on reset

circuit. The reset output (RESET) drives the other components that require a power-on reset. The exception to this is the network controller. This reset is driven by an output bit of the parallel port because processor-controlled reset of the network may be a desirable feature for the future.

The drive capability of the 80188 is 2.0 mA for all outputs, except for /S0-/S2 which can drive 2.5 mA. Maximum output capacitance loading for all pins is 20 pF. The monitoring unit prototype loads the 80188 within specifications, and no extra bus drivers are required.

The 80188-3 used for the prototype is what Intel calls a B-1 stepping. This chip has certain defects, both functional and electrical. The monitoring unit was designed so that both this "defective" chip and a chip that meets all the specification sheet requirements will work. The timing values that do not meet specifications for the B-1 stepping are given in Table 3-1. The impact of the functional defects will be discussed in the software section when the defect involves a software consideration that differs from specification descriptions.

In conclusion, the hardware aspects of the 80188, most notably the high-integration of this component, made it a good processor choice. Although processor characteristics are an important part of processor selection, the availability of other high-integration components in the processor's family is more important. As will be shown, there are many family members that prove very useful to the overall monitoring unit design. The 80188 is a well-documented, available, and interesting processor with many advanced features, and is a member of a family of

TABLE 3-1. ELECTRICAL DEFECTS IN 80188 B-1 STEPPING [4]

The following specifications are the 80188-3 B-1 stepping differences from the data sheet specifications for the completely functional 80188-3. [1],[3]

VCC: 5 V +/- 5% (Instead of 10%)

OPERATING TEMP RANGE: 0°C to 50°C (Instead of 70°C)

DC PARAMETERS:
VCLI = 0.3V

AC PARAMETERS:
TCLDX Min = 20 ns
TCLAV Max = 59 ns
TCHCTV Max = 73 ns
TCLHAV Max = 67 ns
TAVAL Min = TCLCH - 40
TCHSV Max = 65 ns
TCLSH Max = 65 ns
TCLCSV Min = 12 ns
TCHCSX Max = 47 ns
TCVDEX Max = 70 ns

Note: TCVDEX is a new specification. It is the /DEN inactive delay for a non-write cycle.

components with the same characteristics.

3.3.2 Intel 2764 EPROM [2],[5]

The Intel 2764 is a +5 VDC only, 64K-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). This EPROM is used to store the system software and data tables for the monitoring system. After the object code for the system software is obtained from the assembler on the IBM PC (see the section on the software development methodology), it is programmed into the EPROM using an EPROM programming board. The EPROM is then removed and placed into a socket on the monitoring system board. The EPROM resides in the uppermost 8K-byte region of the 80188 address space, where it is selected by the Upper Chip Select (UCS) line. The top of memory is usually used as the system memory because after a hardware reset, the 80188 begins executing at memory location FFFF0H, which is 16 locations from the top of memory. The systems boots here, and then jumps to the beginning of the program, which is located 4K bytes from the top of memory. This leaves 4K bytes for future program or table expansion.

The upper limit of memory defined by the Upper Chip Select line is fixed at FFFFFH, and the lower limit is defined by the block size selected in the UMCS register in the 80188. The UMCS register is at offset A0H in the internal control block. After a hardware reset, the UMCS register is programmed for a 1K-byte area. It must be reprogrammed for a block size of 8K bytes, to

insert zero wait states, and to ignore external ready. The next section describes the computations for determining the number of wait states. It is important to note that the UMCS reprogramming described above must be contained in the upper 1K-byte area, and it is too large to fit in the upper 16 bytes because the upper 4 bytes are reserved by Intel. Therefore, the initial code executed in the upper 16 bytes after a hardware reset jumps to a location 32 bytes from the top of memory, where the reprogramming is done, and finally jumps to the system code which starts 4K bytes from the top of memory.

The specific part chosen (because it happened to be on hand) was the Intel 2764-4, which has an access time of 450 ns. This access time is compatible with the 81088 running at 4 MHz, and with zero wait states inserted (refer to section 3.3.2.1).

There is a subtle problem with using this specific part with the 80188 running at full speed (8 MHz), as opposed to 4 MHz. The memory read hold time, which is the time between Output Enable inactive to data output float (referred to as T_{DF} by Intel) is 130 ns, which violates the 80188 constraint T_{RHAV}, which is T_{CLCL} - 40 = 125 - 40 = 85 ns. T_{RHAV} is the time from /RD inactive to address active and T_{CLCL} is the clock period. This means that a data buffer chip must be inserted between the EPROM data lines and the address/data bus, since the EPROM may continue to drive data information on the processor bus when the processor begins to drive address information for the next bus cycle. Since high speed is not one of the major design goals, the relaxation of the processor clock speed in favor of more available, slower, less expensive (although the cost differential is insignificant)

memory or extra buffer chips was chosen over operation at maximum processor speed.

3.3.2.1 Wait state computation [2]

The 80188 has the capability for programmable wait state generation, whereby anywhere from zero to three wait states can be inserted into the bus access cycle. For 4 MHz operation, each wait state inserts one clock period of 250 ns. The following analysis demonstrates how to calculate the number of wait states to use for the Upper Chip Select Line, which drives the Chip Enable line of the Intel 2764-4 EPROM. Three basic constraints relate the number of inserted wait states to timing parameters of the EPROM and of the 80188. Solution of these equations allows the number of wait states used to be determined. In addition to these three constraints, a fourth constraint relates the 80188 read high to address valid time (T_{RHAV}) to the 2764-4 data hold time, or Output Enable inactive to data output float (T_{DF}). The following equations are solved for the 80188 (using the data sheet values for the timing values) and the 2764-4 EPROM.

1) $T_{ACC} = (3+N)T_{CLCL} - T_{CLAV} - T_{IVOV} - T_{DVCL} > T_{ACC}$

2) $T_{CE} = (3+N)T_{CLCL} - T_{CLCSV} - T_{DVCL} > T_{CE}$

3) $T_{OE} = (2+N)T_{CLCL} - T_{CLRL} - T_{DVCL} > T_{OE}$

4) $T_{RHAV} = T_{CLCL} - 40 > T_{DF}$

where:

T_{ACC} = 450 ns = EPROM Address to output delay

T_{CE} = 450 ns = EPROM /CE to output delay

T_{OE} = 150 ns = EPROM /OE to output delay

T_{DF} = 130 ns = EPROM /OE high to output float

T_{CLCL} = 250 ns = 80188 CLKOUT period

T_{CLAV} = 44 ns = 80188 Address valid delay

T_{DVCL} = 20 ns = 80188 Data in setup time

T_{CLCSV} = 66 ns = Chip select active delay

T_{CLRL} = 70 ns = /RD active delay

T_{IVOV} = 18 ns = 74LS373 Data input valid to data output valid

N = Number of wait states

Solving:

1) N > -1.8

2) N > -1.8

3) N > -1.5

4) 210 > 130

Therefore, N=0 wait states inserted and the T_{RHAV} constraint is not violated.

3.3.3 Intel 2186 IRAM [6],[7],[8]

The Intel 2186 is an 8K by 8-bit integrated random access

memory (iRAM). By integrating refresh control along with dynamic RAM technology on the same chip, static RAM characteristics are obtained at a significantly lower cost than with a pure static RAM or with the extra parts count and added design complexity of a separate dynamic RAM and refresh control. The iRAM is used in the monitoring system for general scratchpad RAM functions and variable storage, as well as for storage of interrupt vector locations.

The lower limit of memory is defined by the Lower Memory Chip Select (LMCS) line from the 80188. The lower limit of memory defined by the LMCS line is always 0H, while the upper limit is programmable giving the lower memory block size. The upper limit of the lower memory block is defined in the 80188 LMCS register, which is at an offset of A2H in the internal control block. The LMCS register is programmed for a block size of 8K-bytes, with 2 wait states and external ready used.

The specific part used was the Intel 2186-35, which has a cycle time of 600 ns. It is capable of being operated in one of two modes, late cycle or pulse. Pulse mode is entered if the /CE signal is low to the device for a maximum of 130 ns, and requires the /RD or /WE command to go active within 90 ns after /CE. Because of these requirements, late cycle mode results in a much simpler interface. The iRAM automatically selects between these two modes.

The 2186 is a leading edge triggered device, which is different from the trailing edge operation used with the 80188 processor bus. This means that the control signals /LCS and /WR must be delayed by one clock cycle so that the active edge of the

control signals (the high-to-low transition) occurs when the processor address and data signals are guaranteed stable.

Refresh is totally automatic and requires no external signals or circuitry. An internal arbitration circuit will resolve any conflicts between internal refresh cycle requests and external data read or write requests. Refresh cycles have priority, which means a processor memory request may be delayed until refresh is finished. This means the iRAM ready line must be factored into the processor-iRAM cycles.

3.3.4 Intel 2817 EEPROM [9],[10]

The Intel 2817A Electrically Erasable Programmable Read Only Memory (EEPROM) is non-volatile memory like a ROM, erasable like an EPROM, but no ultraviolet light is needed. Each individual byte can be erased electrically and rewritten. Furthermore, the 2817A has a high degree of integrated functionality which enables in-circuit byte writes to be performed with minimal hardware and software overhead. The device has complete self-timing which leaves the processor free to perform other tasks until the 2817A signals that it is ready to perform another write. The erase before a write is transparent to the user, and the address and data lines are latched on the chip. The 2817A also contains on-chip write protection circuitry so that data are not accidentally altered during power up or power down. The device requires only a +5 VDC power supply.

An EEPROM was selected for the monitoring unit design because

it allows storage of the parameter threshold values and calibration data when the system power is off, yet it allows these values to be easily altered via a program. Otherwise, a battery powered RAM or an EPROM would have had to been used, making updates and maintainence extremely cumbersome. The 2817A has a life cylce of ten thousand write cycles per byte; however, the parameters and thresholds stored in the EEPROM are changed very infrequently, making the EEPROM a nearly ideal solution to the design requirements.

The EEPROM resides in the middle portion of memory. The Chip Enable line is driven by one of the four 80188 Mid-Range Memory Chip Select lines (MCS0). The size of the memory block defined by the MCS lines is determined by the contents of the MPCS register, which is located at location A8H in the internal control block. The total block size used is 8K-bytes, allowing for four contiguous areas of 2K bytes selected by the four MCS lines. The base address of the mid-range memory block is defined by the contents of the MMCS register, which is located at location A6H in the internal control block. The base address may be set at any integer multiple of the size of the total memory block. In this design, the block is placed immediately after the iRAM block in the low portion of memory.

All four sub-blocks in the middle memory share the same wait state generation control. The second sub-block is occupied by the local area network controller memory, so both memories must be considered for proper wait state generation and ready handling. The network memory can have delays associated with its bus cycle, so three wait states are used along with external

ready.

The 2817A is interfaced to the processor bus like any standard memory (see the 2764 EPROM description), except for one point. After a write cycle is initiated by the WE pulse, the RDY output goes low indicating that a write operation is in progress. After an amount of time has elapsed to insure successful programming, the RDY output returns high to indicate the write operation is complete. There are two basic ways of handling this line in the system; the output can be used as an interrupt to the CPU, or it can be polled to determine when it is all right to perform another write after a previous write. The interrupt technique was chosen for the monitoring unit because it allows for maximum throughput by allowing the processor to continue executing other instructions while the EEPROM continues the write operation. Thus, the RDY output line is connected to the 80188 Interrupt 1 (INT1) line. The interrupt handler sets a flag when the EEPROM is available, and this flag is tested before a write operation is performed.

3.3.5 Intel 8255A Peripheral Interface [11]

The Intel 8255A is a general purpose programmable I/O device which has 24 I/O pins that may be individually programmed in two groups of 12 or three groups of 8, and each group may be used in three major modes of operation. Various handshaking and interrupt handling schemes are possible, and various combinations of groups may be configured as input or output groups. The

device is designed to interface with Intel processor busses.

In the monitoring unit design, this component is used to drive the LCD display with 16 output lines, supply the network controller with a reset signal with one output line, and accept keyboard input data with four input lines. Although the LCD display is capable of being interfaced directly to a microprocessor bus, the cycle times, data setup, and data hold times are too slow to use with the 80188 (see description of LCD display). The LCD display control signals are thus driven directly by the output of a parallel port.

The device is configured in Mode 0, which is the basic input/output mode. This functional configuration provides simple I/O operations for each of the three ports without handshaking. The basic functional definition of Mode 0 includes two 8-bit ports, two 4-bit ports, and configuration of any port to be input or output. Outputs are latched, inputs are not latched, and there are sixteen possible I/O configurations. A control word specifies the I/O configuration, and this word is at I/O location 03h in I/O space. Programming this register with the value 81h configures the device to port A being an output, port B being an input, the upper half of port C being an output, and the lower half of port C being an input. The 80188 has seven Peripheral Chip Select (PCS) lines that allow selection of seven contiguous blocks of 128 bytes above a programmable base address that is a multiple of 1K bytes. This space may be located in either memory or I/O space. The base address of the PCS block is defined by the PACS register, which is at offset A4H in the internal control block. This register also contains three bits that define the

wait state and ready line use for the block, just as with the memory chip selects. Because the local area network controller also shares this I/O space and because its bus cycle may be delayed, three wait states are inserted as well as the ready line factor. I/O space is also used rather than memory mapped I/O mainly for conceptual purposes; memories and buffers are in one space, control registers and I/O ports are in another. The configuration is programmed in the MPCS register, the same register used for the Mid-Range Memory Chip Select.

3.3.6 AND 1864 LCD Display [12]

The AND 1864 LCD display is a display unit that contains on-board drivers, RAM, and ROM, that allow for a simple ASCII interface to a processor. The display unit requires a +5 VDC power supply and contains CMOS circuitry. The AND1864 can display either 256 different characters, or 128 characters with an underline option. The display format is 40 characters, arranged in two rows of 20 characters. The 5.45 mm high characters are easily readable in high light conditions. The viewing angle is claimed to be large (10 to 40 degrees), but in actuality the viewing angle is rather limited, and viewing from zero degrees is very desirable.

This display is used in the monitoring unit for menu prompting and display of values and parameters, as well as display of sensor values out-of-range. The upper line is used for the out of range display and the lower line is used for the

menu selection and parameter display. The 20 character line, however, is rather limiting. A better display choice may be the AND021, which has two lines of 40 characters per line.

The integrated LCD display was chosen for its high integration, relative low cost, good availability, +5 VDC power supply requirement and good display characteristics in light conditions. The LED displays, fluorescent displays, and plasma displays available failed to meet the above requirements. While not an ideal choice because of interface speed problems and viewing angle, the LCD display proved workable.

The major disadvantage of the display involves its interface requirements. The display is designed to interface directly to 4 or 8-bit processor busses, but the slow speed of its CMOS circuitry does not make that possible in this design. The cycle time for a write operation to the display is 1 microsecond, which could be achieved with external ready generation to insert wait states. The display, however, requires a data setup and hold time of 500 ns, which makes direct connection to the 80188 bus not possible. This problem is solved by connecting the display data and control lines to the 16 outputs of the parallel port. The bits are then set and reset to control the display transfers.

The display operates much like a memory. Six address bits (A1-A5) specify which of the 20 character positions are used. A single row bit (R1) specifies the upper or lower row. Eight data bits (D1-D8) specify the character to be placed at the specified display address. A display line (DISP) is low for a write or read, and high for normal display. Read and write lines (RD and WR) are brought high for read or write operations. When the

cursor inhibit (CURINH) input is low, D8 specifies an underline. When CURINH is high, 256 characters are used. This design uses the underline mode which is useful for prompting. Since 16 output lines are conveniently available from the parallel port, and 17 are needed for full display operation as well as another eight bits of input port for the read data, the RD line is not used, so character reads from the display are not possible. This information can be kept in RAM if it is ever required (it is not in this design). Also, since CMOS inputs require a higher logic one input voltage than the outputs of the parallel port provide, pull-up resistors to +5 VDC are used on all display input lines.

3.3.7 National MM74C93 Key Encoder [13]

The National Semiconductor MM74C93 20-key encoder is a CMOS device that provides all the necessary logic to fully encode an array of SPST switches. The keyboard scan can be implemented by either an external clock or external capacitor. An internal debounce circuit needs only a single external capacitor. A Data Available output goes high when a valid keyboard entry has been made. The Data Available output returns low when the entered key is released, and two-key rollover is provided. The key data are latched internally, and the outputs are tristate.

A 16-key matrix keyboard is used in the monitoring system to provide menu selection and value entering. A 20-key encoder was chosen for a possible future need to add more keys. Sixteen keys appear to be sufficient for the current application.

The data output from the encoder is the input to the 4-bit parallel input port and the Data Available line is connected to the 80188 Interrupt 0 (INT0) line, so an interrupt is generated during every key press. The Data Available line is also inverted and applied to the /Output Enable input of the encoder, which enables the latched keyboard data to the tristate outputs. This is referred to as asynchronous data entry onto bus by National. The capacitor connected to the KBM input controls the debounce period and has a value about ten times the scan oscillator capacitor value, which is connected to the OSC input. The values were selected to yield a scan frequency of 500 Hz and a debounce period of 0.01 second.

3.3.8 National ADC0816 A/D Converter [14],[15]

The National Semiconductor ADC0816 is an 8-bit microprocessor compatible analog-to-digital (A/D) converter that contains an on-chip 16-channel analog multiplexer. This 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 16-channel multiplexer can directly access any one of 16 analog inputs and the chip contains logic for additional channel expansion. The converted value of the analog input has 8 bits of resolution with a total unadjusted error of +/- 1/2 LSB. The conversion time is 100 microseconds. This is a CMOS device that requires a single +5

VDC power supply.

The A/D converter is used in the monitoring unit design to convert any one of 64 scaled sensor inputs to an 8-bit value. Sixteen of the 64 sensor signals are input directly to the on-chip multiplexer, while the remaining 48 sensor signals are input to three 4051 16-to-1 CMOS analog multiplexer chips that are connected to the expansion input on the ADC0816, which is connected directly to the comparator input of the converter. In this way, a single A/D chip and a few other components can allow conversion of any one of 64 analog signal inputs.

The ADC0816 and associated components are selected by the /PCS1 line from the 80188, which means they exist in the second 128-byte sub-block of I/O space. The ADC0816 clock is derived from the CLKOUT output of the 80188 (which is a 4-MHz signal) that is passed through a divide-by-four counter yielding a 1-MHz signal. All inputs to the ADC0816 must be pulled up to +5 VDC with a resister because the ADC0816 is a CMOS device.

The interface technique used is somewhat complex and requires some detailed explanation. All control signals are derived from the chip select line (/PCS1), read (/RD), and write (/WR), all of which come from the 80188. Before a conversion cycle is started, a software ready flag is checked to make sure it is clear, indicating no previous conversion cycle is still active. If the ready flag is reset, the flag is then set. To begin a conversion cycle for a given sensor, a "dummy" input instruction is executed with a port address equal to the sensor number plus the base address of the ADC0816 in I/O space. The execution of this instruction will cause the sensor address to be latched and

either the ADC0816 or one of the three 16-to-1 analog multiplexer chips to be selected, depending on the sensor address. If the ADC0816 is selected, the input instruction also causes the sensor number to be internally latched via the ALE input. Note that the "dummy" input instruction is used only to generate control signals and to specify the sensor number, and that the data that are input do not have meaning.

After the sensor number is latched, a "dummy" output instruction is executed to generate the START control signal for the ADC0816. The actual data that are output are not important and the output port address is constrained only to lie within the /PCS1 select space. The conversion cycle now commences. When the conversion cycle is complete, the ADC0816 End of Conversion (EOC) output goes high. This line is connected to the 80188 Interrupt 1 (INT1) interrupt input. The interrupt handler executes a "real" input instruction which reads the converted result. The ready flag is then reset to indicate that the ADC0816 is ready for another conversion cycle. Note that if the port address specified by this "real" input instruction is related to the next sensor number to be converted, then this new sensor number can be latched at the same time the current converted value is retrieved. Both tasks can be accomplished through the same input instruction.

While the interface technique of generating control signals with input and output instructions is somewhat complicated, it has the advantage of not requiring any additional control signals to be generated from other components (i.e., a parallel port). Since the actual instruction sequence required for proper

operation can be imbedded within a subroutine to read the ADC0816 (given the sensor number) and within its associated interrupt handler, this interface technique result in a simple software interface while requiring few hardware components.

3.3.9 SMC COM9026 Network Controller [16],[17],[18]

The Standard Microsystems Corporation COM9026 is a local area network controller that implements the ARCNET network protocol on one chip. ARCNET uses a modified token passing protocol that features self-reconfiguring as nodes are added or deleted from the network, handles variable length data packets, and supports up to 255 nodes per network segment. The data rate is 2.5 MBPS and the protocol is compatible with broadband or baseband systems and with any interconnect media (twisted pair, coax, etc.). Arbitrary network configurations can be used (star, tree, bus, etc.). This chip replaces over 100 MSI/SSI parts, and is the heart of the network interface.

The COM9026 is used in the monitoring unit to allow communication between the 80188 and the IBM PC while other monitoring units are also communicating over the same network through their own COM9026 network controllers. A very important feature of the COM9026 is that it is highly integrated. The chip handles all of the network protocol, leaving other hardware and software free of this difficult job. Messages are loaded into a transmit buffer with a specified destination address and a transmit command is issued. Similarly, a receive command is

issued and message is retrieved from a recieve buffer via a receive command. This layering of function available with this component nicely demonstrates the benefits attained when high-integration components are used in hardware design. Because of the availability of this type of component, local area network capability was added to a design with very little difficulty.

For this system design, the ARCNET local area network specification standard defined by Datapoint was followed. This standard uses a maximum message length of 253 bytes, a token bus configuration using RG-62U coaxial cable, and a maximum cable length of 2000 feet.

An external RAM buffer of up to 2K-byte locations is used to hold up to four data packets with a maximum length of 508 bytes per message. In the ARCNET standard configuration the maximum message length is 253 bytes. These four buffers can be used to implement double-buffered transmit and receive functions. This RAM buffer is accessed both by the processor and the COM9026. The processor can also read status and write commands to the COM9026. The COM9026 provides all signals necessary to allow arbitration of the network data bus, on which the processor, the COM9026, and the buffer memory are connected. This fairly complex connection of devices on the internal bus necessitates some tricky control signal generation for all of the data latches and buffers required.

The network interface requires both address space and I/O space. The 80188 has access to the buffer memory using the /MCS1 chip select line, which means the 2K-byte buffer memory exists in the second 2K-byte sub-block of the mid-range memory space. The

80188 has access to the COM9026 registers through the /PCS2 peripheral chip select line, which means the registers exist in the third 128-byte sub-block of I/O space. On all buffer memory or COM9026 accesses done by the processor, there is the possibility of contention on the internal network bus. This is arbitrated by the COM9026, but means the 80188 is likely to have to insert wait states in its bus access cycle. This is accomplished via the COM9026 WAIT line and the 80188 ARDY line. In addition to the COM9026 arbitration control lines, the 80188 /RD, /WR, and ALE lines are used for control of the latches and buffers that provide bus isolation and address and data latching.

Although the COM9026 can use a 2K-byte RAM if 512 byte messages are used, and needs only a 1K-byte RAM for this application because a message size of 256 bytes is used, a 8K-byte RAM is used in this hardware design. The availability of 8K-byte static RAMS is very good compared with the rather poor availability of 2K-byte RAMS. The 8K-byte RAM, however, is rather expensive, and overall is a poor choice. Availability was an important concern at construction time and the 8K-byte RAM was chosen even though it is expensive and requires slightly more board space. A more ideal chip for the buffer is the Intel 81C28, which is a 2-K byte CMOS RAM that has an internal address latch, which would eliminate the need for the 74LS373 latch. This chip, however, was not available at the time the circuit was constructed.

A slight disadvantage of the COM9026 is that it does not allow software control of the station ID that is used to specify source and destination information for transmitted and received

message packets. The station ID can, however, be read by the processor. An 8-position DIP switch on the monitoring system board is used to set the station ID. The address is converted from parallel to serial with a shift register, and read into the COM9026 through the serial IDDAT input.

3.3.10 SMC COM9032 Transceiver [18]

The Standard Microsystems Corporation COM9032 transceiver chip is a companion to the COM9026 controller. It provides two functions. First, the chip provides two 5 MHz clock signals from a 20 MHz TTL-level oscillator source. One of these signals is free running and is used to drive the CLK input of the COM9026 and the ID shift register. The other clock is synchronized to the received data under the COM9026 control for message reception.

The COM9032 also provides the serial data link to the controller via the /TX and RX pins of the COM9026. The standard ARCNET interface is a baseband system using dipulse signaling on RG-62U coaxial cable and allows cable runs of up to 2000 feet. The output of this chip on the serial data link side consists of a pair of signals, /PULSE1 and /PULSE2. These are then fed to the line interface circuit. For data reception, the input from the line driver is converted to NRZ and sampled to yield a 400 ns pulse on the controller RX input.

3.3.11 Zenith EG-A059101A Line Driver [18],[19]

The Zenith local area network line driver provides the interface between the local area network transceiver and the RG-62U coaxial cable. The other monitoring units in the system as well as the IBM PC are connected to this same cable. This component is a hybrid of digital and analog integrated and discrete components mounted on a ceramic substrate and encapsulated. This component contains all the circuitry necessary for the interface.

The line driver takes the two 100 ns pulses obtained from the network transceiver and generates a 200 ns wide dipulse on the coaxial cable for transmission. Received dipulses are passed through a matched filter and line receiver. The output of the line receiver is connected to the COM9026.

3.4 Cost Analysis

Table 3-2 is an itemized list of the integrated circuit costs as well as the costs of some of the major discrete components. The only major cost not included in the table is the cost of sockets for the integrated circuits as well as that of the prototype board. As the table indicates, the cost of the monitoring unit components is just over $500.00, which would make each monitoring unit cost roughly $600.00 total. This final cost does not include the cost of the various sensors required.

TABLE 3-2 ITEMIZED COMPONENT COST

SCHEM. NUMBER	PART NUMBER	MANUFACT.	DESCRIPTION	COST
IC1	80188-3	Intel	Microprocessor	$ 79.00
IC2	74LS373	TI	8-Bit Latch	1.70
IC3	2764-4	Intel	8Kx8 EPROM	10.80
IC4	2186-35	Intel	8Kx8 IRAM	18.75
IC5	2817A	Intel	2Kx8 EEPROM	52.60
IC6	74LS00	TI	2-Input NAND	0.26
IC7	74LS00	TI	2-Input NAND	0.26
IC8	74LS04	TI	Hex Inverter	0.26
IC9	8255A-5	Intel	Parallel Port	10.80
IC10	ADC0816	National	A/D Converter	9.95
IC11	CD4067B	RCA	Analog Mux 16-1	1.41
IC12	CD4067B	RCA	Analog Mux 16-1	1.41
IC13	CD4067B	RCA	Analog Mux 16-1	1.41
IC14	74LS373	TI	8-Bit Latch	1.70
IC15	74LS74	TI	Dual D-Flip Flop	0.30
IC16	74LS02	TI	2-Input NOR	0.29
IC17	74LS139	TI	Dual 2-4 Decoder	1.06
IC18	74C923	National	Keyboard Encoder	5.75
IC19	COM9026	SMC	Network Controller	95.00
IC20	COM9032	SMC	Network Transceiver	23.00
IC21	EG-A059101A	Zenith	Network Driver	9.10
IC22	HM6264P-2	Hitachi	8Kx8 Static Ram	41.50
IC23	74LS00	TI	2-Input NAND	0.26
IC24	7406	TI	Hex Inverter w/OC	0.27
IC25	74LS244	TI	8-Bit Buffer	0.84
IC26	74LS373	TI	8-Bit Latch	1.70
IC27	74LS373	TI	8-Bit Latch	1.70
IC28	74LS373	TI	8-Bit Latch	1.70
IC29	74LS367	TI	6-Bit Buffer	0.43
IC30	74LS166	TI	8-Bit Shift Register	1.75
DISP1	1864	AND	40 Char. LCD Display	103.00
KEY1	88BB2-082	Grayhill	16-Key Keyboard	11.66
XTAL1			8 MHz Crystal	2.50
XTAL2	XO-43B	Dale	20 MHz Oscillator Pkg.	10.00
SW1			SPDT Pushbutton	2.00
SW2			8-Bit SPST DIP	2.00
			TOTAL COST:	$ 506.12

3.5 Hardware Development Procedure

The monitoring unit prototype was built by starting with the absolute minimum system, which consisted of the 80188 processor and the 2764-4 EPROM. First, this pair was developed to the point of verifying correct clock and timing signals and memory access cycles. The only software in the EPROM was an infinite loop. Then, components were successively added, each tested so that basic operation of the component was verified, until the system was complete. The progression, after the processor and EPROM were verified, continued with the rest of the memory, the parallel port, the keyboard and display, the A/D converter circuitry, and finally the local area network interface.

The above developmental procedure eased the task of debugging a fairly complex digital system by breaking the task into subtasks. Of course, problems with previously "verified" components did occur, but overall the hardware development methodology worked well. A logic analyzer and oscilloscope were available, but an in-circuit emulator for the 80188 would have made the development process much easier. This more professional approach would, however, come at a great cost in comparison with the methodology used.

3.6 ARCNET-PC Local Area Network Controller [20]

The Standard Microsystems' ARCNET-PC board provides an interface between the IBM Personal Computer bus and an ARNCET

local area network. The board, which plugs into the IBM PC backplane, incorporates the SMC COM9026 Local Area Network Controller and the COM9032 Local Area Network Transceiver chips. The circuitry equivalent to the Zenith Local Area Network Driver hybrid is also contained on the board. The board also contains 2K bytes of buffer RAM, on-board Intel 8253 Programmable Interval Timer, and room for 8K bytes of PROM and 2K bytes of RAM. The board costs $550.

In the monitoring system design, this board is used to allow the IBM PC to communicate with the monitoring units over a coaxial cable. This board provided a good off-the-shelf solution to the problem of adding ARCNET capability to a centralized data collector (the IBM PC). The board is software controlled by a program written in Advanced BASIC.

Although the documentation provided with the board is somewhat sparse, the board is fairly simple to install and to write driver software for. The ARCNET-PC board proved to be a vital component of the monitoring system.

CHAPTER 4

SOFTWARE

4.1 Software Overview

The monitoring system consists of two major pieces of software. First is the software written in 80188 assembly language and is programmed into the EPROM. This program controls all of the hardware for a given monitoring unit. Second, there is the software that runs on the IBM PC, which is used as the central data collection unit in the monitoring system. This software consists of two programs, written in BASIC, that allow the IBM PC to perform two functions. One program receives one of three types of messages from a given monitoring unit: the unit is reporting a sensor value either exceeding an upper threshold, falling below a lower threshold, or back to normal range after previously being out of range. The program receives the message, timestamps it, and displays it along with the sending monitoring unit's ID number. The second BASIC program on the IBM PC makes the PC act as a remote "virtual keyboard." This allows an operator to perform all of the functions normally accessed through the keyboard and LCD display on a monitoring unit to be performed remotely.

This chapter discusses language selection issues, software

development methodology, and the structure of the three programs mentioned above. Listings of the three programs are contained in Appendices A, B, and C.

4.2 Language Selection [21],[22]

The monitoring unit, consisting of a large number of high-integration components, as discussed in Chapter 3, requires a large amount of software support. This support includes interrupt handling, bit-intensive operations, I/O intensive operations, and manipulation of fixed location addresses. The program should also be small, so the process of programming a new EPROM during development can be done quickly. These requirements indicate that assembly language should be used. On the other hand, the program is fairly large and complex and will be changed to adapt it for other applications. These requirements indicate that a high-level language should be used.

Upon examination of both sets of requirements, an ideal solution to the language selection problem apears to be a combination of high-level language and assembly language. There are, however, some serious drawbacks to this approach for this design. First, the interface of high-level and assembly languages, while possible, is complex. The code produced by a high-level language may be dependent on a particular operating system and configuration. Input and output facilities are usually a large concern. System utilities and subroutines must be written to interface the high-level language code with the

assembly language code, and this detracts from the original reason for using a high-level language. Second, a high-level language produces much overhead code. For example, a simple loop written in C takes up about 10K bytes of memory. More code means more problems when that code is controlling all aspects of a system. If a problem occurs, the code generated by the high-level language must be examined to see what is really executed, which again detracts from the original goal. These factors combined with my own previous skills at assembly language programming led to the decision to implement all of the monitoring unit software in 80188 (8088) assembly language. This project also unveils the need for high-level language support or assembly language interface techniques that are better than the current state-of-the-art.

The selection criteria for the software that runs on the IBM PC are not so stringent. Although the ARCNET-PC Board (which provides the communication link between the IBM PC and the other monitoring units through ARCNET) does have interrupt capability, it is not used. A high-level language can be used since interrupts do not have to be processed. These programs are fairly simple, prone to change, and do not require high-speed. To fill these requirements, Advanced BASIC for the IBM PC was chosen as the high-level language.

4.3 Software Development Methodology [3],[21],[23]

This section describes the exact procedures used to develop

the 80188 assembly language program that controls the monitoring unit. It begins with an overview of the development process.

The assembly language program is written and assembled on the IBM PC. The object code is used by the linker to produce an executable object file. This object file is then converted to a binary image file and given an absolute starting address, which is the beginning of the EPROM in the monitoring unit. This file contains the actual machine language needed for the monitoring unit. This file is then loaded into the IBM PC memory, and an EPROM is programmed. The EPROM is then placed in the monitoring unit and the program is tested. This loop is repeated until the final code is produced.

The 80188 architecture makes use of segment registers, but when fixed memory locations and EPROM locations are needed, some special considerations are required. Figure 4-1 shows the skeleton structure of the assembly language program segment structure used to obtain the desired result. The segment memory maps are shown in Figure 4-2. First, the code segment is defined as in any other assembly program, but the code must be preceded by an "org" statement to specify at what offset from the code segment base (specified during the EXE2BIN operation) the code will appear in the monitoring unit memory map. The restart location is fixed at the top 16 bytes of memory with this technique. The stack, data, and extra segments are defined using the "segment para at x" qualifier. This allows all references to items in these segments to know of the actual physical address (x0) used in the monitoring unit memory map. Note that the stack segment appears like the other segments. During the link process

```
STACK_seg       segment para at 01C0h
                dw      512 dup (?)

top_stack       equ     this word
STACK_seg       ends


DATA_seg        segment para at 0000h
                  .
                  .
DATA_seg        ends


EXTRA_seg       segment para at 0200h
                  .
                  .
EXTRA_seg       ends


CODE_SEG        segment para 'code'
                assume  CS:CODE_SEG,DS:DATA_seg,SS:STACK_seg
                assume  ES:EXTRA_seg

                mov     AX,DATA_seg
                mov     DS,AX

                mov     AX,EXTRA_seg
                mov     ES,AX

                mov     AX,STACK_seg
                mov     SS,AX

                mov     SP,offset top_stack
                  .
                  .
CODE_SEG        ends
```

Figure 4-1. Skeleton segment structure.

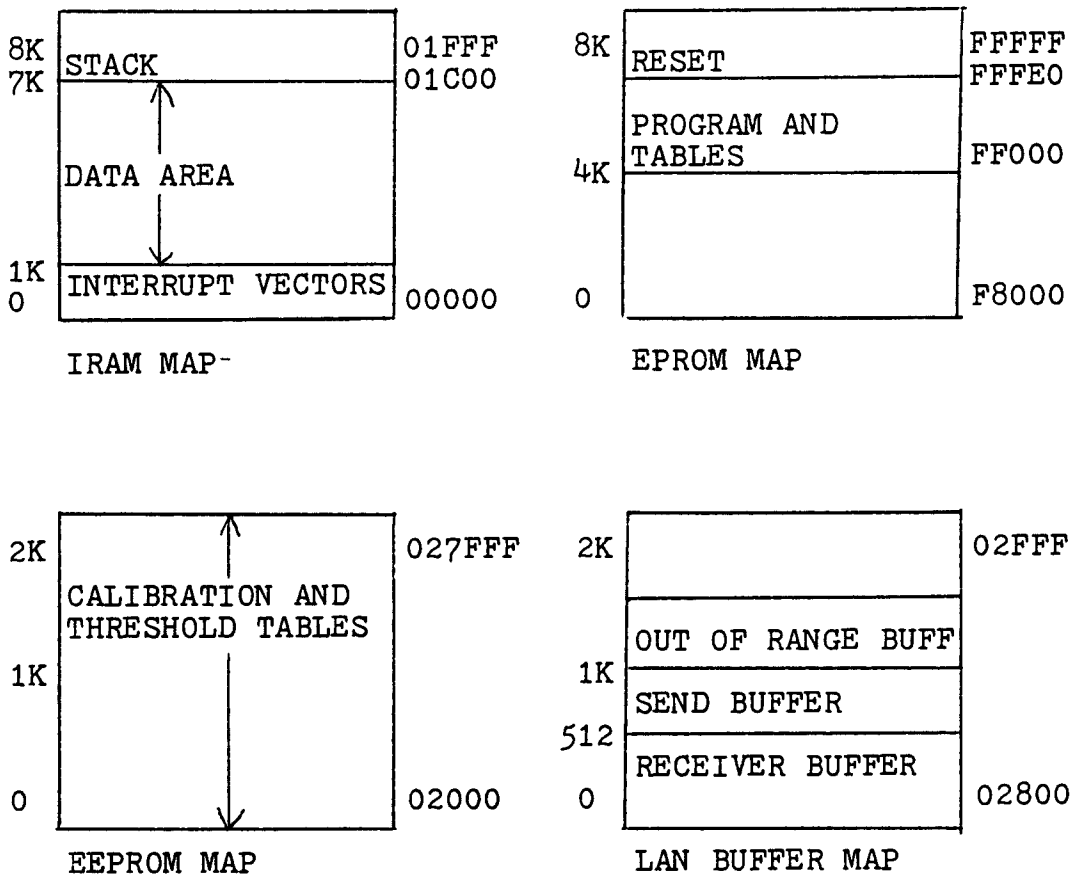


Figure 4-2. Segment memory maps.

(explained below), the linker looks for a segment designated as "stack", and does not find one. The linker then gives an error message, which is not harmful for this type of procedure.

The segment names serve two purposes. First, they allow the code segment to know of the actual physical address of the segments in the monitoring unit memory map. Secondly, the segment names, through their association with the "assmume" statement in the code segment, determine whether overide prefixes are needed for particular memory references. The segment names are not used to signal the existence of certain segments (such as the stack segment) to utilities such as the linker. Assembly language programs meant to run on the IBM PC normally use this identification feature to allow the operating system to manage the segment relocation. The code segment starting address is defined after the link process (by the DOS EXE2BIN facility), so all relative references can be resolved to produce a binary image of the final code. The relocation advantages of the 80188 architecture turn into disadvantages when absolute addresses must be used for many of the monitoring unit memory map references.

The following is a detailed description of the procedures involved. Figure 4-3 shows the file creation path. The source file is entered on the IBM PC, using a text editor (IBM Professional Editor was used). This file is called THESIS.ASM. This file is then assembled using the IBM PC Macro Assembler, Version 1.00. Version 2.00 was tried, but failed to produce the correct code for the first jump instruction. The actual assembler command is "MASM THESIS,,,;". This produces an object file, THESIS.OBJ, a listing file, THESIS.LST, and a file for the

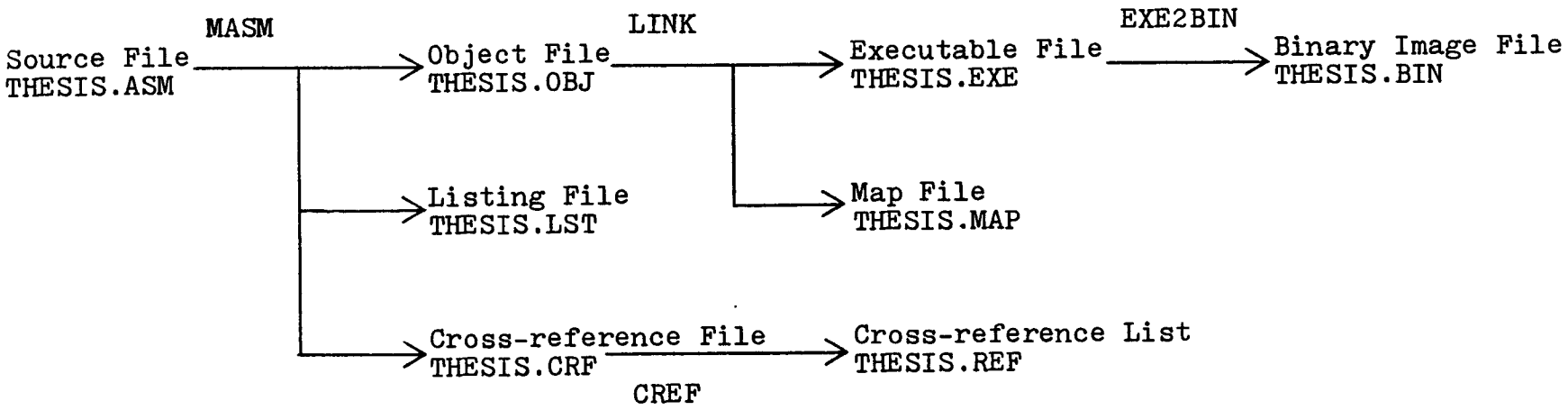


Figure 4-3. File creation path.

cross reference facility, THESIS.CRF. A cross-reference listing is produced by using the IBM PC DOS facility "CREF THESIS;". This produces a reference file called "THESIS.REF". The linker is run using the command "LINK THESIS,,;". This produces an "executable" file, THESIS.EXE (which could be relocated and run if the code were written for the IBM PC and not the monitoring unit), and a map of the link process, THESIS.MAP. An error message indicating that a stack segment was not specified is normal. The THESIS.EXE file is then converted to a binary image file with the IBM PC DOS utility "EXE2BIN THESIS". To produce a binary image file, an absolute starting address is required, and FF00 is specified for this utility. This is the first EPROM paragraph address in the last 4K-byte portion of the monitoring unit memory map. This file contains the code ready to run on the monitoring unit.

The next phase involves getting the code from the THESIS.BIN file into an EPROM. A Tecmar Incorporated PC-Mate E+EEPROM Programmer was used to accomplish this. The board, which fits inside the IBM PC, comes with a menu driven software support package. First, the binary image file must be read into a free portion of RAM in the PC. An EPROM programmer command, PEXEC, allocates a 32K-byte buffer. The starting address of this buffer can be found from one of the menu driven functions (assume it is EBB:0). The binary file, THESIS.BIN, is read into this area using the IBM PC DOS Debug facility, which is invoked by the command "DEBUG". The name of the file is specified by "N THESIS.BIN", and the load is accomplished with "L EBB:0". Debug is terminated with "Q". The EPROM sofware package is entered

with the command "MENU". A copy operation is specified, from memory into EPROM, with a source starting address of 0 and a length of 4096, because the current monitoring unit software is written for the last half (4K-byte block) of the 8K-byte EPROM. Next, the destination starting address is specified which, for our configuration, is H3000. In general, this address is the middle of the location of the EPROM socket to be programmed in the IBM PC memory map. The programming commences and when finished, the EPROM is removed and placed in the monitoring unit.

From the time a source file is created, it takes about 15 minutes to produce an EPROM using the above procedure. Some time is saved by not producing the listing and cross-reference files for intermediate runs.

This software development methodology is both crude and inefficient, but it is inexpensive and uses readily available equipment. All that is needed is an IBM PC (a PC-XT helps because of the higher hard disk access rate), a text editor, an assembler, an EPROM programmer, and a supply of EPROMs. Neither a processor specific development system nor resident development software is required. This equipment cost and availability benefit, however, come at a cost. Program changes necessitate programming a new EPROM, which is time-consuming. The smallest change requires at least 15 minutes of activity. This process may be improved by sending binary programs to the monitoring unit RAM (or EEPROM, but this is expensive) over a serial line (or ARCNET) for each program load. A bootstrap routine would be contained in the monitoring unit EPROM. This would eliminate the time required to program the EPROM, by about 5 minutes. This approach would

add a good deal of complexity for a small increase in speed.

The greatest disadvantage to this methodology is the lack of debugging tools. The only way to observe intermediate values during a program execution is to physically program outputs to the LCD display. This difficulty is compounded with the four interrupt sources. To help overcome this difficulty, the software was developed and tested in small sections, confining the problems (usually) to a manageable size. This approach is very crude and risky, but it worked! A thorough knowledge of the hardware in the system as well as of the software aspects of the processor is essential to system development using this methodology.

4.4 Monitoring Unit Software

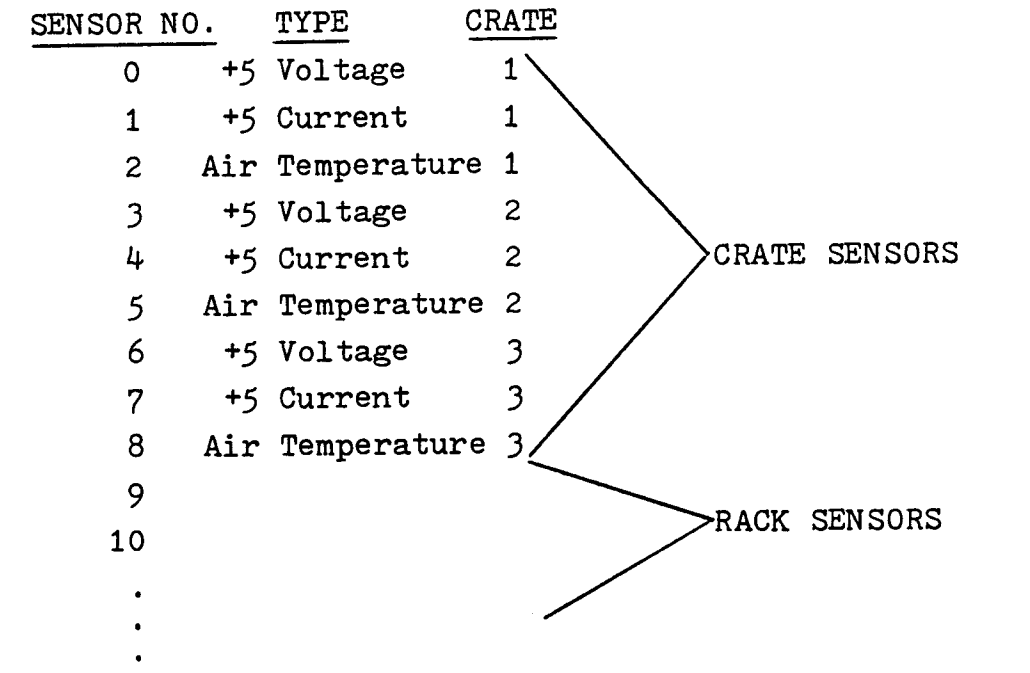
The monitoring unit program is contained in Appendix A. It consists of hardware initialization, a main background loop that scans all of the sensors for out of threshold conditions, and four main interrupt handlers and their associated procedures. If an out-of-threshold condition is discovered, or a previously out-of-threshold condition is alleviated, a message is sent to the IBM PC. The first interrupt handler is called whenever a key is pressed. Depending on the current mode of operation, the key is interperted as a menu selection, digit value entry, display of all out of threshold sensors, or display reset. The second interrupt handler is called when the analog-to-digital converter has completed a conversion. The converted value is then read

from the ADC. The third interrupt handler is called when the EEPROM has completed a write cycle, and a status flag is set to indicate the EEPROM is available again. The fourth interrupt handler is activated when a request message from the IBM PC is received. The requested operation is performed, and a response message may be returned to the IBM PC. The network handler uses many of the same functions used by the keyboard handler, because all of the keyboard funtions can be carried out remotely over ARCNET by the IBM PC.

Most of the software is table-driven, allowing menu screens and functions to be added easily. The software routines are structured into procedures, and new functions can be added by writing additional procedures. There are a few unusual considerations that need explanation. Every write to the EEPROM must be a byte write, which means that a word write must be broken down into two byte writes.

The sensors are numbered from 0 to 63, the maximum number of sensor inputs. The sensors fall into one of two major categories: a sensor can be associated with a given crate, from 1 to 3, or with the rack as a whole. As an example, a +5 VDC sensor might be used in each crate, while a smoke sensor might be used for the entire instrumentation rack. The sensor map arangement and the equations for relating sensor number with sensor type and crate are shown in Figure 4-4. The sensor numbers are virtual, and are translated to real sensor input pin addresses in the ADC read procedure. This allows easier printed circuit board layout, leaving the mapping to a software table.

Each sensor reading comes from the ADC as an 8-bit binary



KIND= 0 +5 Voltage
1 +5 Current
2 Air Temperature

SENSORS/CRATE= 3

CRATE= 1 Top Crate
2 Middle Crate
3 Bottom Crate

CRATE SENSOR NUMBER= (CRATE-1)*SENSORS/CRATE + KIND

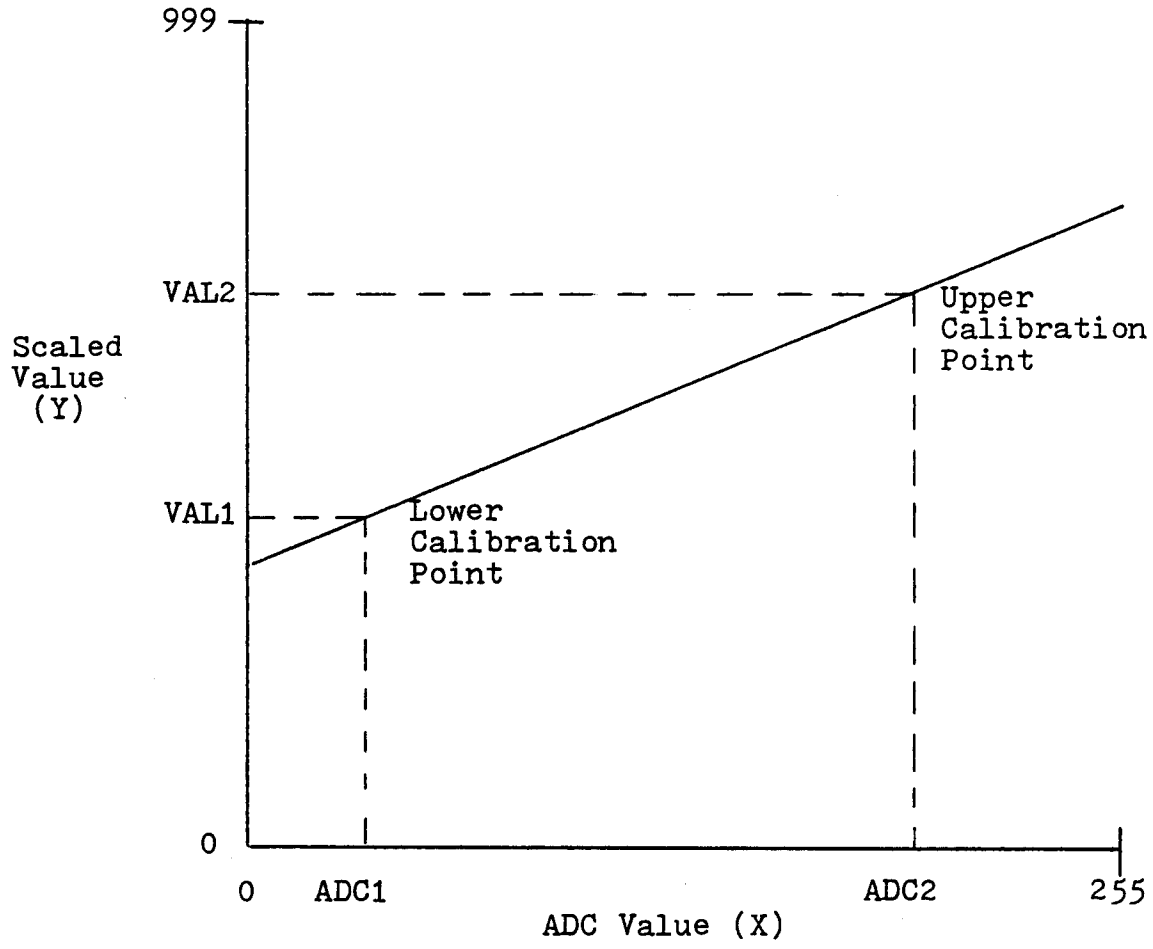
RACK SENSOR NUMBER= 2*SENSORS/CRATE + KIND

Figure 4-4. Sensor map.

value. This is the form that is stored in the threshold tables for comparison to the current sensor values, but the user enters and sees a scaled sensor value. This value is obtained from the sensor 8-bit value by linear interpolation-extrapolation from two calibration points. This process is shown in Figure 4-5. To calibrate, an upper point condition is created and the sensor reading at that point is defined as a given three decimal digit number. A lower condition is then created amd the sensor reading at that new point is defined by another three decimal digit number. From these four values, the two 8-bit sensor values and the 2 three decimal digit values, three constants (A, B, and C) are precomputed. These values are used for finding a three digit scaled number that corresponds to a given 8-bit sensor value, or finding an 8-bit sensor value that corresponds to a given three digit number. All of the sensors must be calibrated before meaningful values are obtained. Note that the A, B, and C values can be precomputed for each sensor at initialization time and stored in RAM or precomputed once and stored in EEPROM for greater computation efficiency. This is not implemented in the prototype because the program already runs very quickly. The A, B, and C are precomputed only for conceptual reasons.

4.5 IBM PC Software

The first program for the IBM PC, listed in Appendix B, merely waits for a message from the monitoring units. The message will indicate that a sensor has exceeded its upper



$$X=\frac{(Y \times C) - B}{A} \qquad Y=\frac{(X \times A) - B}{C}$$

Where:
A=(VAL2 - VAL1)
B=(ADC2 x VAL1) - (ADC1 x VAL2)
C=(ADC2 - ADC1)

Figure 4-5. Calibration relationships.

threshold (indicated by an up arrow), gone below its lower threshold (indicated by a down arrow), or is back in range after previously being out-of-range (indicated by a happy face). The message uses the same parameter type, crate number, and value format as in the LCD display. In addition to this message, a timestamp and the sending station ID are also shown.

The second program for the IBM PC, listed in Appendix C, allows a menu-driven interface to the ARCNET line so that all of the functions that are available via a monitoring unit keyboard are also available via the remotely connected IBM PC. The program formats a request message from the user's menu selections and awaits a response, which uses the same format shown in the LCD display.

CHAPTER 5

LOCAL AREA NETWORK SELECTION

5.1 Local Area Network Overview [24]

The local area network aspects of the monitoring system are very interesting in their own right. One of the requirements of the monitoring system design is to allow communication between each of the monitoring units and the IBM PC and from the IBM PC. The first type of communication scheme considered used a standard serial communications chip with an RS-422 interface to a twisted-pair cable. All of the monitoring units would be in a receive mode until a unit received a "token" from the IBM PC giving it permission to transmit a message back to the PC. A bus configuration, rather than a ring, would provide the best reliability and durability, but would require tapping of the twisted-pair cable. The advantage of this scheme is cost. Serial communication chips are readily available and inexpensive; Intel's 8251A Programmable Communication Interface costs about $4.00. Twisted-pair cable is also available and inexpensive. The major disadvantage with this approach is the added complexity of inventing yet another local area network protocol. An alternative was to use one of the off-the-shelf local area network controller chips available and use a standard network.

The two types of standard networks considered were Ethernet and ARCNET, because these are the only two networks that are available in high-integration controller chips. The Ethernet controller is produced by several manufacturers, including Intel, and the ARCNET controller is produced by Standard Microsystems Corporation. The controller chips handle most of the protocol requirements, allowing a fairly simple interface from a conceptual and software point of view. The controllers also interface with transceivers and line drivers that handle the electrical requirements of the network specifications. The availability of these chips greatly eases the burden of adding network capability to a digital system.

Although both types of network controllers are fairly evenly matched in terms of cost, availability, and the number of other components needed to interface the controller to a microprocessor, the transceiver and cable costs are very different. Ethernet requires a transceiver that is not only large, but also expensive (about $250). In addition, Ethernet requires special coaxial cable. These costs, combined with the other component costs, result in the network capability approaching the cost of all of the other monitoring unit components combined! In comparison, the ARCNET transceiver consists of a $23.00 chip, a $9.00 hybrid, and a BNC connector. The cable consists of some standard 93-ohm RG-62U cable.

There are, of course, some differences in capabilities. Ethernet has a data rate of 10 megabits/sec, compared to the ARCNET rate of 2.5 megabits/sec. The maximum station separation for Ethernet is 2.5 Km, compared to 2000 ft (more if hubs are

used) for ARCNET. Ethernet can have up to 1024 stations, while ARCNET can support 255 stations. There are also some similarities. Both use base-band signalling, bus or tree topologies, and have broadcast capabilities. The monitoring system requirements for speed, number of stations, and distance are very minimal. ARCNET satisfies all of the requirements, and still leaves a great deal of room for expansion.

An alternative network, called Cheapernet, has recently been developed. This network is an Ethernet-like system that uses standard RG-58 coaxial cable. Of course, Cheapernet does not meet all of the Ethernet specifications. While there is no off-the-shelf integrated controller available, this may be a good, future alternative network.

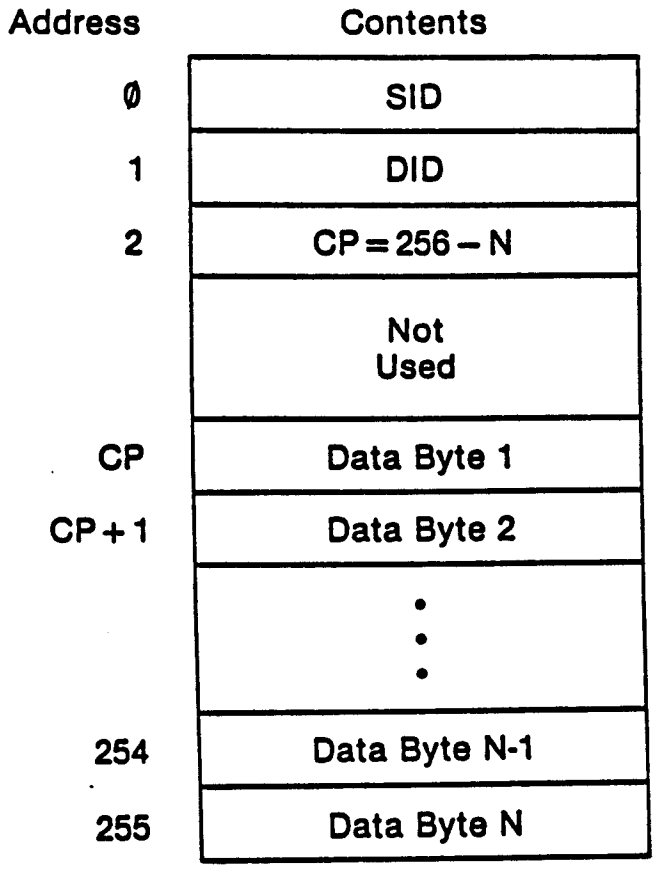
5.2 ARCNET Description [17],[25],[26]

The ARCNET local area network has been used by Datapoint corporation for over ten years. It was originally designed to provide resource sharing and modular expansion for multiple processor systems. It has, however, proved to be of great use for general local area networks. ARCNET is a token passing system. Control is completely dispersed among the nodes of the network, so token detection and re-creation of lost tokens are not dependent on a centralized node. The system is dynamically configurable, allowing nodes to enter and leave the network without major impact. Even rotation of the token and maximum time allotments for each node mean the transmission delay is

deterministic, as opposed to Ethernet's statistical contention resolution protocol. This allegedly makes ARCNET ideal for control and monitoring systems. In practice, however, Ethernet only behaves non-deterministically when it is heavily loaded, which is very difficult to accomplish.

The ARCNET protocol imposes few restrictions on the transmission medium. The receiver and transmitter must be connected by a single path, and the path must have a propagation delay of 31 microseconds or less. The Standard ARCNET interface specifies the RG-62U coaxial cable with dipulse signalling, but fiber optic or twisted pair media could also be used. Active or passive connections called "hubs" ensure that the coaxial cable is properly terminated at each end. These hubs have no insertion loss and no tap loss, yet suppress reflections, even from unterminated lines. Hubs were not used in the monitoring system design because only one prototype monitoring unit and a PC were used, allowing direct connection.

Messages in ARCNET consist of 256 byte packets. The message buffer format is shown in Figure 5-1. The user need only place a packet in a RAM buffer and enable transmit. No other attention to the protocol is needed. The controller chip handles free buffer enquiries, invitations to transmit, data packet transmission, acknowledgements and negative acknowledgements, including CRC checking. Reconfiguration is also automatically handled. This arrangement illustrates how functional layering really reduces design time.



N = Data Length
SID = Source ID (Not Used For Transmit Buffers)
DID = Destination ID (Ø For Broadcasts)

Figure 5-1. Message buffer format.

5.3 Network Experiences

The ARCNET interface worked almost perfectly the first time, which demonstrates the ease of design and robustness of the system. An important piece of the ARCNET system is the ARCNET-PC board which, as discussed earlier, provided at least one known operational station. The only problem encountered was a reflection-like symptom that prevented the IBM PC from receiving acknowledgements from the monitoring unit. This was resolved by further decoupling of the hybrid power supply lines and installation of an R-C decoupling network that Zenith suggests in their application note. ARCNET proved to be both interesting in its characteristics and practical in its implementation, which is usually something difficult to achieve.

CHAPTER 6

PROTOTYPE OPERATION

This chapter describes the prototype operation. The keyboard layout and display format are shown in Figure 6-1. After a power on reset, the upper line of the LCD display shows the network station ID (which can be changed with the DIP switch) and a continuous display of the number of sensors currently exceeding their threshold levels. The lower line of the display shows the selection menu. At any point, key A can be pressed to scroll through the list of sensors that are outside their threshold regions. Key C resets the menu selection to the first screen. Menu items are selected by pressing the underlined digit that corresponds to the desired selection. A right arrow in the rightmost position of the display indicates that there are more items in the menu, and these are displayed with the scroll key, key F. Key E controls backwards scrolling through menu levels.

The menu hierarchy is shown in Figure 6-2. The menu selections allow a user to select sensor type, crate location, and desired operation. The optional operations include displaying a given sensor value, displaying and setting upper or lower thresholds, and displaying and setting upper or lower calibration points. Individual sensors may also be disabled or enabled from the sensor scan.

KEYBOARD LAYOUT

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

KEY	PURPOSE
0-9	Menu Selection and Digit Entry
A	Scroll Out-of-Threshold List
B	Not Used
C	Reset to Menu Beginning
D	Not Used
E	Backward Scroll
F	Horizontal Scroll

ID=015 07TH2 12TH1
+5 Volt (1)= 4.87V

SAMPLE DISPLAY FORMAT

Figure 6-1. Keyboard layout and display format.

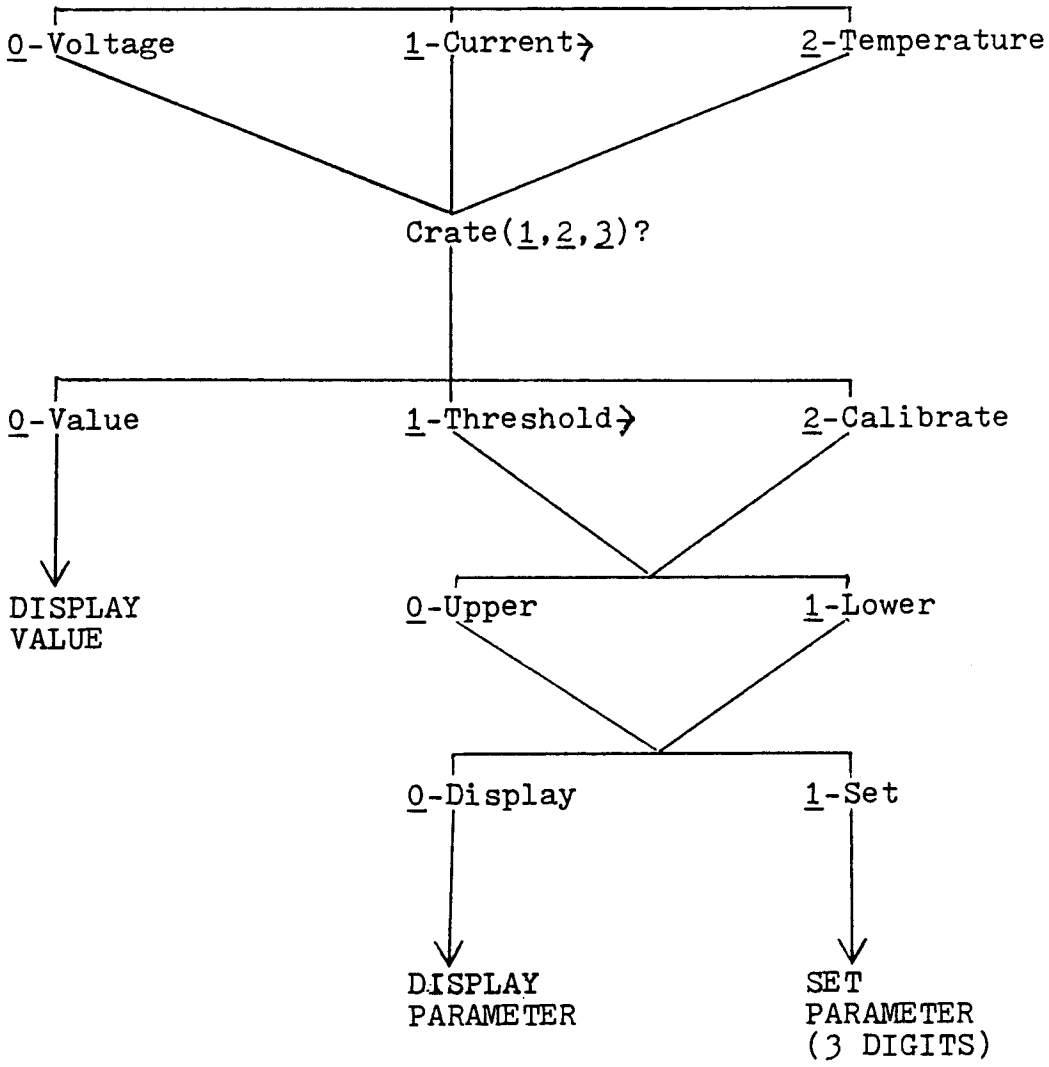


Figure 6-2. Menu hierarchy.

When any digit value entry is required, three underline prompts appear with the decimal point located in a position appropriate for a given sensor type range. Three decimal digits can then be entered. The parameter is set after the last digit is pressed. When any value is viewed, the sensor type is displayed along with a crate number in parenthesis, followed by the value and a unit. For example, "+5 Volts (1)= 4.87V" means that the +5 VDC power supply in crate number one has a value of 4.87 volts. Key C should be pressed after any entry or display to start the menu selection process again.

The IBM PC is connected to the monitoring unit prototype with a RG-62U cable that has a BNC connector on each end. Either of the two BASIC programs can be run to allow either virtual remote keyboard capability or monitoring of threshold values that are crossed. All functions can operate asynchronously.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

In conclusion, this project shows how digital systems can be designed, and even overdesigned, with current high-integration components. It also shows how local area network capability, something considered a luxury a few years ago, can be added to an existing digital system with little difficulty and small cost. There are some problems with current design methodologies. Hardware is so highly integrated and software is so highly used that better hardware and software debugging tools are required. These tools should not be tied to any particular system configuration or operating system. While this sytem design sucessfully used available, inexpensive methodologies, the methodologies described herein may fail for future, more complex digital systems.

As far as the project itself, a larger display, some newer components (such as the LAN RAM buffer), and more extensive capabilities for control as well as monitoring would enhance the current prototype's features. A single program that combines the features of the monitoring program and the virtual terminal program would allow more complete monitoring centralization. The current prototype can be altered for a wide variety of applications.

The project in this thesis can be described by an over-used word, "state-of-the-art". It will be interesting to read this thesis 10 years from now and see just how archaic this project really was. How about a complete monitoring system on one chip?? Easily, with room to spare.

APPENDIX A

MONITORING UNIT CONTROL PROGRAM

This appendix contains the 80188 assembly language program listing for the monitoring unit control program. The link map and the cross-reference listing are also included. The program was assembled using the IBM PC Macro Assembler Version 1.00.

```
1                                       page ,132
2                                       ;*****************************************************************************
3                                       ; EPROM Software for Microprocessor Controlled Monitoring System
4                                       ; Bradley S. Rubin              April 28,1985
5                                       ;
6                                       ;
7                                       ;*****************************************************************************
8                                       ;
9                                       ;
10                                      ;*****************************************************************************
11                                      ; Some global constants:
12                                      ; sensors_per_crate     number of sensors per crate
13                                      ; no_sensors    number of sensors to be monitored
14                                      ; no_types      number of types of sensors
15                                      ; undl          underline
16                                      ; u_0           underline 0
17                                      ; u_1           underline 1
18                                      ; u_2           underline 2
19                                      ; u_3           underline 3
20                                      ; arrow_r       right arrow
21                                      ; arrow_u       up arrow
22                                      ; arrow_d       down arrow
23                                      ; ok            happy face symbol
24                                      ; PC_ID         LAN station ID for IBM PC
25                                      ;*****************************************************************************
26      = 0003                          sensors_per_crate       equ     3
27      = 0009                          no_sensors      equ     9
28      = 0003                          no_types        equ     3
29      = 00FF                          undl            equ     0FFh
30      = 00B0                          u_0             equ     80h or '0'
31      = 00B1                          u_1             equ     80h or '1'
32      = 00B2                          u_2             equ     80h or '2'
33      = 00B3                          u_3             equ     80h or '3'
34      = 001D                          arrow_r         equ     1Dh
35      = 0018                          arrow_u         equ     24
36      = 0019                          arrow_d         equ     25
37      = 0001                          ok              equ     01
38      = 0002                          PC_ID           equ     02
39                                      ;
```

```
                                page
                                ;********************************************************************************
                                ; This is the definition of the stack segment area.  The stack resides
                                ; in IRAM at locations 01C0:0000 to 01C0:03FF (1024 Bytes).  This
                                ; region is the upper 1K bytes of the 8K byte IRAM.
                                ;
                                ; Variable definitions:
                                ; STACK_seg     defines stack boundries
                                ; top_stack     top of stack for pointer initialization
                                ;********************************************************************************
0000                            STACK_seg       segment para at 01C0h
0000  0200 [                                     dw      512 dup (?)     ;reserve 512 words for stack
                ????
                     ]
= 0400                          top_stack       equ     this word
0400                            STACK_seg       ends
                                ;
```

```
                                page
                                ;===============================================================================
                                ; This is the definition of the data segment area.  The data area
                                ; resides in IRAM at locations 0000:0000 to 0000:1BFF (7168 Bytes). This
                                ; region is the lower 7K bytes of the 8K byte IRAM.
                                ;
                                ; Variable definitions:
                                ; DATA_seg      defines data segment boundries
                                ; div0_vect     Divide by zero interrupt vector (type 0)
                                ; key_vect      INT0 interrupt vector (type 12) for keyboard service
                                ; adc_vect      INT1 interrupt vector (type 13) for ADC service
                                ; ee_vect       INT2 interrupt vector (type 14) for EEPROM service
                                ; net_vect      INT3 interrupt vector (type 15) for network service
                                ; adc_port      I/O port address for a given sensor number
                                ; value         current value of the A/D conversion
                                ; ready         ready=0 ADC is doing conversion
                                ;               ready=1 ADC is ready to be read
                                ; key_val       value of key pressed (0-F)
                                ; dig_entry     dig_entry=0  not digit entry mode
                                ; dig_cnt       count of digits entered for set
                                ; screen_no     next menu number to be displayed
                                ; symbol        last character in threshold message
                                ; net           net=0 display in LCD display
                                ;               net=1 put in send_buf  net=2 put in out_buf
                                ; show_count    next sensor number for out_of_limit display
                                ; out_count     number of sensors out of threshold 2
                                ; sensor        current sensor number (from 0 to no_sensors)
                                ; kind          kind of sensor  kind=0 +5 Volts  kind=1 +5 Current
                                ;                               kind=2 Air Temperature
                                ; crate_no      crate number (from 1 to 3)
                                ; param         parameter type  param=1 upper thresh 2
                                ;               param=2 lower thresh 2  param=3 upper thresh 1
                                ;               param=4 lower thresh 1  param=5 lower calibration
                                ;               param=6 lower calibration
                                ; entry         entered value (0-999)
                                ; a,b,c         precomputed coefficients for scale calculation
                                ; x             8-bit value corresp. to y
                                ; y             16-bit scaled value corresp. to x
                                ; ee_ready      ee_ready=0 busy  ee_ready=1 ready for write
                                ; sensor_tbl    sensor status  bit0=1 value outside thresh limits
                                ;                              bit1=1 sensor disabled
                                ;===============================================================================
0000                            DATA_seg        segment para at 00000h
                                ;
0000                                            org     00h             ;type 00 interrupt vector
0000  ????                      div0_vect       dw      ?
0030                                            org     30h             ;type 12 interrupt vector
0030  ????                      key_vect        dw      ?
0034                                            org     34h             ;type 13 interrupt vector
0034  ????                      adc_vect        dw      ?
0038                                            org     38h             ;type 14 interrupt vector
0038  ????                      ee_vect         dw      ?
003C                                            org     3Ch             ;type 15 interrupt vector
```

```
111     003C  ????                    net_vect        dw      ?
112                                    ;
113     0400                           org     0400h           ;data area after first 1K bytes
114     0400  ????                    adc_port        dw      ?
115     0402  ??                      value           db      ?
116     0403  ??                      ready           db      ?
117     0404  ??                      key_val         db      ?
118     0405  ????                    dig_entry       dw      ?
119     0407  ??                      dig_cnt         db      ?
120     0408  ????                    screen_no       dw      ?
121     040A  ??                      symbol          db      ?
122     040B  ??                      net             db      ?
123     040C  ????                    show_count      dw      ?
124     040E  ??                      out_count       db      ?
125     040F  ????                    sensor          dw      ?
126     0411  ????                    kind            dw      ?
127     0413  ??                      crate_no        db      ?
128     0414  ????                    param           dw      ?
129     0416  ????                    entry           dw      ?
130     0418  ????                    a               dw      ?
131     041A  ????                    b               dw      ?
132     041C  ????                                    dw      ?
133     041E  ??                      c               db      ?
134     041F  ??                      x               db      ?
135     0420  ????                    y               dw      ?
136     0422  ??                      ee_ready        db      ?
137     0423    09 [                  sensor_tbl      db      no_sensors dup (?)
138                ??
139                    ]
140
141     042C                          DATA_seg        ends
142                                   ;
```

```
                                page
                                ;================================================================================
                                ; This is the definition of the extra segment area.  The first 2K region
                                ; resides in EEPROM at locations 0200:0000 to 0200:07FF.
                                ; This region is used to hold the threshold table.  The second 2K region
                                ; resides in RAM at locations 0200:0800 to 0200:0FFF.
                                ; This region is the LAN message buffer.
                                ;
                                ; Variable definitions:
                                ; EXTRA_seg     defines extra segment boundries
                                ; thresh_u2     upper threshold 2 table
                                ; thresh_u1     upper threshold 1 table
                                ; thresh_l2     lower threshold 2 table
                                ; thresh_l1     lower threshold 1 table
                                ; adc_1         first adc calibration value table
                                ; adc_2         second adc calibration value table
                                ; val_1         first scaled calibration value table
                                ; val_2         second scaled calibration value table
                                ; rec_buf       LAN buffer for recieved requests
                                ; send_buf      LAN buffer for response transmission
                                ; out_buf       LAN buffer for threshold message transmission
                                ;================================================================================
0000                            EXTRA_seg       segment para at 00200h  ;beginning of 2K EEPROM
0000  09 [                      thresh_u2       db      no_sensors dup (?)
            ??
               ]

0009  09 [                      thresh_u1       db      no_sensors dup (?)
            ??
               ]

0012  09 [                      thresh_l1       db      no_sensors dup (?)
            ??
               ]

001B  09 [                      thresh_l2       db      no_sensors dup (?)
            ??
               ]

0024  09 [                      adc_1           db      no_sensors dup (?)
            ??
               ]

002D  09 [                      adc_2           db      no_sensors dup (?)
            ??
               ]

0036  09 [                      val_1           dw      no_sensors dup (?)
            ????
                  ]

0048  09 [                      val_2           dw      no_sensors dup (?)
            ????
```

```
                        ]
                                        ;
0800                                    org     800h            ;beginning of 2K network memory
0800  0200 [            rec_buf         db      512 dup (?)
              ??
                    ]
0A00  0200 [            send_buf        db      512 dup (?)
              ??
                    ]
0C00  0200 [            out_buf         db      512 dup (?)
              ??
                    ]
0E00                    EXTRA_seg       ends
                        ;
```

```
                        page
                        ;===============================================================================
                        ;This is the code segment area.
                        ;
                        ; Variable definitions:
                        ;
                        ; UMCS_reg      upper memory chip select I/O address
                        ; LMCS_reg      lower memory chip select I/O address
                        ; PACS_reg      peripheral address chip select I/O address
                        ; MMCS_reg      mid-range memory chip select I/O address
                        ; MPCS_reg      mid-range/peripheral chip select I/O address
                        ; EOI_reg       end of interrupt I/O adddress
                        ; INT0_reg      interrupt 0 control I/O address
                        ; INT1_reg      interrupt 1 control I/O address
                        ; INT2_reg      interrupt 2 control I/O address
                        ; INT3_reg      interrupt 3 control I/O address
                        ; PMASK_reg     interrupt priority mask I/O address
                        ;
                        ; PPORT_porta   parallel port A I/O address
                        ; PPORT_portb   parallel port B I/O address
                        ; PPORT_portc   parallel port C I/O address
                        ; PPORT_cntl    parallel port contol register I/O address
                        ; PPORT_cntlval parallel port control value
                        ;
                        ; UMCS_value    upper memory chip select value
                        ; LMCS_value    lower memory chip select value
                        ; PACS_value    periperal address chip select value
                        ; MMCS_value    mid-range memory chip select value
                        ; MPCS_value    mid-range/peripheral chip select value
                        ; EOI_value     end of interrupt value
                        ; INT0_value    interrupt 0 control value
                        ; INT1_value    interrupt 1 control value
                        ; INT2_value    interrupt 2 control value
                        ; INT3_value    interrupt 3 control value
                        ;
                        ; ADC_loc       analog to digital converter I/O address
                        ; NET_com       LANC command I/O address
                        ; NET_mask      LANC interrupt mask I/O address
                        ; NET_status    LANC status I/O address
                        ;===============================================================================
0000                    CODE_SEG        segment para 'code'
                                        assume  CS:CODE_SEG,DS:DATA_seg,SS:STACK_seg
                                        assume  ES:EXTRA_seg
                        ;
= FFA0                  UMCS_reg        equ     0FFA0h
= FFA2                  LMCS_reg        equ     0FFA2h
= FFA4                  PACS_reg        equ     0FFA4h
= FFA6                  MMCS_reg        equ     0FFA6h
= FFA8                  MPCS_reg        equ     0FFA8h
= FF22                  EOI_reg         equ     0FF22h
= FF38                  INT0_reg        equ     0FF38h
= FF3A                  INT1_reg        equ     0FF3Ah
= FF3C                  INT2_reg        equ     0FF3Ch
```

```
                        INT3_reg        equ     0FF3Eh
                        PMASK_reg       equ     0FF2Ah
                        ;
                        PPORT_porta     equ     00000h
                        PPORT_portb     equ     00001h
                        PPORT_portc     equ     00002h
                        PPORT_cntl      equ     00003h
                        PPORT_cntlval   equ     81h
                        ;
                        UMCS_value      equ     0FE3Ch
                        LMCS_value      equ     001FBh
                        PACS_value      equ     0003Bh
                        MMCS_value      equ     003FBh
                        MPCS_value      equ     081BBh
                        EOI_value       equ     08000h
                        INT0_value      equ     00003h
                        INT1_value      equ     00001h
                        INT2_value      equ     00000h
                        INT3_value      equ     00002h
                        ;
                        ADC_loc         equ     080h
                        NET_com         equ     101h
                        NET_mask        equ     100h
                        NET_status      equ     100h
                        ;
0000                    Initialize      proc    far
0FF0                                    org     0FF0h           ;POR address
0FF0  EA 0FE0 ---- R                    jmp     far ptr Jump
                        ;
0FE0                                    org     0FE0h           ;32 bytes from top
0FE0                    Jump            proc    far
0FE0  BA FFA0                           mov     DX,UMCS_reg     ;initialize upper memory CS
0FE3  B8 FE3C                           mov     AX,UMCS_value
0FE6  EF                                out     DX,AX
0FE7  EA 0000 ---- R                    jmp     far ptr Initialize      ;goto beginning of EPROM
0FEC                    Jump            endp
                        ;
                        ;
0000                                    org     0               ;beginning of EPROM
0000  BA FFA2                           mov     DX,LMCS_reg     ;initialize lower memory CS
0003  B8 01FB                           mov     AX,LMCS_value
0006  EF                                out     DX,AX
                        ;
0007  BA FFA4                           mov     DX,PACS_reg     ;initialize peripheral CS
000A  B8 003B                           mov     AX,PACS_value
000D  EF                                out     DX,AX
                        ;

000E  BA FFA6                           mov     DX,MMCS_reg     ;initialize mid memory CS
0011  B8 03FB                           mov     AX,MMCS_value
0014  EF                                out     DX,AX
                        ;
0015  BA FFA8                           mov     DX,MPCS_reg     ;initialize mid/periph CS
```

```
0018  B8 81BB                    mov   AX,MPCS_value
001B  EF                        out   DX,AX
                                ;
001C  B8  ---- R                mov   AX,DATA_seg     ;initialize data segment
001F  8E D8                     mov   DS,AX
                                ;
0021  B8  ---- R                mov   AX,EXTRA_seg    ;initialize extra segment
0024  8E C0                     mov   ES,AX
                                ;
0026  B8  ---- R                mov   AX,STACK_seg    ;initialize stack segment
0029  8E D0                     mov   SS,AX
                                ;
002B  BC 0400 R                 mov   SP,offset top_stack     ;initialize stack point.

002E  BA 0003                   mov   DX,PPORT_cntl   ;initialize parallel port
0031  B0 81                     mov   AL,PPORT_cntlval
0033  EE                        out   DX,AL
                                ;
0034  BA 0002                   mov   DX,PPORT_portc  ;toggle LANC reset line
0037  B0 08                     mov   AL,00001000b
0039  EE                        out   DX,AL
003A  E8 0434 R                 call  delay
003D  B0 00                     mov   AL,00000000b
003F  EE                        out   DX,AL
                                ;
0040  BA 0101                   mov   DX,NET_com      ;clear LANC flags
0043  B0 1E                     mov   AL,1Eh
0045  EE                        out   DX,AL
                                ;
0046  B0 04                     mov   AL,04h          ;enbable rec_buf reception
0048  EE                        out   DX,AL
                                ;
0049  BA 0100                   mov   DX,NET_mask     ;mask all but recieve interrupt
004C  B0 80                     mov   AL,10000000b
004E  EE                        out   DX,AL
                                ;
004F  BA FF38                   mov   DX,INT0_reg     ;initialize interrupt 0
0052  B8 0003                   mov   AX,INT0_value
0055  EF                        out   DX,AX
                                ;
0056  BA FF3A                   mov   DX,INT1_reg     ;initialize interrupt 1
0059  B8 0001                   mov   AX,INT1_value
005C  EF                        out   DX,AX
                                ;
005D  BA FF3C                   mov   DX,INT2_reg     ;initialize interrupt 2
0060  B8 0000                   mov   AX,INT2_value
0063  EF                        out   DX,AX
                                ;
0064  BA FF3E                   mov   DX,INT3_reg     ;initialize interrupt 3
0067  B8 0002                   mov   AX,INT3_value
006A  EF                        out   DX,AX
                                ;
006B  C7 06 0000 R 0A42 R       mov   div0_vect,offset Div0int ;setup div by 0 vector
```

```
0071  C7 06 0002 R  ---- R                mov   div0_vect+2,seg  Div0int
                                ;
0077  C7 06 0030 R 083C R                 mov   key_vect,offset Keyint  ;setup keyboard vector
007D  C7 06 0032 R  ---- R                mov   key_vect+2,seg  Keyint
                                ;
0083  C7 06 0034 R 091C R                 mov   adc_vect,offset Adcint  ;setup adc vector
0089  C7 06 0036 R  ---- R                mov   adc_vect+2,seg  Adcint
                                ;
008F  C7 06 0038 R 0935 R                 mov   ee_vect,offset EEint    ;setup eeprom vector
0095  C7 06 003A R  ---- R                mov   ee_vect+2,seg  EEint
                                ;
009B  C7 06 003C R 0946 R                 mov   net_vect,offset Netint  ;setup LAN vector
00A1  C7 06 003E R  ---- R                mov   net_vect+2,seg  Netint
                                ;
                                ;
00A7  C7 06 0408 R 0000                   mov   screen_no,0     ;show first menu
00AD  E8 07C6 R                           call  Disp_screen
                                ;
00B0  C6 06 0403 R 00                     mov   ready,0         ;initialize variables
00B5  C7 06 0405 R 0000                   mov   dig_entry,0
00BB  C6 06 0407 R 00                     mov   dig_cnt,0
00C0  C6 06 040B R 00                     mov   net,0
00C5  C6 06 040E R 00                     mov   out_count,0
00CA  C7 06 040C R 0000                   mov   show_count,0
00D0  BE 0000                             mov   SI,0
00D3  EB 15 90                            jmp   top_loop
                                ;
00D6  49 44 3D 20 20 20   top_line        db    'ID=      TH2   TH1 '  ;top line display format
      20 20 20 20 54 48
      32 20 20 20 54 48
      31 20
                                ;
00EA  8B C6               top_loop:       mov   AX,SI           ;display top line using format
00EC  86 E0                               xchg  AH,AL
00EE  2E: 8A 84 00D6 R                    mov   AL,top_line[SI]
00F3  E8 0783 R                           call  Display
00F6  46                                  inc   SI
00F7  83 FE 14                            cmp   SI,20
00FA  75 EE                               jne   top_loop
                                ;
00FC  BE 0000                             mov   SI,0              ;initialize sensor table to
00FF  C6 84 0423 R 00     clr_loop:       mov   sensor_tbl[SI],0  ; all enabled and none out
0104  46                                  inc   SI                ; of threshold
0105  83 FE 09                           cmp   SI,no_sensors
0108  75 F5                               jne   clr_loop
                                ;
010A  26: A0 0801 R                       mov   AL,rec_buf+1    ;get station ID
010E  B4 00                               mov   AH,0            ;convert to 3 ASCII digits
0110  B3 0A                               mov   BL,10           ; and display
0112  F6 FB                               idiv  BL
0114  50                                  push  AX
0115  B4 00                               mov   AH,0
0117  F6 FB                               idiv  BL
```

```
0119  50                                push    AX
011A  B4 03                             mov     AH,3
011C  0C 30                             or      AL,00110000b
011E  E8 0783 R                         call    display
0121  58                                pop     AX
0122  86 E0                             xchg    AH,AL
0124  B4 04                             mov     AH,4
0126  0C 30                             or      AL,00110000b
0128  E8 0783 R                         call    Display
012B  58                                pop     AX
012C  86 E0                             xchg    AH,AL
012E  B4 05                             mov     AH,5
0130  0C 30                             or      AL,00110000b
0132  E8 0783 R                         call    display
0135  FB                                sti                     ;enable interrupts
0136  EB 01 90                          jmp     Main            ;goto sensor scanning loop
0139                          Initialize endp
                              ;
                              ;
0139                          Main      proc    near
0139  BE 0000                          mov     SI,0            ;sensor number
013C  B1 00                             mov     CL,0            ;number of times all scanned
                              ;
013E  F6 84 0423 R 02         main_loop: test   sensor_tbl[SI],02h ;sensor enabled?
0143  75 41                             jne     next_sensor     ;if not, check next one
0145  BA FF2A                           mov     DX,PMASK_reg    ;disable keyboard and LAN
0148  B0 01                             mov     AL,01h          ; interrupts
014A  EE                                out     DX,AL
                              ;
014B  FF 36 040F R                      push    sensor          ;read sensor value
014F  89 36 040F R                      mov     sensor,SI
0153  E8 049E R                         call    Read_adc
0156  8F 06 040F R                      pop     sensor
                              ;
015A  8A 1E 0402 R                      mov     BL,value
015E  26: 3A 9C 0000 R                  cmp     BL,thresh_u2[SI]  ;value > upper thresh 2?
0163  77 3D                             ja      up_exceed       ;if so, goto upper exceeded
                              ;
0165  26: 38 9C 001B R                  cmp     thresh_l2[SI],BL  ;value < lower thresh 2?
016A  77 50                             ja      low_exceed      ;if so, goto low exceeded
                              ;
016C  F6 84 0423 R 01                   test    sensor_tbl[SI],01h ;sensor prev. out of range?
0171  74 0D                             je      re_enable       ;if not, reenable key and LAN
                              ;
0173  80 A4 0423 R FE                   and     sensor_tbl[SI],0FEh;if so, clear out of range
0178  C6 06 040A R 01                   mov     symbol,ok       ; status and create symbol
017D  E8 01D6 R                         call    Send_mess       ;send ok now message
                              ;
0180  BA FF2A                 re_enable: mov    DX,PMASK_reg    ;re-enable keyboard and LAN
0183  B0 07                             mov     AL,07h
0185  EE                                out     DX,AL
                              ;
0186  46                     next_sensor: inc   SI              ;next sensor (circularly)
```

```
0187  83 FE 09                        cmp   SI,no_sensors
018A  75 B2                          jne   main_loop
                                ;
018C  BE 0000                        mov   SI,0
018F  FE C1                          inc   CL              ;check if all scanned 64 times
0191  80 F9 3F                       cmp   CL,3Fh
0194  75 05                          jne   main_reset      ;if not, continue
                                ;
0196  E8 081C R                      call  Disp_count      ;if so, display out of thresh 2
0199  B1 00                          mov   CL,0            ; count
019B  C6 06 040E R 00      main_reset:mov   out_count,0     ;reset out of thresh 2 tally
01A0  EB 9C                          jmp   main_loop       ;scan all sensors again
                                ;
                                ;
01A2  FE 06 040E R         up_exceed: inc   out_count       ;out of thresh 2 tally
01A6  F6 84 0423 R 01                test  sensor_tbl[SI],01h ;previously out of range?
01AB  75 D3                          jne   re_enable          ;if so, continue
01AD  80 8C 0423 R 01                or    sensor_tbl[SI],01h ;if not, set out of range
01B2  C6 06 040A R 18                mov   symbol,arrow_u     ; status and create symbol
01B7  E8 01D6 R                      call  Send_mess          ;send upper exceeded message
01BA  EB C4                          jmp   re_enable          ;contiue on
                                ;
                                ;
01BC  FE 06 040E R         low_exceed:inc   out_count       ;out of thresh 2 tally
01C0  F6 84 0423 R 01                test  sensor_tbl[SI],01h ;previously out of range?
01C5  75 B9                          jne   re_enable          ;if so, continue
01C7  80 8C 0423 R 01                or    sensor_tbl[SI],01h ;if not, set out of range
01CC  C6 06 040A R 19                mov   symbol,arrow_d     ; status and create symbol
01D1  E8 01D6 R                      call  Send_mess          ;send lower exceeded message
01D4  EB AA                          jmp   re_enable          ;continue on
01D6                       Main      endp
                                ;
                                ;
01D6                       Send_mess proc  near
01D6  B4 00                          mov   AH,0            ;save variables
01D8  FF 36 040F R                   push  sensor
01DC  FF 36 0414 R                   push  param
01E0  FF 36 0411 R                   push  kind
01E4  A0 0413 R                      mov   AL,crate_no
01E7  50                             push  AX
01E8  FF 36 0418 R                   push  a
01EC  FF 36 041A R                   push  b
01F0  A0 041E R                      mov   AL,c
01F3  50                             push  AX
01F4  A0 041F R                      mov   AL,x
01F7  50                             push  AX
01F8  FF 36 0420 R                   push  y
01FC  FF 36 0416 R                   push  entry
                                ;
0200  89 36 040F R                   mov   sensor,SI
0204  E8 06C0 R                      call  Expand          ;expand sensor no. to variables
                                ;
0207  C6 06 040B R 02                mov   net,2           ;format message with sensor
```

```
0020C  E8 057C R                        call    Val_param       ; value in out_buf
020F  C6 06 040B R 00                   mov     net,0
0214  A0 040A R                         mov     AL,symbol       ;tack-on symbol to end
0217  26: A2 0CFF R                     mov     out_buf+0FFh,AL
021B  26: C6 06 0C01 R 02               mov     out_buf+1,PC_ID ;destination ID
0221  26: C6 06 0C02 R EC               mov     out_buf+2,0ECh  ;beginning of message pointer
                                ;
0227  BA 0101                           mov     DX,NET_com      ;send message in out_buf
022A  B0 13                             mov     AL,13h
022C  EE                                out     DX,AL
                                ;
022D  E8 0434 R                         call    Delay           ;wait for send message
                                ;
0230  8F 06 0416 R                      pop     entry           ;restore variables
0234  8F 06 0420 R                      pop     y
0238  58                                pop     AX
0239  A2 041F R                         mov     x,AL
023C  58                                pop     AX
023D  A2 041E R                         mov     c,AL
0240  8F 06 041A R                      pop     b
0244  8F 06 0418 R                      pop     a
0248  58                                pop     AX
0249  A2 0413 R                         mov     crate_no,AL
024C  8F 06 0411 R                      pop     kind
0250  8F 06 0414 R                      pop     param
0254  8F 06 040F R                      pop     sensor
0258  C3                                ret
0259                            Send_mess       endp
                                ;
                                ;
0259  025F R                    format          dw      format_0
025B  0276 R                            dw      format_1
025D  028D R                            dw      format_2
                                ;
025F  2B 35 20 56 6F 6C         format_0        db      '+5 Volt  ( )= ',undl,'.',undl,undl,'V '
      74 20 20 28 20 29
      3D 20 FF 2E FF FF
      56 20
0273  2E 30 31                          db 2Eh,30h,31h
0276  2B 35 20 43 75 72         format_1        db      '+5 Curr  ( )= ',undl,undl,undl,'.A '
      72 20 20 28 20 29
      3D 20 FF FF FF 2E
      41 20
028A  2E 2F 30                          db 2Eh,2Fh,30h
028D  41 69 72 20 54 65         format_2        db      'Air Temp ( )= ',undl,undl,'.',undl,'C '
      6D 70 20 28 20 29
      3D 20 FF FF 2E FF
      43 20
02A1  2E 2F 31                          db 2Eh,2Fh,31h
                                ;
                                ;
02A4  02B2 R                    screen          dw      screen_0
02A6  02D8 R                            dw      screen_1
```

```
02A8  02FA R                                    dw      screen_2
02AA  0324 R                                    dw      screen_3
02AC  034A R                                    dw      screen_4
02AE  036C R                                    dw      screen_5
02B0  0392 R                                    dw      screen_6
                                        ;
02B2  B0 2D 56 6F 6C 74         screen_0        db      u_0,'-Voltage ',u_1,'-Current',arrow_r
      61 67 65 20 B1 2D
      43 75 72 72 65 6E
      74 1D
02C6  0000 0467 R 0001 0467 R                   dw      0,Class,1,Class,0Eh,Back,0Fh,Scroll,0FFFFh
      000E 0453 R 000F 043F R
      FFFF
02D8  B2 2D 54 65 6D 70         screen_1        db      u_2,'-Temperature       '
      65 72 61 74 75 72
      65 20 20 20 20 20
      20 20
02EC  0002 0467 R 000E 0453 R                   dw      2,Class,0Eh,Back,0Fh,Scroll,0FFFFh
      000F 043F R FFFF
02FA  43 72 61 74 65 20         screen_2        db      'Crate (',u_1,',',u_2,',',u_3,')?       '
      28 B1 2C B2 2C B3
      29 3F 20 20 20 20
      20 20
030E  0001 0480 R 0002 0480 R                   dw      1,Crate,2,Crate,3,Crate,0Eh,Back,0Fh,Scroll
      0003 0480 R 000E 0453 R
      000F 043F R
0322  FFFF                                      dw      0FFFFh
0324  B0 2D 56 61 6C 75         screen_3      db      u_0,'-Value ',u_1,'-Threshold',arrow_r
      65 20 B1 2D 54 68
      72 65 73 68 6F 6C
      64 1D
0338  0000 04CA R 0001 04D7 R                   dw      0,Val,1,Thresh,0Eh,Back,0Fh,Scroll,0FFFFh
      000E 0453 R 000F 043F R
      FFFF
034A  B2 2D 43 61 6C 69         screen_4        db      u_2,'-Calibrate         '
      62 72 61 74 65 20
      20 20 20 20 20 20
      20 20
035E  0002 04E4 R 000E 0453 R                   dw      2,Calib,0Eh,Back,0Fh,Scroll,0FFFFh
      000F 043F R FFFF
036C  B0 2D 55 70 70 65         screen_5        db      u_0,'-Upper ',u_1,'-Lower       '
      72 20 B1 2D 4C 6F
      77 65 72 20 20 20
      20 20
0380  0000 04F1 R 0001 04F8 R                   dw      0,Up,1,Down,0Eh,Back,0Fh,Scroll,0FFFFh
      000E 0453 R 000F 043F R
      FFFF
0392  B0 2D 44 69 73 70         screen_6        db      u_0,'-Display ',u_1,'-Set       '
      6C 61 79 20 B1 2D
      53 65 74 20 20 20
      20 20
03A6  0000 0503 R 0001 050D R                   dw      0,Disp,1,Set,0Eh,Back,0Fh,Scroll,0FFFFh
      000E 0453 R 000F 043F R
```

```
                FFFF
                                        ;
                                        ;
03B8  0001 0000 0002 0004       scroll_table    dw      1,0,2,4,3,5,6
      0003 0005 0006
                                        ;
03C6  0000 0000 0000 0000       back_table      dw      0,0,0,0,0,3,5
      0000 0003 0005
                                        ;
                                        ;
03D4  00 00 00 00 00 00         Sensor_map      db      0,0,0,0,0,0,0,0,32
      00 00 20
03DD    37 [                            db      64-9 dup (1)
            01
               ]
                                        ;
                                        ;
0414  0000 R 0000 001B R        Dig_table       dw      thresh_u2,0,thresh_l2,0
      0000
041C  0009 R 0000 0012 R                        dw      thresh_u1,0,thresh_l1,0
      0000
0424  0048 R 0001 0036 R                        dw      val_2,1,val_1,1
      0001
                                        ;
                                        ;
042C                            Hold            proc    near
042C  F6 06 0422 R FF           hold2:         test    ee_ready,0FFh   ;EEPROM finished write cycle?
0431  74 F9                                    jz      hold2           ;if not, check again
0433  C3                                       ret
0434                            Hold            endp
                                        ;
                                        ;
0434                            Delay           proc    near
0434  51                                       push    CX
0435  B9 FFFF                                  mov     CX,0FFFFh       ;count to FFFFh
0438  49                       again_0:       dec     CX
0439  E3 02                                    jcxz    delay_ret
043B  EB FB                                    jmp     again_0
043D  59                        delay_ret:     pop     CX
043E  C3                                       ret
043F                            Delay           endp
                                        ;
                                        ;
043F                            Scroll          proc    near
043F  56                                       push    SI
0440  53                                       push    BX
0441  8B 36 0408 R                             mov     SI,screen_no    ;given current screen number
0445  D1 E6                                    sal     SI,1            ;look up next screen number
0447  2E: 8B 9C 03B8 R                         mov     BX,scroll_table[SI] ;in scroll table
044C  89 1E 0408 R                             mov     screen_no,BX
0450  5B                                       pop     BX
0451  5E                                       pop     SI
```

```
0452 C3                                 ret
0453                            Scroll  endp
                                ;
                                ;
0453                            Back    proc    near
0453 56                                 push    SI
0454 53                                 push    BX
0455 8B 36 0408 R                       mov     SI,screen_no    ;given current screen number
0459 D1 E6                              sal     SI,1            ;look up previous screen no.
045B 2E: 8B 9C 03C6 R                   mov     BX,back_table[SI] ;in backwards scroll table
0460 89 1E 0408 R                       mov     screen_no,BX
0464 5B                                 pop     BX
0465 5E                                 pop     SI
0466 C3                                 ret
0467                            Back    endp
                                ;
                                ;
0467                            Class   proc    near
0467 50                                 push    AX
0468 A0 0404 R                          mov     AL,key_val      ;given sensor kind
046B B4 00                              mov     AH,0
046D A3 0411 R                          mov     kind,AX
                                ;
0470 A3 040F R                          mov     sensor,AX       ;help form sensor number
0473 83 06 040F R 06                    add     sensor,2*sensors_per_crate
0478 C7 06 0408 R 0002                  mov     screen_no,2     ;show screen 2
047E 58                                 pop     AX
047F C3                                 ret
0480                            Class   endp
                                ;
                                ;
0480                            Crate   proc    near
0480 50                                 push    AX
0481 A0 0404 R                          mov     AL,key_val      ;given crate number
0484 A2 0413 R                          mov     crate_no,AL
                                ;
0487 FE C8                              dec     AL              ;help form sensor number
0489 B4 03                              mov     AH,sensors_per_crate
048B F6 E4                              mul     AH
048D B4 00                              mov     AH,0
048F 03 06 0411 R                       add     AX,kind
0493 A3 040F R                          mov     sensor,AX
0496 C7 06 0408 R 0003                  mov     screen_no,3     ;show screen 3
049C 58                                 pop     AX
049D C3                                 ret
049E                            Crate   endp
                                ;
                                ;
049E                            Read_adc proc   near
049E 50                                 push    AX
049F 53                                 push    BX
04A0 52                                 push    DX
04A1 BB 03D4 R                          mov     BX,offset sensor_map    ;translate sensor:
```

```
744   04A4  A1 040F R                        mov   AX,sensor             ;virtual to real
745   04A7  2E: D7                           xlat  sensor_map            ;using sensor map table
746                               ;
747   04A9  05 0080                          add   AX,adc_loc            ;form sensor I/O locat.
748   04AC  8B D0                            mov   DX,AX
749   04AE  A3 0400 R                        mov   adc_port,AX
750                               ;
751   04B1  F6 06 0403 R FF       no_ready_1: test  ready,0FFh      ;adc ready?
752   04B6  75 F9                           jnz   no_ready_1      ;if not, check again
753   04B8  C6 06 0403 R 01                 mov   ready,1         ;if so, set status to not ready
754   04BD  EC                              in    AL,DX           ;latch sensor address
755   04BE  EE                              out   DX,AL           ;start conversion
756   04BF  F6 06 0403 R FF       no_ready_2: test  ready,0FFh      ;adc ready?
757   04C4  75 F9                           jnz   no_ready_2      ;if not, check again
758   04C6  5A                              pop   DX
759   04C7  5B                              pop   BX
760   04C8  58                              pop   AX
761   04C9  C3                              ret
762   04CA                        Read_adc  endp
763                               ;
764                               ;
765   04CA                        Val       proc  near
766   04CA  E8 049E R                       call  Read_adc        ;get current sensor value
767   04CD  E8 057C R                       call  Val_param       ;format message with sensor
768   04D0  C7 06 0408 R FFFF               mov   screen_no,0FFFFh; value in display
769   04D6  C3                              ret
770   04D7                        Val       endp
771                               ;
772                               ;
773   04D7                        Thresh    proc  near
774   04D7  C7 06 0414 R 0001               mov   param,1         ;form parameter
775   04DD  C7 06 0408 R 0005               mov   screen_no,5     ;show screen 5
776   04E3  C3                              ret
777   04E4                        Thresh    endp
778                               ;
779                               ;
780   04E4                        Calib     proc  near
781   04E4  C7 06 0414 R 0005               mov   param,5         ;form parameter
782   04EA  C7 06 0408 R 0005               mov   screen_no,5     ;show screen 5
783   04F0  C3                              ret
784   04F1                        Calib     endp
785                               ;
786                               ;
787   04F1                        Up        proc  near
788   04F1  C7 06 0408 R 0006               mov   screen_no,6     ;leave parameter alone and
789   04F7  C3                              ret                   ; show screen 6
790   04F8                        Up        endp
791                               ;
792                               ;
793   04F8                        Down      proc  near
794   04F8  FF 06 0414 R                    inc   param           ;form parameter
795   04FC  C7 06 0408 R 0006               mov   screen_no,6     ;show screen 6
796   0502  C3                              ret
```

```
0503                        Down        endp
                            ;
                            ;
0503                        Disp        proc    near
0503  E8 0595 R                         call    Get_param       ;show sensor parameter
0506  C7 06 0408 R FFFF                 mov     screen_no,0FFFFh
050C  C3                               ret
050D                        Disp        endp
                            ;
                            ;
050D                        Set         proc    near
050D  50                               push    AX
050E  E8 07F0 R                        call    disp_form       ;format display for digit prompt
0511  A1 0414 R                        mov     AX,param        ;set digit entry mode
0514  A3 0405 R                        mov     dig_entry,AX
0517  C7 06 0408 R FFFF                mov     screen_no,0FFFFh
051D  58                               pop     AX
051E  C3                               ret
051F                        Set         endp
                            ;
                            ;
051F                        Enable      proc    near
051F  56                               push    SI
0520  8B 36 040F R                     mov     SI,sensor               ;set sensor status to
0524  80 A4 0423 R FD                  and     sensor_tbl[SI],0FDh     ; enabled
0529  5E                               pop     SI
052A  C3                               ret
052B                        Enable      endp
                            ;
                            ;
052B                        Disable     proc    near
052B  56                               push    SI
052C  8B 36 040F R                     mov     SI,sensor               ;set sensor status to
0530  80 8C 0423 R 02                  or      sensor_tbl[SI],02h      ; disabled
0535  5E                               pop     SI
0536  C3                               ret
0537                        Disable     endp
                            ;
                            ;
0537                        Show_out    proc    near
0537  56                               push    SI
0538  8B 36 040C R                     mov     SI,show_count   ;get next sensor to check
                            ;
053C  F6 84 0423 R 02       look:       test    sensor_tbl[SI],02h      ;sensor enabled?
0541  75 26                            jne     next_show               ;if not, next one
                            ;
0543  F6 84 0423 R 01                  test    sensor_tbl[SI],01h      ;sensor out of thresh 2?
0548  74 1F                            je      next_show               ;if not, next one
                            ;
054A  89 36 040F R                     mov     sensor,SI               ;get sensor value
054E  56                               push    SI
054F  E8 049E R                        call    Read_adc
0552  E8 06C0 R                        call    Expand  ;expand sensor no. to variables
```

```
0555 E8 057C R                          call    Val_param       ;format display w/ sensor value
0558 5E                                 pop     SI
0559 46                                 inc     SI              ;save next sensor number
055A 83 FE 09                           cmp     SI,no_sensors
055D 75 03                              jne     cir_inc
055F BE 0000                            mov     SI,0
0562 89 36 040C R       cir_inc: mov    show_count,SI
0566 EB 12 90                           jmp     show_ret
                        ;
0569 46                 next_show:      inc     SI
056A 3B 36 040C R                       cmp     SI,show_count   ;none to display?
056E 74 0A                              je      show_ret        ;if so, return
                        ;
0570 83 FE 09                           cmp     SI,no_sensors   ;circular increment
0573 75 C7                              jne     look
0575 BE 0000                            mov     SI,0
0578 EB C2                              jmp     look
                        ;
057A 5E                 show_ret:       pop     SI
057B C3                                 ret
057C                    Show_out        endp
                        ;
                        ;
057C                    Val_param       proc    near
057C 50                                 push    AX
057D A0 0402 R                          mov     AL,value        ;get sensor value
0580 A2 041F R                          mov     x,AL
0583 E8 07F0 R                          call    Disp_form       ;format display
0586 E8 066C R                          call    pre_comp        ;pre-compute a,b,c
0589 E8 0760 R                          call    find_y          ;find scaled value
058C 8B 16 0420 R                       mov     DX,y
0590 E8 06ED R                          call    convert         ;convert value for display
0593 58                                 pop     AX
0594 C3                                 ret
0595                    Val_param       endp
                        ;
                        ;
0595                    Get_param       proc    near
0595 53                                 push    BX
0596 52                                 push    DX
0597 56                                 push    SI
                        ;
0598 E8 07F0 R                          call    disp_form       ;format display
                        ;
059B 8B 36 0414 R                       mov     SI,param ;index into digit table for
059F 4E                                 dec     SI              ; paramameter table name and
05A0 D1 E6                              sal     SI,1            ; size
05A2 D1 E6                              sal     SI,1
05A4 2E: 8B 9C 0414 R                   mov     BX,dig_table[SI]
05A9 2E: 8B 94 0416 R                   mov     DX,dig_table[SI+2]
05AE 8B 36 040F R                       mov     SI,sensor
05B2 85 D2                              test    Dx,Dx
05B4 74 08                              je      read_byte
```

```
                                ;
05B6  D1 E6                     sal     SI,1
05B8  26: 8B 10                 mov     DX,ES:[BX+SI]   ;read word from EEPROM
05BB  EB 12 90                  jmp     read_ret
                                ;
05BE  26: 8A 10                 read_byte:  mov     DL,ES:[BX+SI]   ;read byte from EEPROM
05C1  88 16 041F R              mov     x,DL
05C5  E8 066C R                 call    Pre_comp        ;pre-compute a,b,c
05C8  E8 0760 R                 call    Find_y          ;find scaled value
05CB  8B 16 0420 R              mov     DX,y
05CF  E8 06ED R                 read_ret:   call    convert         ;convert value for display
                                ;
05D2  5E                        pop     SI
05D3  5A                        pop     DX
05D4  5B                        pop     BX
05D5  C3                        ret
05D6                            Get_param   endp
                                ;
                                ;
05D6                            Set_param   proc    near
05D6  50                        push    AX
05D7  53                        push    BX
05D8  52                        push    DX
05D9  56                        push    SI
                                ;
05DA  8B 36 0414 R              mov     SI,param        ;use digit entry value as index
05DE  4E                        dec     SI              ; into the digit table to find
05DF  D1 E6                     sal     SI,1            ; the base address of the table
05E1  D1 E6                     sal     SI,1            ; where the binary value of the
05E3  2E: 8B 9C 0414 R          mov     BX,dig_table[SI] ;entered digits will be stored
05E8  2E: 8B 94 0416 R          mov     DX,dig_table[SI+2] ; and whether byte or word
                                ;
05ED  8B 36 040F R              mov     SI,sensor       ;use sensor value as index into
                                                        ; the storage table
                                ;
05F1  83 3E 0414 R 05           cmp     param,5         ;upper calibration?
05F6  75 16                     jne     chk_6           ;if not, check lower calibration
                                ;
05F8  E8 049E R                 call    read_adc        ;get sensor value
05FB  A0 0402 R                 mov     AL,value
05FE  C6 06 0422 R 00           mov     ee_ready,0      ;write to EEPROM
0603  26: 88 84 002D R          mov     adc_2[SI],AL
0608  E8 042C R                 call    hold
060B  EB 1B 90                  jmp     store           ;store entered digits
                                ;
060E  83 3E 0414 R 06           chk_6:  cmp     param,6         ;lower calibration?
0613  75 13                     jne     store           ;if not, store entered digits
0615  E8 049E R                 call    read_adc
0618  A0 0402 R                 mov     AL,value
061B  C6 06 0422 R 00           mov     ee_ready,0      ;write to EEPROM
0620  26: 88 84 0024 R          mov     adc_1[SI],AL
0625  E8 042C R                 call    hold
                                ;
```

```
0628  85 D2                  store:      test    DX,DX           ;if byte table entry, store it
062A  74 20                               je      store_byte
                            ;
062C  D1 E6                               sal     SI,1
062E  8B 16 0416 R                        mov     DX,entry        ;write lower byte to EEPROM
0632  C6 06 0422 R 00                     mov     ee_ready,0
0637  26: 88 10                           mov     ES:[BX+SI],DL   ;store word table entry
063A  E8 042C R                           call    hold
                            ;
063D  C6 06 0422 R 00                     mov     ee_ready,0      ;write upper byte to EEPROM
0642  26: 88 70 01                        mov     ES:[BX+SI+1],DH
0646  E8 042C R                           call    hold
0649  EB 1C 90                            jmp     set_ret         ;and continue
                            ;
064C  A1 0416 R             store_byte:   mov     AX,entry        ;get entered value
064F  A3 0420 R                           mov     y,AX
0652  E8 066C R                           call    Pre_comp        ;pre-compute a,b,c
0655  E8 073D R                           call    Find_x          ;find adc value from scaled
                            ;
0658  8A 16 041F R                        mov     DL,x            ;write byte to EEPROM
065C  C6 06 0422 R 00                     mov     ee_ready,0
0661  26: 88 10                           mov     ES:[BX+SI],DL   ;store byte table entry
0664  E8 042C R                           call    hold
                            ;
0667  5E                    set_ret:      pop     SI
0668  5A                                 pop     DX
0669  5B                                 pop     BX
066A  58                                 pop     AX
066B  C3                                 ret
066C                        Set_param     endp
                            ;
                            ;
066C                        Pre_comp      proc    near
066C  50                                 push    AX
066D  52                                 push    DX
066E  56                                 push    SI
066F  57                                 push    DI
                            ;
0670  8B 36 040F R                        mov     SI,sensor
0674  8B 3E 040F R                        mov     DI,sensor
0678  D1 E7                               sal     DI,1
067A  26: 8B 85 0048 R                    mov     AX,val_2[DI]
067F  26: 2B 85 0036 R                    sub     AX,val_1[DI]
0684  A3 0418 R                           mov     A,AX            ;a=(val2-val1)
                            ;
0687  26: 8A 84 002D R                    mov     AL,adc_2[SI]
068C  26: 2A 84 0024 R                    sub     AL,adc_1[SI]
0691  A2 041E R                           mov     C,AL            ;c=(adc2-adc1)
                            ;
0694  26: 8A 84 002D R                    mov     AL,adc_2[SI]
0699  B4 00                               mov     AH,0
069B  26: F7 AD 0036 R                    imul    val_1[DI]
06A0  A3 041A R                           mov     B,AX
```

```
1009  06A3  89 16 041C R                    mov   B+2,DX
1010  06A7  26: 8A 84 0024 R                mov   AL,adc_1[SI]
1011  06AC  B4 00                           mov   AH,0
1012  06AE  26: F7 AD 0048 R                imul  val_2[DI]
1013  06B3  29 06 041A R                    sub   b,AX            ;b=(adc2*val1)-
1014  06B7  19 16 041C R                    sbb   b+2,DX          ;  (adc1*val2)
1015                                ;
1016  06BB  5F                              pop   DI
1017  06BC  5E                              pop   SI
1018  06BD  5A                              pop   DX
1019  06BE  58                              pop   AX
1020  06BF  C3                              ret
1021  06C0                        Pre_comp  endp
1022                                ;
1023                                ;
1024  06C0                        Expand    proc  near
1025  06C0  50                              push  AX
1026  06C1  53                              push  BX
1027                                ;
1028  06C2  A1 040F R                       mov   AX,sensor   ;sensor no. < types*sensor/crate?
1029  06C5  3D 0009                         cmp   AX,sensors_per_crate*no_types
1030  06C8  72 0E                           jb    in_crate    ;if so, sensor is in crate
1031                                ;
1032  06CA  2D 0006                         sub   AX,sensors_per_crate*(no_types-1)
1033  06CD  A3 0411 R                       mov   kind,AX         ;form kind
1034  06D0  C6 06 0413 R 00                 mov   crate_no,0      ;form crate number
1035  06D5  EB 13 90                        jmp   expand_ret
1036                                ;
1037  06D8  B4 00                 in_crate: mov   AH,0            ;form crate number
1038  06DA  B3 03                           mov   BL,sensors_per_crate
1039  06DC  F6 FB                           idiv  BL
1040  06DE  FE C0                           inc   AL
1041  06E0  A2 0413 R                       mov   crate_no,AL
1042                                ;
1043  06E3  86 E0                           xchg  AH,AL           ;form kind
1044  06E5  B4 00                           mov   AH,0
1045  06E7  A3 0411 R                       mov   kind,AX
1046                                ;
1047  06EA  5B                    expand_ret: pop   BX
1048  06EB  58                              pop   AX
1049  06EC  C3                              ret
1050  06ED                        Expand    endp
1051                                ;
1052                                ;
1053  06ED                        Convert   proc  near
1054  06ED  50                              push  AX
1055  06EE  53                              push  BX
1056  06EF  52                              push  DX
1057                                ;
1058  06F0  B0 20                           mov   AL,' '
1059  06F2  F7 C2 FFFF                      test  DX,0FFFFh       ;number < 0?
1060  06F6  79 04                           jns   pos             ;if not, skip
1061                                ;
```

```
1062  06F8  F7 DA                          neg   DX              ;if so, display minus sign
1063  06FA  B0 2D                          mov   AL,'-'
1064                                ;
1065  06FC  B4 2D                   pos:   mov   AH,2Dh          ;sucessively divide DX by 10
1066  06FE  E8 0783 R                      call  Display         ; to generate three digits
1067  0701  8B C2                          mov   AX,DX
1068  0703  B3 0A                          mov   BL,10
1069  0705  F6 FB                          idiv  BL
1070  0707  50                             push  AX
1071  0708  B4 00                          mov   AH,0
1072  070A  F6 FB                          idiv  BL
1073  070C  50                             push  AX
1074  070D  8B 36 0411 R                   mov   si,kind
1075  0711  D1 E6                          sal   SI,1
1076  0713  2E: 8B 9C 0259 R               mov   BX,format[si]   ;display three digits in
1077  0718  2E: 8A 67 14                   mov   AH,CS:[BX]+20   ; position specified in format
1078  071C  0C 30                          or    AL,00110000b    ; table indexed with sensor kind
1079  071E  E8 0783 R                      call  Display
1080                                ;
1081  0721  58                             pop   AX
1082  0722  86 E0                          xchg  AH,AL
1083  0724  2E: 8A 67 15                   mov   AH,CS:[BX]+21
1084  0728  0C 30                          or    AL,00110000b
1085  072A  E8 0783 R                      call  Display
1086                                ;
1087  072D  58                             pop   AX
1088  072E  86 E0                          xchg  AH,AL
1089  0730  2E: 8A 67 16                   mov   AH,CS:[BX]+22
1090  0734  0C 30                          or    AL,00110000b
1091  0736  E8 0783 R                      call  Display
1092                                ;
1093  0739  5A                             pop   DX
1094  073A  5B                             pop   BX
1095  073B  58                             pop   AX
1096  073C  C3                             ret
1097  073D                        Convert  endp
1098                                ;
1099                                ;
1100  073D                        Find_x   proc  near
1101  073D  50                             push  AX
1102  073E  53                             push  BX
1103  073F  52                             push  DX
1104                                ;
1105  0740  A1 0420 R                      mov   AX,y    ;x=(y*c)-b/a
1106  0743  8A 16 041E R                   mov   DL,c
1107  0747  B6 00                          mov   DH,0
1108  0749  F7 EA                          imul  DX
1109  074B  2B 06 041A R                   sub   AX,b
1110  074F  1B 16 041C R                   sbb   DX,B+2
1111  0753  8B 1E 0418 R                   mov   BX,a
1112  0757  F7 FB                          idiv  BX
1113  0759  A2 041F R                      mov   x,AL
1114                                ;
```

```
1115    075C  5A                              pop     DX
1116    075D  5B                              pop     BX
1117    075E  58                              pop     AX
1118    075F  C3                              ret
1119    0760                          Find_x  endp
1120                                  ;
1121                                  ;
1122    0760                          Find_y  proc    near
1123    0760  50                              push    AX
1124    0761  53                              push    BX
1125    0762  52                              push    DX
1126                                  ;
1127    0763  A0 041F R                       mov     AL,x    ;y=(x*a)-b/c
1128    0766  B4 00                           mov     AH,0
1129    0768  F7 2E 0418 R                    imul    A
1130    076C  03 06 041A R                    add     AX,b
1131    0770  13 16 041C R                    adc     DX,b+2
1132    0774  8A 1E 041E R                    mov     BL,c
1133    0778  B7 00                           mov     BH,0
1134    077A  F7 FB                           idiv    BX
1135    077C  A3 0420 R                       mov     y,AX
1136                                  ;
1137    077F  5A                              pop     DX
1138    0780  5B                              pop     BX
1139    0781  58                              pop     AX
1140    0782  C3                              ret
1141    0783                          Find_y  endp
1142                                  ;
1143                                  ;
1144    0783                          Display proc    near
1145    0783  50                              push    AX
1146    0784  53                              push    BX
1147    0785  52                              push    DX
1148                                  ;
1149    0786  F6 06 040B R FF                 test    net,0FFh        ;if net=0, write display
1150    078B  74 22                           je      cont_disp
1151                                  ;
1152    078D  B7 00                           mov     BH,0
1153    078F  8A DC                           mov     BL,AH
1154    0791  F6 06 040B R 01                 test    net,01h
1155    0796  75 0F                           jne     for_send
1156                                  ;
1157    0798  F6 06 040B R 02                 test    net,02h
1158    079D  74 23                           je      disp_ret
1159                                  ;
1160    079F  26: 88 87 0CCC R                mov     out_buf+0CCh[BX],AL  ;if net=2, write out_bug
1161    07A4  EB 1C 90                        jmp     disp_ret
1162                                  ;
1163    07A7  26: 88 87 0ACC R        for_send: mov     send_buf+0CCh[BX],AL ;if net=1, write send_buf
1164    07AC  EB 14 90                        jmp     disp_ret
1165                                  ;
1166    07AF  BA 0000                 cont_disp: mov    DX,PPORT_porta  ;toggle display control signals
1167    07B2  EE                              out     DX,AL           ; to display character in AL
```

```
1168                                    ;                                       ; in position in AH
1169    07B3  BA 0001                                   mov     DX,PPORT_portb
1170    07B6  8A C4                                     mov     AL,AH
1171    07B8  EE                                        out     DX,AL
1172                                    ;
1173    07B9  0C 40                                     or      AL,01000000b
1174    07BB  EE                                        out     DX,AL
1175                                    ;
1176    07BC  24 BF                                     and     AL,10111111b
1177    07BE  EE                                        out     DX,AL
1178                                    ;
1179    07BF  B0 80                                     mov     AL,10000000b
1180    07C1  EE                                        out     DX,AL
1181                                    ;
1182    07C2  5A                        disp_ret:       pop     DX
1183    07C3  5B                                        pop     BX
1184    07C4  58                                        pop     AX
1185    07C5  C3                                        ret
1186    07C6                            Display         endp
1187                                    ;
1188                                    ;
1189    07C6                            Disp_screen     proc    near
1190    07C6  56                                        push    SI
1191    07C7  50                                        push    AX
1192    07C8  53                                        push    BX
1193                                    ;
1194    07C9  81 3E 0408 R FFFF                         cmp     screen_no,0FFFFh        ;don't display?
1195    07CF  74 1B                                     je      done_3                  ;then just return
1196                                    ;
1197    07D1  8B 36 0408 R                              mov     SI,screen_no    ;index into screen list
1198    07D5  D1 E6                                     sal     SI,1            ; with screen number and display
1199    07D7  2E: 8B 9C 02A4 R                          mov     BX,screen[SI]   ; menu screen of 20 characters
1200    07DC  B4 20                                     mov     AH,20h
1201                                    ;
1202    07DE  2E: 8A 07                 loop_3:         mov     AL,CS:[BX]
1203    07E1  E8 0783 R                                 call    Display
1204    07E4  FE C4                                     inc     AH
1205    07E6  43                                        inc     BX
1206    07E7  80 FC 34                                  cmp     AH,20h+20
1207    07EA  75 F2                                     jne     loop_3
1208                                    ;
1209    07EC  5B                        done_3:         pop     BX
1210    07ED  58                                        pop     AX
1211    07EE  5E                                        pop     SI
1212    07EF  C3                                        ret
1213    07F0                            Disp_screen     endp
1214                                    ;
1215                                    ;
1216    07F0                            Disp_form       proc    near
1217    07F0  56                                        push    SI
1218    07F1  50                                        push    AX
1219    07F2  53                                        push    BX
1220                                    ;
```

```
1221    07F3  8B 36 0411 R                        mov   SI,kind          ;index into format table with
1222    07F7  D1 E6                               sal   SI,1             ; kind value and display
1223    07F9  2E: 8B 9C 0259 R                    mov   BX,format[SI]    ; message format
1224    07FE  B4 20                               mov   AH,20h
1225                                  ;
1226    0800  2E: 8A 07               loop_4:     mov   AL,CS:[BX]
1227    0803  E8 0783 R                           call  Display
1228    0806  FE C4                               inc   AH
1229    0808  43                                  inc   BX
1230    0809  80 FC 34                            cmp   AH,20h+20
1231    080C  75 F2                               jne   loop_4
1232    080E  A0 0413 R                           mov   AL,crate_no
1233    0811  0C 30                               or    AL,00110000b
1234    0813  B4 2A                               mov   AH,2Ah
1235    0815  E8 0783 R                           call  Display
1236                                  ;
1237    0818  5B                                  pop   BX
1238    0819  58                                  pop   AX
1239    081A  5E                                  pop   SI
1240    081B  C3                                  ret
1241    081C                          Disp_form   endp
1242                                  ;
1243                                  ;
1244    081C                          Disp_count  proc  near
1245    081C  50                                  push  AX
1246    081D  53                                  push  BX
1247                                  ;
1248    081E  A0 040E R                           mov   AL,out_count     ;sucessively divide out of
1249    0821  B4 00                               mov   AH,0             ; threshold count by 10 to
1250    0823  B3 0A                               mov   BL,10            ; generate two digits for
1251    0825  F6 FB                               idiv  BL               ; display
1252    0827  50                                  push  AX
1253                                  ;
1254    0828  B4 07                               mov   AH,7
1255    082A  0C 30                               or    AL,00110000b
1256    082C  E8 0783 R                           call  Display
1257                                  ;
1258    082F  58                                  pop   AX
1259    0830  86 E0                               xchg  AH,AL
1260    0832  B4 08                               mov   AH,8
1261    0834  0C 30                               or    AL,00110000b
1262    0836  E8 0783 R                           call  Display
1263                                  ;
1264    0839  5B                                  pop   BX
1265    083A  58                                  pop   AX
1266    083B  C3                                  ret
1267    083C                          Disp_count  endp
1268                                  ;
```

```
                        page
                        ;********************************************************************************
                        ; Keyint is the interrupt handler for the keyboard.
                        ;********************************************************************************
083C                    Keyint          proc    far
                        ;
083C  50                                push    AX              ;save used registers
083D  53                                push    BX
083E  52                                push    DX
083F  56                                push    SI
                        ;
0840  FB                                sti                     ;enable interrupts for possible
                                                                ; ADC or Network requests
                        ;
0841  E4 02                             in      AL,PPORT_portc  ;get key value pressed (0-F)
0843  24 0F                             and     AL,00001111b    ;clear out high-order nibble
0845  B4 00                             mov     AH,0            ; and rest of AX register
0847  A2 0404 R                         mov     key_val,AL      ;store away key value pressed
                        ;
084A  3C 0C                             cmp     AL,0Ch          ;if a C was not pressed,
084C  75 0C                             jne     chk_A           ; continue onward
084E  C7 06 0408 R 0000                 mov     screen_no,0     ;if a C was pressed, reset
0854  E8 07C6 R                         call    Disp_screen     ; display to first menu and
0857  E9 0910 R                         jmp     exit            ; exit keyboard routine
                        ;
085A  3C 0A             chk_A:          cmp     AL,0Ah          ;if an A was not pressed,
085C  75 06                             jne     cont_0          ; continue onward
085E  E8 0537 R                         call    Show_out        ;if an A was pressed, show
0861  E9 0910 R                         jmp     exit            ; out of threshold 2 sensors
                        ;
0864  F7 06 0405 R 00FF cont_0:         test    dig_entry,0FFh  ;check for digit entry mode
086A  74 33                             je      cont_1          ;if not, then continue onward
                        ;
086C  3C 09                             cmp     AL,9            ;if not a digit, just exit
086E  76 03                             jbe     cont_2
0870  E9 0910 R                         jmp     exit
                        ;
0873  0C 30             cont_2:         or      AL,00110000b    ;change BCD to ASCII
0875  8B 36 0411 R                      mov     SI,kind         ;set SI to be the index into
0879  D1 E6                             sal     SI,1            ; format table to locate the
087B  2E: 8B 9C 0259 R                  mov     BX,format[SI]   ; correct digit diplay positions
                        ;
0880  F6 06 0407 R FF                   test    dig_cnt,0FFh    ;if digit count is not 0, then
0885  75 1B                             jne     more_digits     ; this is not the first digit
                        ;
0887  2E: 8A 67 14                      mov     AH,CS:[BX]+20   ;display in first digit position
088B  E8 0783 R                         call    Display
088E  24 0F                             and     AL,00001111b    ;change ASCII to BCD
0890  B2 64                             mov     DL,100          ;multiply entered digit by 100
0892  F6 E2                             mul     DL              ; and start digit to binary
0894  A3 0416 R                         mov     entry,AX        ; conversion by storing entry
0897  C6 06 0407 R 02                   mov     dig_cnt,2       ;next time, the digit entry will
089C  EB 72 90                          jmp     exit            ; be the second digit
```

```
1322                                    ;
1323    089F  EB 41 90                  cont_1:         jmp     cont_3
1324                                    ;
1325    08A2  80 3E 0407 R 02           more_digits:    cmp     dig_cnt,2       ;if this is not the second digit
1326    08A7  75 19                                     jne     third_digit     ; entry, then it must be third
1327                                    ;
1328    08A9  2E: 8A 67 15                              mov     AH,CS:[BX]+21   ;display in 2nd digit position
1329    08AD  E8 0783 R                                 call    Display
1330    08B0  24 0F                                     and     AL,00001111b    ;change ASCII to BCD
1331    08B2  B2 0A                                     mov     DL,10           ;multiply entered digit by 10
1332    08B4  F6 E2                                     mul     DL              ; and continue digit to binary
1333    08B6  01 06 0416 R                              add     entry,AX        ; conversion by adding to entry
1334    08BA  C6 06 0407 R 03                           mov     dig_cnt,3       ;next time, the digit entry will
1335    08BF  EB 4F 90                                  jmp     exit            ; be the third digit
1336                                    ;
1337    08C2  2E: 8A 67 16              third_digit:    mov     AH,CS:[BX]+22   ;display in 3rd digit position
1338    08C6  E8 0783 R                                 call    Display
1339    08C9  24 0F                                     and     AL,00001111b    ;change ASCII to BCD
1340    08CB  B4 00                                     mov     AH,0            ;zero out rest of AX, and finish
1341    08CD  01 06 0416 R                              add     entry,AX        ; conversion by adding to entry
1342    08D1  C6 06 0407 R 00                           mov     dig_cnt,0       ;reset digit count
1343                                    ;
1344    08D6  E8 05D6 R                                 call    Set_param
1345    08D9  C7 06 0405 R 0000                         mov     dig_entry,0     ;terminate digit entry mode
1346    08DF  EB 2F 90                                  jmp     exit            ; and leave keyboard routine
1347                                    ;
1348    08E2  81 3E 0408 R FFFF         cont_3:         cmp     screen_no,0FFFFh ;if leave screen alone,
1349    08E8  74 26                                     je      exit            ;  then just leave routine
1350                                    ;
1351    08EA  8B 36 0408 R                              mov     SI,screen_no    ;use screen number as index into
1352    08EE  D1 E6                                     sal     SI,1            ; screen table to get address of
1353    08F0  2E: 8B 9C 02A4 R                          mov     BX,screen[SI]   ; valid key list which begins
1354    08F5  BE 0014                                   mov     SI,20           ; after the screen contents
1355                                    ;
1356    08F8  2E: 3B 00                 loop_0:         cmp     AX,CS:[BX+SI]   ;scan through valid key list
1357    08FB  74 0C                                     je      match           ; and stop when a match is found
1358                                    ;
1359    08FD  2E: 81 38 FFFF                            cmp     word ptr CS:[BX+SI],0FFFFh ;if end of list is
1360    0902  74 0C                                     je      exit            ; detected, just leave routine
1361                                    ;
1362    0904  83 C6 04                                  add     SI,4            ;otherwise, point to next key
1363    0907  EB EF                                     jmp     short loop_0    ; and see if it matches
1364                                    ;
1365    0909  2E: FF 50 02              match:          call    word ptr CS:[BX+SI+2] ;if a match is found, call
1366                                                                            ; routine corresponding to key
1367                                    ;
1368    090D  E8 07C6 R                                 call    Disp_screen     ;display next screen
1369                                    ;
1370    0910  BA FF22                   exit:           mov     DX,EOI_reg      ;signal end of keyboard
1371    0913  B8 8000                                   mov     AX,EOI_value    ; interrupt
1372    0916  EF                                        out     DX,AX
1373                                    ;
1374    0917  5E                                        pop     SI              ;restore used registers
```

```
0918  5A                        pop     DX
0919  5B                        pop     BX
091A  58                        pop     AX
                        ;
091B  CF                        iret                    ;return from interrupt
091C                    Keyint  endp
                        ;
                        ;
                        ;*************************************************************************
                        ; Adcint is the interrupt handler for the analog-to-digital converter
                        ;*************************************************************************
091C                    Adcint  proc    far
091C  52                        push    DX
091D  50                        push    AX
                        ;
091E  8B 16 0400 R              mov     DX,adc_port
0922  EC                        in      AL,DX           ;get converted value
0923  A2 0402 R                mov     value,AL
0926  C6 06 0403 R 00           mov     ready,0         ;set adc status to ready
                        ;
092B  BA FF22                   mov     DX,EOI_reg      ;end of interrupt to controller
092E  B8 8000                   mov     AX,EOI_value
0931  EF                        out     DX,AX
                        ;
0932  58                        pop     AX
0933  5A                        pop     DX
0934  CF                        iret
0935                    Adcint  endp
                        ;
                        ;
                        ;*************************************************************************
                        ; EEint is the interrupt handler for the EEPROM write cycle
                        ;*************************************************************************
0935                    EEint   proc    far
0935  52                        push    DX
0936  50                        push    AX
                        ;
0937  C6 06 0422 R 01           mov     ee_ready,1      ;set EEPROM status to ready
                        ;
093C  BA FF22                   mov     DX,EOI_reg      ;end of interrupt to controller
093F  B8 8000                   mov     AX,EOI_value
0942  EF                        out     DX,AX
                        ;
0943  58                        pop     AX
0944  5A                        pop     DX
0945  CF                        iret
0946                    EEint   endp
                        ;
                        ;
                        ;*************************************************************************
                        ; Netint is the interrupt handler for the LAN message received
                        ;*************************************************************************
0946                    Netint  proc    far
```

```
1428  0946  52                              push  dx
1429  0947  50                              push  ax
1430                              ;
1431  0948  B4 00                           mov   AH,0            ;save variables
1432  094A  FF 36 040F R                    push  sensor
1433  094E  FF 36 0414 R                    push  param
1434  0952  FF 36 0411 R                    push  kind
1435  0956  A0 0413 R                       mov   AL,crate_no
1436  0959  50                              push  AX
1437  095A  FF 36 0418 R                    push  a
1438  095E  FF 36 041A R                    push  b
1439  0962  A0 041E R                       mov   AL,c
1440  0965  50                              push  AX
1441  0966  A0 041F R                       mov   AL,x
1442  0969  50                              push  AX
1443  096A  FF 36 0420 R                    push  y
1444  096E  FF 36 0416 R                    push  entry
1445                              ;
1446  0972  BA FF2A                         mov   DX,PMASK_reg    ;disable keyboard and network
1447  0975  B0 01                           mov   AL,01h
1448  0977  EE                              out   DX,AL
1449  0978  FB                             sti                   ;enable interrupts
1450                              ;
1451  0979  B4 00                           mov   AH,0            ;read out message from rec_buf
1452  097B  26: A0 0803 R                   mov   AL,rec_buf+3
1453  097F  A3 040F R                       mov   sensor,AX       ;get sensor number
1454                              ;
1455  0982  26: A0 0804 R                   mov   AL,rec_buf+4
1456  0986  A3 0414 R                       mov   param,AX        ;get parameter
1457                              ;
1458  0989  26: A0 0805 R                   mov   AL,rec_buf+5
1459  098D  26: 8A 26 0806 R                mov   AH,rec_buf+6
1460  0992  A3 0416 R                       mov   entry,AX        ;get entry
1461                              ;
1462  0995  C6 06 040B R 01                 mov   net,1           ;expand sensor no. to variables
1463  099A  E8 06C0 R                       call  Expand
1464                              ;
1465  099D  26: 80 3E 0807 R 00             cmp   rec_buf+7,0     ;operation display sensor value?
1466  09A3  75 09                           jne   net_1
1467  09A5  E8 049E R                       call  Read_adc        ;get sensor value
1468  09A8  E8 057C R                       call  Val_param       ;format sensor value display
1469  09AB  EB 36 90                        jmp   net_ret
1470                              ;
1471  09AE  26: 80 3E 0807 R 01   net_1:    cmp   rec_buf+7,1     ;operation display param value?
1472  09B4  75 06                           jne   net_2
1473  09B6  E8 0595 R                       call  Get_param       ;get parameter value
1474  09B9  EB 28 90                        jmp   net_ret
1475                              ;
1476  09BC  26: 80 3E 0807 R 02   net_2:    cmp   rec_buf+7,2     ;operation set param value?
1477  09C2  75 06                           jne   net_3
1478  09C4  E8 05D6 R                       call  Set_param       ;set parameter value
1479  09C7  EB 1A 90                        jmp   net_ret
1480                              ;
```

```
1481  09CA  26: 80 3E 0807 R 03    net_3:      cmp   rec_buf+7,3     ;operation disable sensor?
1482  09D0  75 06                               jne   net_4
1483  09D2  E8 052B R                           call  disable       ;disable sensor
1484  09D5  EB 0C 90                            jmp   net_ret
1485                                     ;
1486  09D8  26: 80 3E 0807 R 04    net_4:      cmp   rec_buf+7,4     ;operation enable sensor?
1487  09DE  75 03                               jne   net_ret
1488  09E0  E8 051F R                           call  enable        ;enable sensor
1489                                     ;
1490  09E3  C6 06 040B R 00        net_ret:    mov   net,0
1491  09E8  26: C6 06 0A01 R 02                 mov   send_buf+1,PC_ID;response message source ID
1492  09EE  26: C6 06 0A02 R EC                 mov   send_buf+2,0ECh ;pointer to first character
1493                                     ;
1494  09F4  BA 0101                             mov   DX,NET_com    ;send message in send_buf
1495  09F7  B0 0B                               mov   AL,0Bh
1496  09F9  EE                                  out   DX,AL
1497                                     ;
1498  09FA  BA 0100                             mov   DX,NET_status ;transmission acknowledged?
1499  09FD  EC                     test_4:      in    AL,DX
1500  09FE  24 03                               and   AL,03h
1501  0A00  3C 03                               cmp   AL,03h
1502  0A02  75 F9                               jne   test_4        ;if not, check again
1503                                     ;
1504  0A04  BA 0101                             mov   DX,NET_com    ;enable another receive in
1505  0A07  B0 04                               mov   AL,04h        ; ref_buf
1506  0A09  EE                                  out   DX,AL
1507                                     ;
1508  0A0A  8F 06 0416 R                        pop   entry         ;restore variables
1509  0A0E  8F 06 0420 R                        pop   y
1510  0A12  58                                  pop   AX
1511  0A13  A2 041F R                           mov   x,AL
1512  0A16  58                                  pop   AX
1513  0A17  A2 041E R                           mov   c,AL
1514  0A1A  8F 06 041A R                        pop   b
1515  0A1E  8F 06 0418 R                        pop   a
1516  0A22  58                                  pop   AX
1517  0A23  A2 0413 R                           mov   crate_no,AL
1518  0A26  8F 06 0411 R                        pop   kind
1519  0A2A  8F 06 0414 R                        pop   param
1520  0A2E  8F 06 040F R                        pop   sensor
1521                                     ;
1522  0A32  BA FF2A                             mov   DX,PMASK_reg  ;enable keyboard and network
1523  0A35  B0 07                               mov   AL,07h
1524  0A37  EE                                  out   DX,AL
1525                                     ;
1526  0A38  BA FF22                             mov   DX,EOI_reg    ;end of interrupt to controller
1527  0A3B  B8 8000                             mov   AX,EOI_value
1528  0A3E  EF                                  out   DX,AX
1529                                     ;
1530  0A3F  58                                  pop   AX
1531  0A40  5A                                  pop   DX
1532  0A41  CF                                  iret
1533  0A42                         Netint       endp
```

```
                        ;
                        ;
                        ;*************************************************************************
                        ; DivOint is the interrupt handler for divide by zero
                        ;*************************************************************************
0A42                    DivOint         proc    far
0A42  CF                                iret            ;just return
0A43                    DivOint         endp
                        ;
                        ;
0A43                    CODE_SEG        ends
                                        end     Initialize
```

The IBM Personal Computer MACRO Assembler 05-02-85 PAGE Symbols-1

Segments and groups:

Name	Size	align	combine	class
CODE_SEG	0FF5	PARA	NONE	'CODE'
DATA_SEG	042C	AT	0000	
EXTRA_SEG	0E00	AT	0200	
STACK_SEG	0400	AT	01C0	

Symbols:

Name	Type	Value	Attr	
A	L WORD	0418	DATA_SEG	
ADCINT	F PROC	091C	CODE_SEG	Length =0019
ADC_1	L BYTE	0024	EXTRA_SEG	Length =0009
ADC_2	L BYTE	002D	EXTRA_SEG	Length =0009
ADC_LOC	Number	0080		
ADC_PORT	L WORD	0400	DATA_SEG	
ADC_VECT	L WORD	0034	DATA_SEG	
AGAIN_0	L NEAR	0438	CODE_SEG	
ARROW_D	Number	0019		
ARROW_R	Number	001D		
ARROW_U	Number	0018		
B	L WORD	041A	DATA_SEG	
BACK	N PROC	0453	CODE_SEG	Length =0014
BACK_TABLE	L WORD	03C6	CODE_SEG	
C	L BYTE	041E	DATA_SEG	
CALIB	N PROC	04E4	CODE_SEG	Length =000D
CHK_6	L NEAR	060E	CODE_SEG	
CHK_A	L NEAR	085A	CODE_SEG	
CIR_INC	L NEAR	0562	CODE_SEG	
CLASS	N PROC	0467	CODE_SEG	Length =0019
CLR_LOOP	L NEAR	00FF	CODE_SEG	
CONT_0	L NEAR	0864	CODE_SEG	
CONT_1	L NEAR	089F	CODE_SEG	
CONT_2	L NEAR	0873	CODE_SEG	
CONT_3	L NEAR	08E2	CODE_SEG	
CONT_DISP	L NEAR	07AF	CODE_SEG	
CONVERT	N PROC	06ED	CODE_SEG	Length =0050
CRATE	N PROC	0480	CODE_SEG	Length =001E
CRATE_NO	L BYTE	0413	DATA_SEG	
DELAY	N PROC	0434	CODE_SEG	Length =000B
DELAY_RET	L NEAR	043D	CODE_SEG	
DIG_CNT	L BYTE	0407	DATA_SEG	
DIG_ENTRY	L WORD	0405	DATA_SEG	
DIG_TABLE	L WORD	0414	CODE_SEG	
DISABLE	N PROC	052B	CODE_SEG	Length =000C
DISP	N PROC	0503	CODE_SEG	Length =000A
DISPLAY	N PROC	0783	CODE_SEG	Length =0043
DISP_COUNT	N PROC	081C	CODE_SEG	Length =0020
DISP_FORM	N PROC	07F0	CODE_SEG	Length =002C
DISP_RET	L NEAR	07C2	CODE_SEG	

DISP_SCREEN.	N PROC	07C6	CODE_SEG	Length =002A
DIVOINT.	F PROC	0A42	CODE_SEG	Length =0001
DIVO_VECT.	L WORD	0000	DATA_SEG	
DONE_3	L NEAR	07EC	CODE_SEG	
DOWN	N PROC	04F8	CODE_SEG	Length =000B
EEINT.	F PROC	0935	CODE_SEG	Length =0011
EE_READY	L BYTE	0422	DATA_SEG	
EE_VECT.	L WORD	0038	DATA_SEG	
ENABLE	N PROC	051F	CODE_SEG	Length =000C
ENTRY.	L WORD	0416	DATA_SEG	
EOI_REG.	Number	FF22		
EOI_VALUE.	Number	8000		
EXIT	L NEAR	0910	CODE_SEG	
EXPAND	N PROC	06C0	CODE_SEG	Length =002D
EXPAND_RET	L NEAR	06EA	CODE_SEG	
FIND_X	N PROC	073D	CODE_SEG	Length =0023
FIND_Y	N PROC	0760	CODE_SEG	Length =0023
FORMAT	L WORD	0259	CODE_SEG	
FORMAT_0	L BYTE	025F	CODE_SEG	
FORMAT_1	L BYTE	0276	CODE_SEG	
FORMAT_2	L BYTE	028D	CODE_SEG	
FOR_SEND	L NEAR	07A7	CODE_SEG	
GET_PARAM.	N PROC	0595	CODE_SEG	Length =0041
HOLD	N PROC	042C	CODE_SEG	Length =0008
HOLD2.	L NEAR	042C	CODE_SEG	
INITIALIZE	F PROC	0000	CODE_SEG	Length =0139
INTO_REG	Number	FF38		
INTO_VALUE	Number	0003		
INT1_REG	Number	FF3A		
INT1_VALUE	Number	0001		
INT2_REG	Number	FF3C		
INT2_VALUE	Number	0000		
INT3_REG	Number	FF3E		
INT3_VALUE	Number	0002		
IN_CRATE	L NEAR	06D8	CODE_SEG	
JUMP	F PROC	0FE0	CODE_SEG	Length =000C
KEYINT	F PROC	083C	CODE_SEG	Length =00E0
KEY_VAL.	L BYTE	0404	DATA_SEG	
KEY_VECT	L WORD	0030	DATA_SEG	
KIND	L WORD	0411	DATA_SEG	
LMCS_REG	Number	FFA2		
LMCS_VALUE	Number	01FB		
LOOK	L NEAR	053C	CODE_SEG	
LOOP_0	L NEAR	08F8	CODE_SEG	
LOOP_3	L NEAR	07DE	CODE_SEG	
LOOP_4	L NEAR	0800	CODE_SEG	
LOW_EXCEED	L NEAR	01BC	CODE_SEG	
MAIN	N PROC	0139	CODE_SEG	Length =009D
MAIN_LOOP.	L NEAR	013E	CODE_SEG	
MAIN_RESET	L NEAR	019B	CODE_SEG	
MATCH.	L NEAR	0909	CODE_SEG	
MMCS_REG	Number	FFA6		
MMCS_VALUE	Number	03FB		

```
MORE_DIGITS. . . . . .    L NEAR  08A2    CODE_SEG
MPCS_REG . . . . . .      Number  FFA8
MPCS_VALUE . . . . . .    Number  81BB
NET. . . . . .            L BYTE  040B    DATA_SEG
NETINT . . . . . .        F PROC  0946    CODE_SEG        Length =00FC
NET_1. . . . . .          L NEAR  09AE    CODE_SEG
NET_2. . . . . .          L NEAR  09BC    CODE_SEG
NET_3. . . . . .          L NEAR  09CA    CODE_SEG
NET_4. . . . . .          L NEAR  09D8    CODE_SEG
NET_COM. . . . . .        Number  0101
NET_MASK . . . . . .      Number  0100
NET_RET. . . . . .        L NEAR  09E3    CODE_SEG
NET_STATUS . . . . . .    Number  0100
NET_VECT . . . . . .      L WORD  003C    DATA_SEG
NEXT_SENSOR. . . . . .    L NEAR  0186    CODE_SEG
NEXT_SHOW. . . . . .      L NEAR  0569    CODE_SEG
NO_READY_1 . . . . . .    L NEAR  04B1    CODE_SEG
NO_READY_2 . . . . . .    L NEAR  04BF    CODE_SEG
NO_SENSORS . . . . . .    Number  0009
NO_TYPES . . . . . .      Number  0003
OK . . . . . .            Number  0001
OUT_BUF. . . . . .        L BYTE  0C00    EXTRA_SEG       Length =0200
OUT_COUNT. . . . . .      L BYTE  040E    DATA_SEG
PACS_REG . . . . . .      Number  FFA4
PACS_VALUE . . . . . .    Number  003B
PARAM. . . . . .          L WORD  0414    DATA_SEG
PC_ID. . . . . .          Number  0002
PMASK_REG. . . . . .      Number  FF2A
POS. . . . . .            L NEAR  06FC    CODE_SEG
PPORT_CNTL . . . . . .    Number  0003
PPORT_CNTLVAL. . . . . .  Number  0081
PPORT_PORTA. . . . . .    Number  0000
PPORT_PORTB. . . . . .    Number  0001
PPORT_PORTC. . . . . .    Number  0002
PRE_COMP . . . . . .      N PROC  066C    CODE_SEG        Length =0054
READY. . . . . .          L BYTE  0403    DATA_SEG
READ_ADC . . . . . .      N PROC  049E    CODE_SEG        Length =002C
READ_BYTE. . . . . .      L NEAR  05BE    CODE_SEG
READ_RET . . . . . .      L NEAR  05CF    CODE_SEG
REC_BUF. . . . . .        L BYTE  0800    EXTRA_SEG       Length =0200
RE_ENABLE. . . . . .      L NEAR  0180    CODE_SEG
SCREEN . . . . . .        L WORD  02A4    CODE_SEG
SCREEN_0 . . . . . .      L BYTE  02B2    CODE_SEG
SCREEN_1 . . . . . .      L BYTE  02D8    CODE_SEG
SCREEN_2 . . . . . .      L BYTE  02FA    CODE_SEG
SCREEN_3 . . . . . .      L BYTE  0324    CODE_SEG
SCREEN_4 . . . . . .      L BYTE  034A    CODE_SEG
SCREEN_5 . . . . . .      L BYTE  036C    CODE_SEG
SCREEN_6 . . . . . .      L BYTE  0392    CODE_SEG
SCREEN_NO. . . . . .      L WORD  0408    DATA_SEG
SCROLL . . . . . .        N PROC  043F    CODE_SEG        Length =0014
SCROLL_TABLE . . . . . .  L WORD  03B8    CODE_SEG
SEND_BUF . . . . . .      L BYTE  0A00    EXTRA_SEG       Length =0200
```

SEND_MESS.	N PROC	01D6	CODE_SEG	Length =0083
SENSOR	L WORD	040F	DATA_SEG	
SENSORS_PER_CRATE.	Number	0003		
SENSOR_MAP	L BYTE	03D4	CODE_SEG	
SENSOR_TBL	L BYTE	0423	DATA_SEG	Length =0009
SET.	N PROC	050D	CODE_SEG	Length =0012
SET_PARAM.	N PROC	05D6	CODE_SEG	Length =0096
SET_RET.	L NEAR	0667	CODE_SEG	
SHOW_COUNT	L WORD	040C	DATA_SEG	
SHOW_OUT	N PROC	0537	CODE_SEG	Length =0045
SHOW_RET	L NEAR	057A	CODE_SEG	
STORE.	L NEAR	0628	CODE_SEG	
STORE_BYTE	L NEAR	064C	CODE_SEG	
SYMBOL	L BYTE	040A	DATA_SEG	
TEST_4	L NEAR	09FD	CODE_SEG	
THIRD_DIGIT.	L NEAR	08C2	CODE_SEG	
THRESH	N PROC	04D7	CODE_SEG	Length =000D
THRESH_L1.	L BYTE	0012	EXTRA_SEG	Length =0009
THRESH_L2.	L BYTE	001B	EXTRA_SEG	Length =0009
THRESH_U1.	L BYTE	0009	EXTRA_SEG	Length =0009
THRESH_U2.	L BYTE	0000	EXTRA_SEG	Length =0009
TOP_LINE	L BYTE	00D6	CODE_SEG	
TOP_LOOP	L NEAR	00EA	CODE_SEG	
TOP_STACK.	E WORD	0400	STACK_SEG	
UMCS_REG	Number	FFA0		
UMCS_VALUE	Number	FE3C		
UNDL	Number	00FF		
UP	N PROC	04F1	CODE_SEG	Length =0007
UP_EXCEED.	L NEAR	01A2	CODE_SEG	
U_0.	Number	00B0		
U_1.	Number	00B1		
U_2.	Number	00B2		
U_3.	Number	00B3		
VAL.	N PROC	04CA	CODE_SEG	Length =000D
VALUE.	L BYTE	0402	DATA_SEG	
VAL_1.	L WORD	0036	EXTRA_SEG	Length =0009
VAL_2.	L WORD	0048	EXTRA_SEG	Length =0009
VAL_PARAM.	N PROC	057C	CODE_SEG	Length =0019
X.	L BYTE	041F	DATA_SEG	
Y.	L WORD	0420	DATA_SEG	

Warning Errors	Severe Errors
0	0

```
Warning: No STACK segment

 Start  Stop   Length  Name                    Class
 00000H 00FF4H 0FF5H  CODE_SEG                CODE

 Origin   Group

Program entry point at 0000:0000
```

```
Symbol Cross Reference                (#.is definition)      Cref-1

A. . . . . .  1308  519   552   999  1111  1129  1437  1515
ADCINT . . . . . .  378   379  1386#  1402
ADC_1. . . . . .  1828  953  1002  1010
ADC_2. . . . . .  1868  944  1001  1005
ADC_LOC. . . . . .  2878  747
ADC_PORT . . . . . .  1148  749  1390
ADC_VECT . . . . . .  1078  378   379
AGAIN_0. . . . . .  6748  676
ARROW_D. . . . . .  368  506
ARROW_R. . . . . .  348  594   615
ARROW_U. . . . . .  358  497

B. . . . . .  1318  520   551  1008  1009  1013  1014  1109  1110  1130  1131  1438  1514
BACK . . . . . .  596   602   609   617   623   630   637  6958   705
BACK_TABLE . . . . . .  6448  700

C. . . . . .  1338  521   550  1003  1106  1132  1439  1513
CALIB. . . . . .  623   7808  784
CHK_6. . . . . .  939  9488
CHK_A. . . . . .  1289  12948
CIR_INC. . . . . .  854  8568
CLASS. . . . . .  595   595   602  7088   719
CLR_LOOP . . . . . .  4148  417
CODE . . . . . .  254
CODE_SEG . . . . . .  2548  255  1544
CONT_0 . . . . . .  1295  12998
CONT_1 . . . . . .  1300  13238
CONT_2 . . . . . .  1303  13068
CONT_3 . . . . . .  1323  13488
CONT_DISP. . . . . .  1158  11668
CONVERT. . . . . .  881   913  1053#  1097
CRATE. . . . . .  608   608   609  7228   736
CRATE_NO . . . . . .  1278  517   554   725  1034  1041  1232  1435  1517

DATA_SEG . . . . . .  1008  141   255   323
DELAY. . . . . .  341   543  6718   679
DELAY_RET. . . . . .  675  6778
DIG_CNT. . . . . .  1198  393  1311  1320  1325  1334  1342
DIG_ENTRY. . . . . .  1188  392   811  1299  1345
DIG_TABLE. . . . . .  6568  898   899   932   933
DISABLE. . . . . .  8278  833  1483
DISP . . . . . .  636  8008  804
DISPLAY. . . . . .  408   429   434   439  1066  1079  1085  1091  1144#  1186  1203  1227  1235  1256
                    1262  1315  1329  1338

DISP_COUNT . . . . . .  487  1244#  1267
DISP_FORM. . . . . .  809   877   892  1216#  1241
DISP_RET . . . . . .  1158  1161  1164  1182#
DISP_SCREEN. . . . . .  389  1189#  1213  1291  1368
DIVOINT. . . . . .  372   373  1539#  1541
DIVO_VECT. . . . . .  1038  372   373
DONE_3 . . . . . .  1195  1209#
DOWN . . . . . .  629  7938  797

EEINT. . . . . .  381   382  1408#  1421
```

Symbol Cross Reference (# is definition) Cref-2

```
EE_READY . . . . . 1368  665   943   952   961   965   976   1412
EE_VECT. . . . . 1090  381   382
ENABLE . . . . . 8188  824   1488
ENTRY. . . . . 1298  526   545   960   970   1319  1333  1341  1444  1460  1508
EOI_REG. . . . . 2638  1370  1395  1414  1526
EOI_VALUE. . . . . 2818  1371  1396  1415  1527
EXIT . . . . . 1292  1297  1304  1321  1335  1346  1349  1360  13708
EXPAND . . . . . 529   849   10248 1050  1463
EXPAND_RET . . . . . 1035  10478
EXTRA_SEG. . . . . 1658  212   256   326

FIND_X . . . . . 973   11008 1119
FIND_Y . . . . . 879   911   11228 1141
FORMAT . . . . . 5628  1076  1223  1309
FORMAT_0 . . . . . 562   5668
FORMAT_1 . . . . . 563   5718
FORMAT_2 . . . . . 564   5768
FOR_SEND . . . . . 1155  11638

GET_PARAM. . . . . 801   8878  919   1473

HOLD . . . . . 6648  668   945   954   963   967   978
HOLD2. . . . . 6658  666

INITIALIZE . . . . . 2928  301   442   1545
INT0_REG . . . . . 2648  356
INT0_VALUE . . . . . 2828  357
INT1_REG . . . . . 2658  360
INT1_VALUE . . . . . 2838  361
INT2_REG . . . . . 2668  364
INT2_VALUE . . . . . 2848  365
INT3_REG . . . . . 2678  368
INT3_VALUE . . . . . 2858  369
IN_CRATE . . . . . 1030  10378

JUMP . . . . . 294   2978  302

KEYINT . . . . . 375   376   12738 1380
KEY_VAL. . . . . 1178  710   724   1286
KEY_VECT . . . . . 1058  375   376
KIND . . . . . 1268  516   555   712   731   1033  1045  1074  1221  1307  1434  1518

LMCS_REG . . . . . 2598  306
LMCS_VALUE . . . . . 2778  307
LOOK . . . . . 8408  864   866
LOOP_0 . . . . . 13568 1363
LOOP_3 . . . . . 12028 1207
LOOP_4 . . . . . 12268 1231
LOW_EXCEED . . . . . 465   5028

MAIN . . . . . 441   4458  509
MAIN_LOOP. . . . . 4498  480   490
MAIN_RESET . . . . . 485   4898
MATCH. . . . . 1357  13658
MMCS_REG . . . . . 2618  315
```

```
Symbol Cross Reference                  (# is definition)      Cref-3

MMCS_VALUE . . . . . .  2790  316
MORE_DIGITS. . . . . .  1312  1325#
MPCS_REG . . . . . .  2620  319
MPCS_VALUE . . . . . .  2800  320

NET. . . . . .  1220  394   531   533   1149  1154  1157  1462  1490
NETINT . . . . . .  384   385   1427#  1533
NET_1. . . . . .  1466  1471#
NET_2. . . . . .  1472  1476#
NET_3. . . . . .  1477  1481#
NET_4. . . . . .  1482  1486#
NET_COM. . . . . .  2880  345   539   1494  1504
NET_MASK . . . . . .  2890  352
NET_RET. . . . . .  1469  1474  1479  1484  1487  1490#
NET_STATUS . . . . . .  2900  1498
NET_VECT . . . . . .  1110  384   385
NEXT_SENSOR. . . . . .  450   4780
NEXT_SHOW. . . . . .  841   844   859#
NO_READY_1 . . . . . .  7510  752
NO_READY_2 . . . . . .  7560  757
NO_SENSORS . . . . . .  270   137   166   170   174   178   182   186   190   194   416   479   853   863
NO_TYPES . . . . . .  280   1029  1032

OK . . . . . .  370   471
OUT_BUF. . . . . .  2080  535   536   537   1160
OUT_COUNT. . . . . .  1240  395   489   493   502   1248

PACS_REG . . . . . .  2600  310
PACS_VALUE . . . . . .  2780  311
PARAM. . . . . .  1280  515   556   774   781   794   810   894   928   938   948   1433  1456  1519
PC_ID. . . . . .  380   536   1491
PMASK_REG. . . . . .  2680  451   474   1446  1522
POS. . . . . .  1060  1065#
PPORT_CNTL . . . . . .  2730  334
PPORT_CNTLVAL. . . . . .  2740  335
PPORT_PORTA. . . . . .  2700  1166
PPORT_PORTB. . . . . .  2710  1169
PPORT_PORTC. . . . . .  2720  338   1283
PRE_COMP . . . . . .  878   910   972   988#  1021

READY. . . . . .  1160  391   751   753   756   1393
READ_ADC . . . . . .  457   739#  762   766   848   941   950   1467
READ_BYTE. . . . . .  902   908#
READ_RET . . . . . .  906   913#
REC_BUF. . . . . .  2000  419   1452  1455  1458  1459  1465  1471  1476  1481  1486
RE_ENABLE. . . . . .  468   474#  495   499   504   508

SCREEN . . . . . .  583#  1199  1353
SCREEN_0 . . . . . .  583   591#
SCREEN_1 . . . . . .  584   598#
SCREEN_2 . . . . . .  585   604#
SCREEN_3 . . . . . .  586   612#
SCREEN_4 . . . . . .  587   619#
SCREEN_5 . . . . . .  588   625#
SCREEN_6 . . . . . .  589   632#
```

Symbol Cross Reference (# is definition) Cref-4

```
SCREEN_NO. . . . . 120# 388  685  688  698  701  716  733  768  775  782  788  795  802
                   812  1194 1197 1290 1348 1351

SCROLL . . . . . 596  603  610  617  624  630  637  682#  692
SCROLL_TABLE . . . . . 641# 687
SEND_BUF . . . . . 204# 1163 1491 1492
SEND_MESS. . . . . 472  498  507  512# 559
SENSOR . . . . . 125# 455  456  458  514  528  557  714  715  732  744  820  829  846
                 900  935  994  995  1028 1432 1453 1520

SENSORS_PER_CRATE. . . . . 26# 715  728  1029 1032 1038
SENSOR_MAP . . . . . 648# 743  745
SENSOR_TBL . . . . . 137# 414  449  467  470  494  496  503  505  821  830  840  843
SET. . . . . 636  807# 815
SET_PARAM. . . . . 922# 985  1344 1478
SET_RET. . . . . 968  980#
SHOW_COUNT . . . . . 123# 396  838  856  860
SHOW_OUT . . . . . 836# 870  1296
SHOW_RET . . . . . 857  861  868#
STACK_SEG. . . . . 50#  56   255  329
STORE. . . . . 946  949  956#
STORE_BYTE . . . . . 957  970#
SYMBOL . . . . . 121# 471  497  506  534

TEST_4 . . . . . 1499# 1502
THIRD_DIGIT. . . . . 1326 1337#
THRESH . . . . . 616  773# 777
THRESH_L1. . . . . 174# 658
THRESH_L2. . . . . 178# 464  656
THRESH_U1. . . . . 170# 658
THRESH_U2. . . . . 166# 461  656
TOP_LINE . . . . . 400# 407
TOP_LOOP . . . . . 398  405# 411
TOP_STACK. . . . . 55#  332

UMCS_REG . . . . . 258# 298
UMCS_VALUE . . . . . 276# 299
UNDL . . . . . 29#  568  568  568  573  573  573  578  578  578
UP . . . . . 629  787# 790
UP_EXCEED. . . . . 462  493#
U_0. . . . . 30#  591  612  625  632
U_1. . . . . 31#  592  605  613  626  633
U_2. . . . . 32#  598  605  619
U_3. . . . . 33#  605

VAL. . . . . 616  765# 770
VALUE. . . . . 115# 460  875  942  951  1392
VAL_1. . . . . 190# 660  999  1007
VAL_2. . . . . 194# 660  997  1012
VAL_PARAM. . . . . 532  767  850  873# 884  1468

X. . . . . 134# 523  548  876  909  975  1113 1127 1441 1511

Y. . . . . 135# 525  546  880  912  971  1105 1135 1443 1509
```

APPENDIX B

IBM PC MONITORING PROGRAM

This appendix contains the Advanced BASIC program listing for the monitoring system monitoring program.

```
10 'IBM PC MONITORING PROGRAM
20 'BRADLEY S. RUBIN   5/2/85
30 '
40 'THIS PROGRAM MONITORS MESSAGES SENT BY A MONITORING UNIT OVER
50 'AN ARCNET CONNECTION
60 '
70 'RESET ARCNET BOARD AND DELAY (APPROX 100 MSEC)
80 OUT &H208,&HFF
90 FOR X=1 TO 100:NEXT X
100 '
110 'CLEAR CONTROLLER FLAGS
120 OUT &H201,&H1E
130 '
140 'MEMORY LOCATION OF LAN BUFFER
150 DEF SEG = &H8000
160 '
170 CLS
180 '
190 'ENABLE RECEIVE INTO BUFFER 1
200 OUT &H201,&HC
210 '
220 'CONTROLLER STATUS REGISTER
230 STATUS = INP(&H200)
240 '
250 'RECEIVED MESSAGE?
260 IF STATUS < 128 THEN 230
270 '
280 'TIMESTAMP
290 PRINT DATE$,TIME$,
300 '
310 'PRINT STATION ID
320 PRINT "Station";PEEK(512),
330 '
340 'PRINT MESSAGE FROM BUFFER
350 FOR X=0 TO 19
360 PRINT CHR$(PEEK(512+&HEC+X));
370 NEXT X
380 PRINT
390 '
400 'RECEIVE ANOTHER
410 GOTO 200
420 '
430 END
```

APPENDIX C

IBM PC VIRTUAL TERMINAL PROGRAM

This appendix contains the Advanced BASIC program listing for the virtual terminal program.

```
10 'IBM PC VIRTUAL TERMINAL PROGRAM
20 'BRADLEY S. RUBIN   5/2/85
30 '
40 'THIS PROGRAM ALLOWS COMMUNICATION BETWEEN THE IBM PC AND ANY
50 'MONITORING UNIT TO PROVIDE VIRTURAL KEYBOARD SERVICE
60 '
70 'SENSORS PER CRATE
80 PER=3
90 '
100 'RESET ARCNET BOARD AND DELAY (APPROX 100 MSEC)
110 OUT &H208,&HFF
120 FOR X=1 TO 100:NEXT X
130 '
140 'CLEAR CONTROLLER FLAGS
150 OUT &H201,&H1E
160 '
170 'MEMORY LOCATION OF LAN BUFFER
180 DEF SEG = &H8000
190 '
200 CLS
210 INPUT "Enter Station ID";ID
220 '
230 PRINT: PRINT "Enter Type:"
240 PRINT "0- +5 Volts DC"
250 PRINT "1- +5 Amps DC"
260 PRINT "2- Air Temperature"
270 INPUT KIND
280 '
290 PRINT: INPUT "Enter Crate Number";CRATE
300 '
310 PRINT: PRINT "Enter Function:"
320 PRINT "0- Display Sensor Value"
330 PRINT "1- Display Parameter Value"
340 PRINT "2- Set Parameter Value"
350 PRINT "3- Disable Sensor"
360 PRINT "4- Enable Sensor"
370 INPUT OPER
380 '
390 'NEED PARAMETER LIST?
400 IF OPER = 0 THEN 590
410 IF OPER = 3 THEN 590
420 IF OPER = 4 THEN 590
430 '
440 PRINT: PRINT "Enter Parameter"
450 PRINT "1- Upper Threshold 2"
460 PRINT "2- Lower Threshold 2"
470 PRINT "3- Upper Threshold 1"
480 PRINT "4- Lower Threshold 1"
490 PRINT "5- Upper Calibration"
500 PRINT "6- Lower Calibration"
```

```
510 INPUT PARAM
520 '
530 'NEED ENTRY PROMPT?
540 IF OPER <> 2 THEN GOTO 590
550 '
560 PRINT:INPUT "Enter Value";ENTRY
570 '
580 'COMPUTE SENSOR NUMBER
590 IF CRATE=0 THEN SENSOR=2*PER+KIND ELSE SENSOR=(CRATE-1)*PER+KIND
600 '
610 'LOAD TRANSMIT BUFFER 0 WITH REQUEST MESSAGE
620 POKE &H1,ID
630 POKE &H2,3
640 POKE &H3,SENSOR
650 POKE &H4,PARAM
660 POKE &H5,ENTRY-INT(ENTRY/256)*256
670 POKE &H6,INT(ENTRY/256)
680 POKE &H7,OPER
690 '
700 'TRANSMIT BUFFER 0
710 OUT &H201,&H3
720 '
730 'RECEIVE BUFFER 1
740 OUT &H201,&HC
750 '
760 'WAIT FOR RESPONSE
770 FOR X=1 TO 100:NEXT X
780 '
790 'IF NO RESPONSE EXPECTED, TRY ANOTHER
800 IF OPER = 2 THEN 900
810 IF OPER = 3 THEN 900
820 IF OPER = 4 THEN 900
830 '
840 'PRINT OUT RECEIVED MESSAGE
850 FOR X=0 TO 19
860 PRINT CHR$(PEEK(512+&HEC+X));
870 NEXT X
880 '
890 'TRY ANOTHER?
900 PRINT: INPUT"Another?",Q$
910 IF Q$="y" THEN 200
920 '
930 END
```

APPENDIX D

HARDWARE SCHEMATICS

This appendix contains the hardware schematics for the prototype monitoring unit. Figure D-1 contains the microprocessor, memory and input/output circuitry. Figure D-2 contains the A/D converter and multiplexer circuitry. Figure D-3 contains the local area network circuitry.

Figure D-1. Microprocessor, memory, and I/O circuitry schematic.

Figure D-2. A/D converter and multiplexer circuitry schematic.

Figure D-3. Local area network circuitry schematic.

LIST OF REFERENCES

[1] "INTEL iAPX 188 High Integration 8-Bit Microprocessor : Preliminary Data Sheet," Intel Corporation, 1982.

[2] "Introduction to the 80186 : Application Note AP-186," Intel Corporation, 1983.

[3] "iAPX 86/88, 186/188 User's Manual: Programmer's Reference," Intel Corporation, 1983.

[4] "80188 Errata : Internal Document," Intel Corporation, 1983.

[5] "2764 64K (8K x 8) UV Erasable PROM : Data Sheet," Intel Corporation, 1983.

[6] "2186A Family 8192 x 8 Bit Integrated RAM : Preliminary Data Sheet," Intel Corporation, 1983.

[7] "Designing Memory Systems with the 8K x 8 iRAM : Application Note AP-132," Intel Corporation, 1982.

[8] Fallin, J.F., and Atnether, J.P., "The Chip that Refreshes Itself," Computer Design, March, 1983.

[9] "2817A 16K (2K x 8) Electrically Erasable PROM : Preliminary Data Sheet," Intel Corporation, 1983.

[10] "Designing with the System Friendly 2817A 5V-Only EEPROM : Application Note AP-158," Intel Corporation, 1983.

[11] "8255A/8255A-5 Programmable Peripheral Interface : Data Sheet," Intel Corporation, 1982.

[12] "AND1864 Interface Data," AND.

[13] "MM54C923/MM74C923 20 Key Encoder : Data Sheet," National Semiconductor.

[14] "ADC0816,ADC0817 8-Bit Microprocessor Compatible A/D Converters with 16-Channel Multiplexer : Data Sheet," National Semiconductor. [15] "Using the ADC0808/ADC0809 8-Bit Microprocessor Compatible Analog Converters with 8-Channel Analog Multiplexer : Preliminary Application Note AN-247," National Semiconductor, 1980.

[16] "COM9026 Local Area Network Controller LANC : Data Sheet," Standard Microsystems Corporation.

[17] "ARCNET Designer's Handbook," Datapoint, 1982.

[18] "Using the COM9026 Local Area Network Controller and the COM9032 Local Area Network Transceiver : Technical Note TN5-2," Standard Microsystems Corporation.

[19] "LAND Local Area Network Driver Hybrid : Preliminary Application Note," Zenith Microcircuits, 1985.

[20] "ARCNET-PC Local Area Network Controller Users' Guide," Standard Microsystems Corporation, 1984.

[21] "Macro Assembler Manual," IBM, 1981.

[22] "Disk Operating System Manual," IBM, 1983.

[23] "BASIC Manual," IBM, 1982.

[24] Joshi, S. and Iyer, V. "Protocols and Network-control Chips: A Symbiotic R6lationship," Electronics, January 12, 1984.

[25] Herman, M. "LAN Controller Regulates Token-passing Traffic," Electronic Design, December 22, 1983.

[26] Murphy, J.A., "Token-passing protocol boosts throughput in local networks," Electronics, September, 1982.