01 Jan 1984

A Microprocessor-Controlled Message Display System

Paul D. Stigall
Missouri University of Science and Technology, tigall@mst.edu

Brian E. Lenharth

Recommended Citation

P. D. Stigall and B. E. Lenharth, "A Microprocessor-Controlled Message Display System," *IEEE Micro*, vol. 4, no. 2, pp. 10 - 25, Institute of Electrical and Electronics Engineers; Computer Society, Jan 1984.
The definitive version is available at https://doi.org/10.1109/MM.1984.291315

Microprocessors replace electromechanical components to make this message display system both more efficient and less costly.

A Microprocessor-controlled Message Display System

Paul D. Stigall
Brian E. Lenharth
University of Missouri—Rolla

Electronically controlled message systems which use matrices of ordinary incandescent lamps have become a common means of providing public information, from time-and-temperature displays to stadium scoreboards. The more sophisticated displays, which offer animation and halftone pictures, need one or more computers for real-time control. The smaller and simpler display systems, however, have not benefited from these technological advances, but instead have relied largely on electromechanical components for their modest control requirements. Our goal was to determine whether the smaller systems could also benefit from the use of microprocessors.

We chose to study the kind of message display system typically used for public service and advertising messages. This system has one or more banks of incandescent lamps arranged in a matrix of 60-70 columns by seven or eight rows. Approximately 12-15 characters of a message may be displayed in a given frame, with a maximum message length of 5-10 frames.

System design

After examining the available message display systems, we decided which features would be most desirable in a

0272-1732/84/0400-0010$01.00 © 1984 IEEE

microprocessor-controlled message display system (MCMDS). We wanted a system with

- three separate messages, each containing up to 100 characters;
- two separate displays: the main display (incandescent lamps) and a compact console display (LEDs);
- a display matrix of 64 columns by seven or eight rows of lamps;
- two display modes: a "segmented" display mode and a "scrolled" display mode;
- capability to display message input on the console without affecting messages on the main display;
- capability to call up any stored message;
- a digital thermometer and clock for generating time and temperature data;
- a serial data line or fiber-optic cable for transmission between the control console and display; and
- a minimum of hardware components with as many functions as possible performed by the system software.

Our design of this system is shown in Figure 1. The main processor receives and formats input from various sources, including the keyboard, control switches, status lights, temperature sensor, and time-of-day clock. It stores display data in internal memory and, at the proper time, transfers the data to the main and console displays via serial data links. As each display processor receives a message from the main processor, it translates the message into the desired format and updates the display. The use of three microprocessors may seem excessive for a system of this type, but we located one at each display for a very good reason: a single serial data line between the main processor and display eliminates the expensive and bulky cables used in other systems. The 8-bit character representations used in the display are compressed into 6-bit codes (through the elimination of unused codes) for transmission over the serial line and recreated at the display processor. This compression permits a 25-percent reduction in transmission time compared to systems that use cable.

After considering the available microprocessors, we chose the Intel 8085: it is inexpensive, self-contained (except for required memory, I/O, and event timing circuits), and fast, and it has the speed and architectural features to perform the required functions. One major factor in choosing the 8085 was the availability of the Microkit (now Futuredata) development system to aid in software development. The Futuredata support software consists of a text editor, assembler, and microemulator monitor with the ability (via a hardware accessory) to program 2708-type EPROMs. (Despite the use of 2716 EPROMs, only a single hardware adapter and software routine were required to change from 2708 to 2716 programming capability.)*

All circuitry for the MCMDS logic uses a single 5-volt power supply. Previous generations of microprocessor circuitry (the 8080 processor, 2107 RAMs, 2708 EPROMs, etc.) required up to three separate supply voltages. This created complexity and a greater possibility for failure. The ability to use a 5-volt power supply was a welcome contribution to our goal of keeping hardware to a minimum.

Processor support circuits were chosen from the selection of sophisticated multifunction integrated circuits presently available. The use of these ICs helped minimize the parts count and enabled us to use a simple high-level software interface. The 8155 timer + RAM + I/O circuit provides two or three I/O ports, a 14-bit timer for baud rate and system timer generation, and a 256-byte read/write memory—all needed to support the processor in this application. Read/write memory in the main processor (in addition to that provided by the 8155s) comes from 2114 static RAMs. We chose static RAM over dynamic because the added complexity required to implement the dynamic RAM refresh circuitry far outweighed its price advantage. Program and constant data for all processors are contained on Intel 2716 EPROMs, each of which stores 2048 bytes.

To control the serial data lines between the main processor and the displays, we used Intel 8251 universal syn-

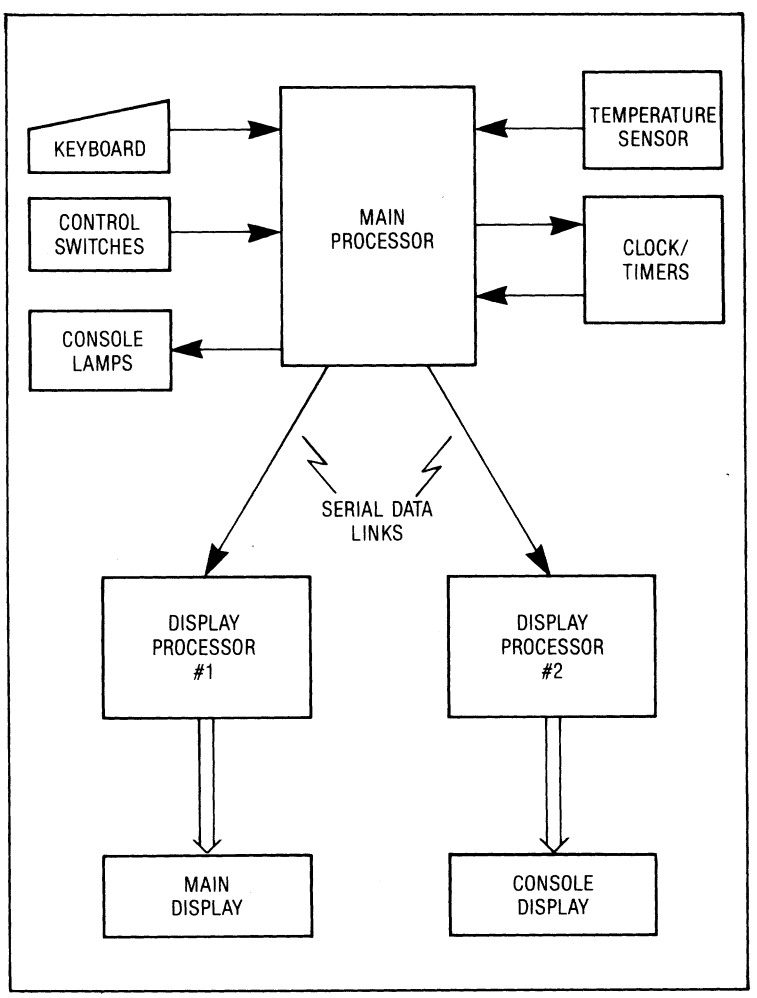


Figure 1. Block diagram of MCMDS—the microprocessor-controlled message display system.

*The Microkit 8085 development system was based on cassette tape and sold in 1978 for $3850. Although this system is no longer available, any one of several current personal computers with the abovementioned hardware and software support could be used as a development system.

Wire-wrapped connections

Low-volume or prototype microprocessor systems usually require an inexpensive method of circuit board layout. The cost and time involved in using single-sided, double-sided, or multilayer PC boards cannot always be justified even though these boards may offer more reliable connections, occupy less space, and have a more pleasing appearance. Inexpensive circuit board layout methods usually compromise ruggedness and packaging density but can improve the ease of servicing.

Wire-wrapping is one of the more popular construction techniques for inexpensive circuit boards that require solderless connections. Contact is made by tightly wrapping a round wire around a square terminal: five complete turns around the terminal give the electrical connection a total of 20 contact points. The wire-wrapping process can be broken into three steps:

- stripping insulation from one end of the wire,
- wrapping the wire around a terminal, and
- unwrapping the wire to correct mistakes or change the design.

Wire-wrapping tools that perform all three steps range from $8 for a manual tool[1] to over $100,000 for a fully automatic wire-wrap machine.[2]

Two popular forms of wire-wrapping, the standard wrap and modified wrap, are illustrated here. The standard wrap winds only bare wire around the terminal; the modified wrap winds a portion of insulated wire around the terminal in addition to the bare wire, increasing the connection's ability to withstand stress and vibration.

Wire-wrapping has five main advantages as a construction technique:

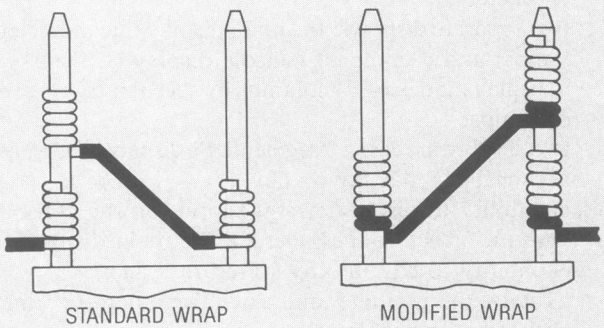
- It is inexpensive, even with the necessary tools. In addition to tools and integrated circuits, the materials include the board (as low as $5 to $10), wire-wrap sockets ($.40 to $2), and the wire ($6 to $7 per 100 ft.).[1]
- It is fast compared to techniques that require soldering.
- It is easy to change bad components and salvage components when the board is no longer needed.
- It is easy to make changes or corrections by unwrapping and correctly rewrapping the connection. (Excessive unwrapping and rewrapping on the same terminal, however, will "round" its edges, reducing the number of reliable contact points.)
- It eliminates thermal shock introduced by soldering.

Wire-wrapping also has several disadvantages:

- Noise and crosstalk are introduced by high-frequency signals, parallel wires, and the length of the wires themselves. However, noise can be reduced with a point-to-point (or "rat nest") wire-wrapping scheme, even though it is less attractive than parallel wires. Circuitry using higher-frequency signals can be isolated during board layout, and connections can be kept as short as possible.
- Tracing a wire from point to point during debugging can be difficult, especially with a rat-nest wire-wrapping scheme. However, this problem can be overcome by very gently tugging on one end of the wire and watching for movement on the other end. Color-coding the wires can also reduce confusion—for example:

  Yellow— Address Lines
  White — Data Lines
  Blue — Control Lines
  Red — +5V Power
  Black — Ground

- Circuit boards with wire-wrap sockets and integrated circuits can be over one inch thick.
- The mechanical connection within the socket can become damaged or corroded and is generally less reliable than a soldered connection.
- Poor insertion into the sockets can result in damaged pins on the integrated circuits.



When used correctly and for the right application, wire-wrapping can be a valuable circuit board layout technique for microprocessor systems. More information on wire-wrapping—as well as on other solderless methods—is available in books[3,4] and magazines.[5]

References

1. *Digi-Key Catalog,* Digi-Key Corporation, Thief River Falls, MN, Jan. 1984.
2. *Wire-Wrap Tools for Solderless Wrapped Connections,* Gardner-Denver, Grand Haven, MI, 1983.
3. B. A. Artwick, *Microcomputer Interfacing,* Prentice-Hall, Englewood Cliffs, NJ, 1980.
4. A. Clements, *Microcomputer Design and Construction,* Prentice-Hall, Englewood Cliffs, NJ, 1982.
5. A. Mangirei, "Wire-Wrapping and Proto-System Techniques," *Byte,* Vol. 6, No. 5, May 1981, pp. 152-170.

chronous/asynchronous receiver/transmitters (USARTs). In this application, the communication on the data line operates in a synchronous mode, transmitting 6-bit characters at 19,200 baud. By using USARTs we eliminated the program's—and programmer's—concern with timing, synchronization, parallel-to-serial conversion, etc.

The MCMDS software is as important as the hardware. The main software timing control is a system interrupt occurring every 100 milliseconds. This causes the system to execute a complete cycle of the control software in this time period so that both displays, including time and temperature information, can be updated as necessary and control switch and keyboard interrupts can be handled. Thus the speed and complexity of each software routine had to be considered when each hardware option was considered. The system software was written in assembly language. Although this was less convenient than a higher-level language, it provided the most compact and efficient code. Structured programming techniques were used in software generation.

Implementation of the system

The electronic hardware of the MCMDS was implemented in a modular fashion. We used wire-wrapping techniques[1] for the main and display processors since it is not time- or cost-effective to design and fashion printed-circuit cards for a single-use application. We did, however, use printed-circuit cards for the circuitry of the display; these plugged directly into a printed-circuit display motherboard. We were especially careful to eliminate noise and crosstalk during the layout and wiring of the unit.

Main processor. The most complex entity in the MCMDS is the main processor (Figure 2). Built around an Intel 8085 microprocessor, it receives input from the keyboard, console control switches, and an analog-to-digital converter wired to a temperature sensor. The processor provides output to the console status lights and to two serial data lines which transmit formatted display data to the display processors.

The heart of the processor is the 8085 and its associated buffers, address and device decoders, program memory, and read/write memory. The eight low-order address lines (which on the 8085 are multiplexed onto the same pins as the data lines) are separated by an 8212 eight-bit input/output port. This port latches and buffers these lines in synchronization with the 8085's address latch enable line. Other address and control lines are buffered and/or inverted by 8T97 and 8T98 circuits, respectively. Low-power Schottky (74LS series) integrated circuits are used in timing, device-selection, and address-decoding circuitry. Programs and constant data are stored in 2716 EPROMs. Data storage is provided by 2114 RAMs (each of 1024 × 4 bits); 512 bytes of RAM are provided in the two 8155 RAM + I/O + timer circuits. Memory address assignments are listed in Table 1.

Operation of the main processor requires a number of timing signals derived from the 8085 clock output (3.072

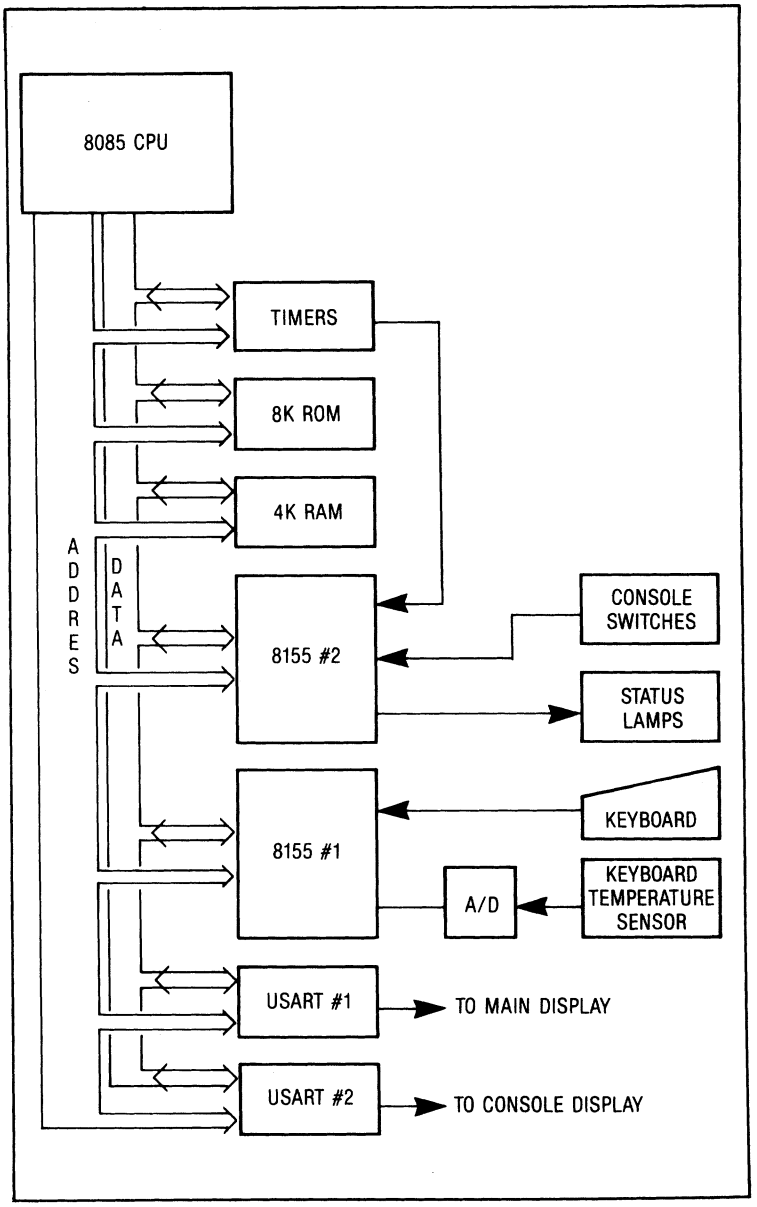


Figure 2. MCMDS main processor block diagram.

Table 1.
MCMDS main processor memory assignment.

STARTING ADDRESS	ENDING ADDRESS	LENGTH (BYTES)	MEMORY TYPE	MEMORY UTILIZATION
0000	07FF	2048	EPROM (#1)	PROGRAM/TABLES
0800	0FFF	2048	EPROM (#2)	PROGRAM/TABLES
1000	17FF	2048	EPROM (#3)	NOT USED
1800	1FFF	2048	EPROM (#4)	NOT USED
2000	23FF	1024	RAM (#1)	MESSAGE STORAGE
2400	27FF	1024	RAM (#2)	MESSAGE STORAGE
2800	2BFF	1024	RAM (#3)	NOT USED
2C00	2FFF	1024	RAM (#4)	NOT USED
3000	33FF	1024	RAM (#5)	NOT USED
3400	37FF	1024	RAM (#6)	NOT USED
3800	3BFF	1024	RAM (#7)	NOT USED
3C00	3FFF	1024	RAM (#8)	NOT USED
4000	40FF	256	RAM (8155 #1)	VARIABLES
4100	41FF	256	RAM (8155 #2)	PROGRAM STACK

Table 2.
MCMDS main processor input/output address assignments.

I/O ADDRESS	DEVICE	PORT	FUNCTION
00	8155 #1	—	COMMAND/STATUS REGISTER
01	8155 #1	PA	TEMPERATURE A/D INPUT
02	8155 #1	PB	KEYBOARD INPUT
03	8155 #1	PC	I/O SYNCHRONIZATION BITS
04	8155 #1	—	TIMER REGISTER (LOW-ORDER)
05	8155 #1	—	TIMER REGISTER (HIGH-ORDER)
10	8155 #2	—	COMMAND/STATUS REGISTER
11	8155 #2	PA	CONTROL SWITCH INPUT
12	8155 #2	PB	STATUS LAMP OUTPUT
13	8155 #2	PC	DEVICES READY AND TIMER
14	8155 #2	—	TIMER REGISTER (LOW-ORDER)
15	8155 #2	—	TIMER REGISTER (HIGH-ORDER)
20	8251 #1	D	DATA REGISTER
21	8251 #1	C	COMMAND/STATUS REGISTER
40	8251 #2	D	DATA REGISTER
41	8251 #2	C	COMMAND/STATUS REGISTER
—	8085	SOD	CLOCK INTERRUPT CLEAR

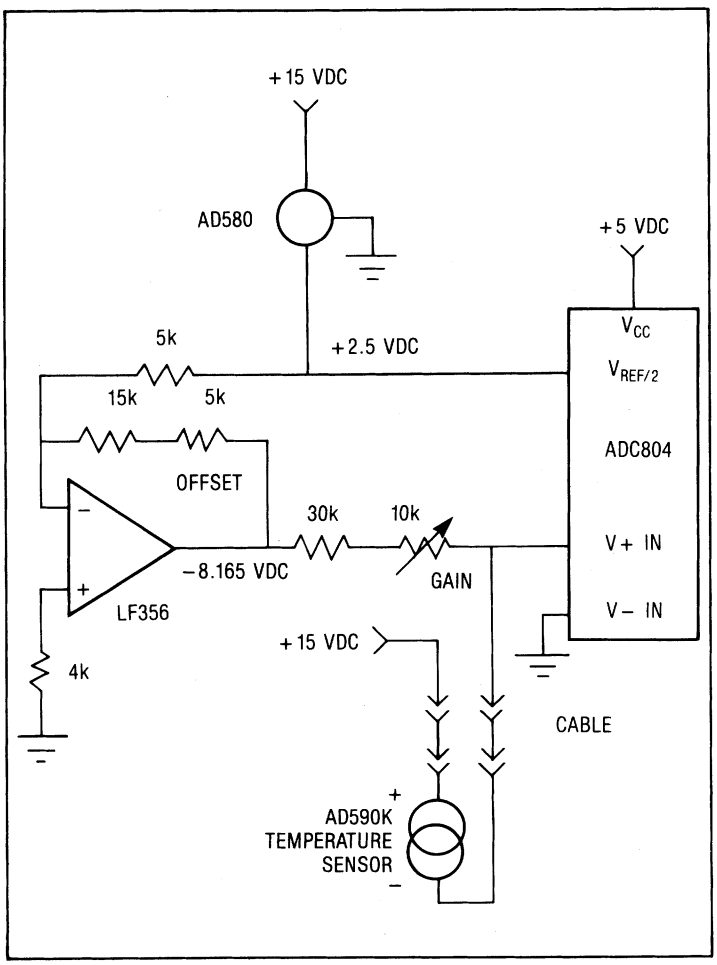


Figure 3. Temperature sensor analog interface.

MHz). To create the 19,200-Hz baud rate for the USARTs, we used the timer in one of the 8155s to divide the processor clock by 160. The timer in the second 8155 divides this figure by 1920 to create the 0.1-Hz interrupt, which controls execution of the software. This figure is again divided by 600, generating a one-pulse-per-minute update of the time-of-day clock. To facilitate rapid setting of the time, a latch and gate enable the insertion of the 0.1-Hz clock on the time input. In addition to the above timing signals, the USARTs require a clock input at least 30 times the baud rate in synchronous operation. This is created by using two flip-flops to divide the system clock by four, yielding a 768-kHz USART clock.

The system control switches (I/O address 11) and status lights (I/O address 12) are connected to ports A and B of the second 8155 circuit (Table 2). The switches connect directly to the port; the status lamp outputs are buffered by 7404 circuits to provide current to the LEDs.

The keyboard encodes the keys into an 8-bit output along with a strobe line, but the code is not the standard ASCII. The eight lines are connected to port B of the first 8155, which operates in a strobed-input mode (ALT4). The strobe line from the 8155 is routed through two monostable multivibrators, delaying the strobe pulse approximately 5 milliseconds to compensate for keyboard-switch bounce. We determined this time delay by observing the keyboard signals with an oscilloscope. The "buffer full" signal from 8155 input port B is connected to a bit of port C on the second 8155; this enables the software to determine if a key has been pressed. Figure 3 shows the temperature input circuitry to the MCMDS. The temperature sensing element is an AD590 two-terminal temperature transducer which provides a PTAT (proportional to absolute temperature) current output of one microampere per degree Kelvin.[2] (The less-expensive LM334 can be made to perform the same function, but it is less accurate.[3]) An AD580 provides a stable 2.5-V reference to the circuitry. An operational amplifier with good DC stability (an LF356) and associated resistors provide current-to-voltage conversion of the transducer signal, as well as gain and offset adjustment of the transducer output. The compensated signal present at the input of the analog-to-digital converter (an ADC804) is scaled and offset so that zero volts corresponds to −40°F (−40°C) and maximum input (5V, which corresponds to a converter output code of FF) is 212°F (100°C). Thus, a 1-bit change in the converter's output indicates a change of 1°F, which corresponds to a 19.61-millivolt change at the input of the converter.

If we assume that the gain and scale trims are properly adjusted, there are two major factors affecting the accuracy of the AD590 temperature transducer and ADC804 converter. The ADC804 has a specified accuracy of one least-significant bit (which in this application corresponds to 1°F), and the mid-priced AD590-K has an accuracy of 1.1°F (when using both gain and scale adjustments). The resulting uncertainty of about 2° could have been reduced to 0.8° by using parts with tighter tolerances (such as the AD590-M and ADC801), but their high cost made them impractical for our purposes.

The main processor provides two serial outputs—one to the console display and the other to the main display. Each serial line is controlled by a separate 8251 USART programmed to operate in synchronous mode at 19,200 baud. The clock signals are generated by the timing circuitry previously described. There are a number of ways

to connect transmitters and receivers; we used a single wire, since the distance was less than one foot. For longer spans, fiber-optic data links provide reliable connection with freedom from interference pickup, ground loops, and generation of radio frequency interference. (The Hewlett-Packard HFBR-1500/2500 is good up to five meters; the HFBR-0200, to 500 meters.[4])

Display processors and displays. The configuration of the display processors is much simpler than that of the main processor, due to their reduced I/O and computing requirements. We used a single 8155 for RAM and timing, a 2716 EPROM for program and data table storage, and an 8251 for serial data reception. Figure 4 illustrates the console display with its three functional components. The interface decodes the address and provides eight buffered "section-enable" lines (each section contains eight columns of the display). These in turn are provided with three buffered data lines to the section controllers. Each section controller contains a 7442 decoder to decode the section-enable and low-order address lines into individual "column-enable" signals. In addition, the data lines are buffered to provide sufficient drive capability for eight display columns. Each display column has two 7475 4-bit latches, which are clocked by the appropriate column-enable line. The display data stored in the latches are displayed on the LED display. The LEDs are driven directly from the complemented outputs of the latches.

The processor interface, section controllers, and data latches of the main display are identical to those of the console display. However, since the main display uses 120V incandescent lamps and the console display uses LEDs, the method used to drive the lamps must differ. As Figure 5 shows, we used thyristors (triacs) in the main display to provide line voltage switching for the incandescent lamps. To eliminate any potential hazards due to the connection of the microprocessor and associated controller components to the alternating current mains, we used optical isolators to provide separation. These Motorola circuits (MOC3031)[5] have self-contained zero-voltage switching circuitry to synchronize the triac turn-on with the zero crossing of the mains. This eliminates RFI from the unsynchronized switching of the triacs.

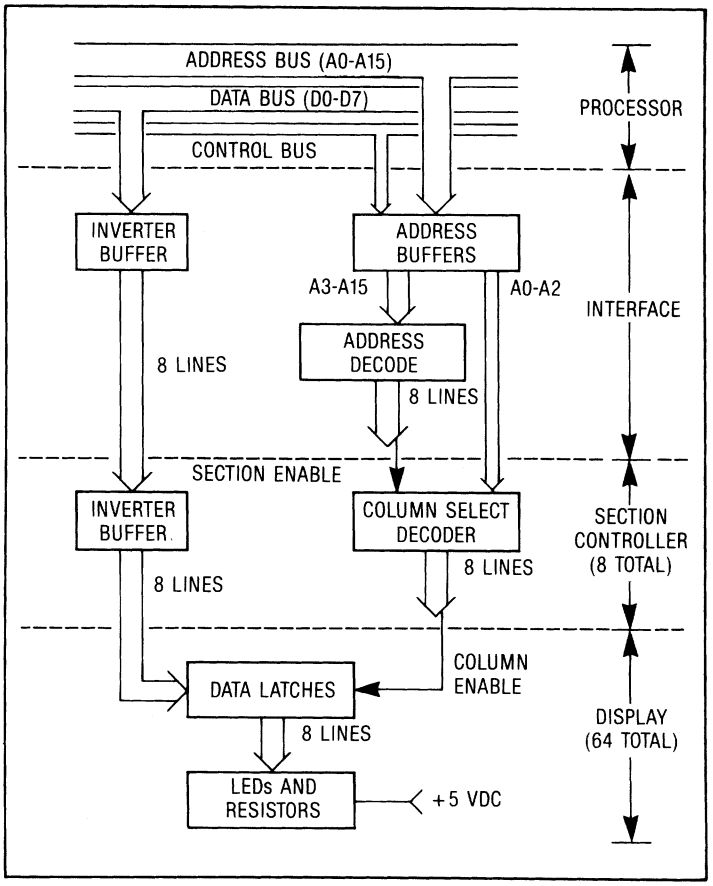


Figure 4. Console display and interface block diagram.

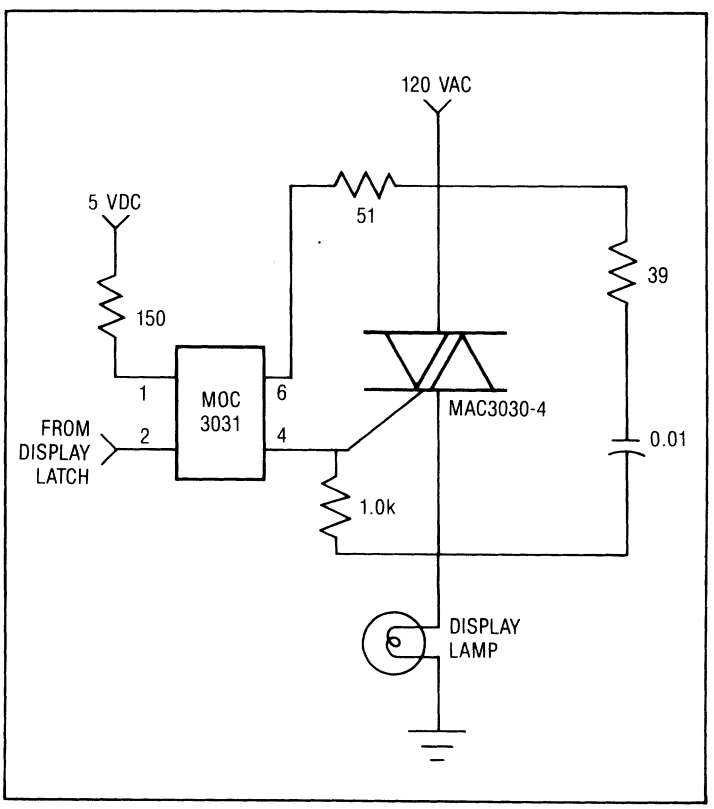


Figure 5. Circuit for logic control of the incandescent display lamp.

MCMDS software

As mentioned earlier, the MCMDS software was written in Intel 8085 assembly language, converted to object machine code, and programmed onto the system EPROMs using the Microkit development system and associated software.** Structured programming techniques were used as extensively as possible. The ensuing descriptions will follow the functional lines of the software.

Main processor program. As the system is turned on, a hardware reset on the 8085 initiates program execution

**A detailed program listing is available from the author for a $20 copying, handling, and mailing charge (outside the United States add $5.00). See the address following Professor Stigall's biography.

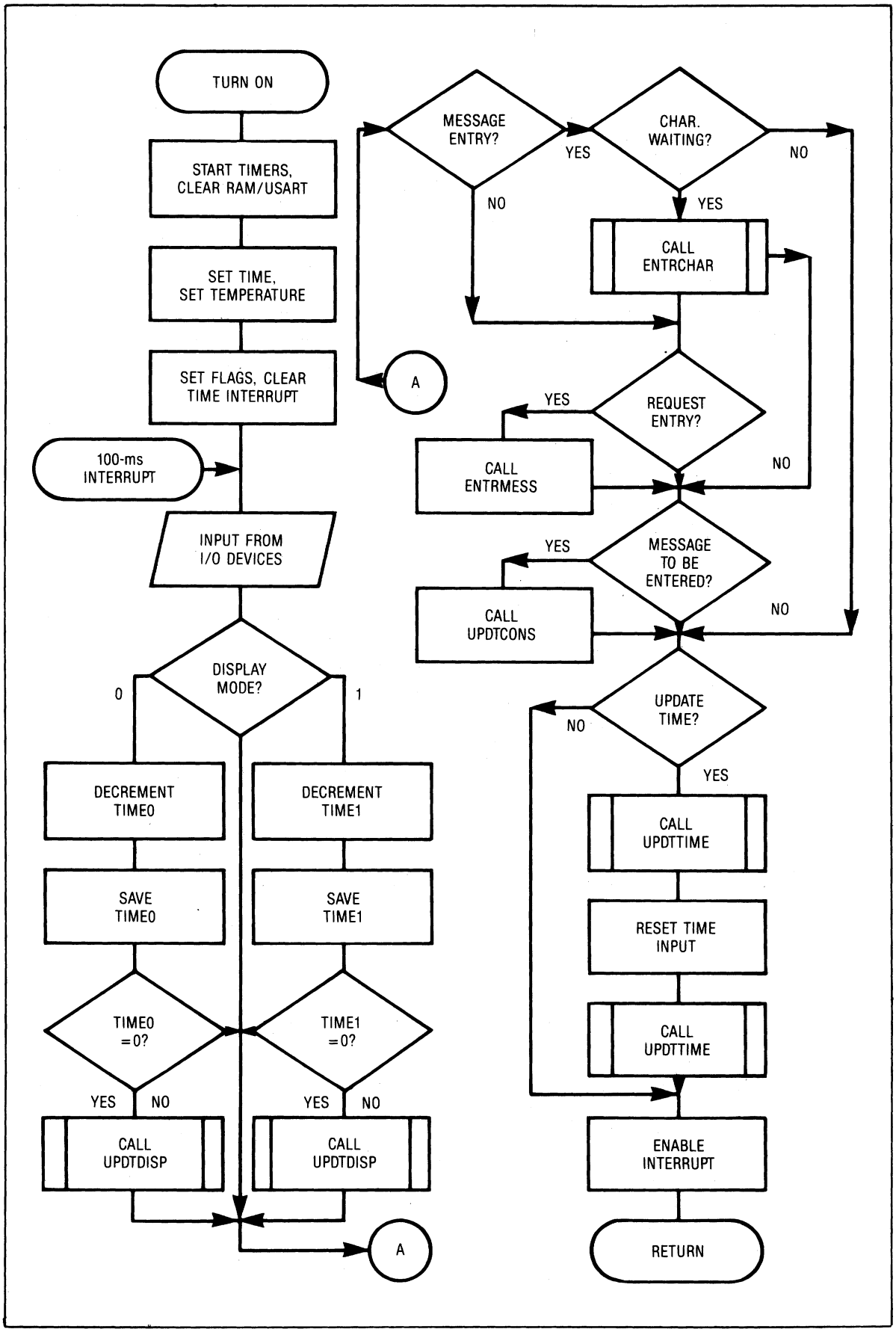


Figure 6. Flowchart of the main program.

at location zero in memory. This begins the turn-on routine, which performs a number of initialization tasks:

- setting the programmable timers on the two 8155s to their proper values,
- clearing console lights,
- clearing the read/write memory,
- initializing the USARTs,
- setting certain data values in the system, and
- initializing the system stack.

Upon completion of these tasks, control is passed to the main control routine.

Main program. The central system software routine, MAINPGM (Figure 6), is executed 10 times per second and is triggered by an interrupt generated on the 8085's RST7.5 interrupt line by the 10.0-Hz clock. It begins by inputting the current data values of all peripherals (time-of-day clock, control switches, keyboard and temperature converter) and their status (empty or data-ready) and storing them for future reference. The display update software timers are then decremented.

When it is time to update the display (counter equals zero), subprogram UPDTDSP is called to perform this task. If a message is currently being entered on the keyboard, and if the keyboard had a waiting character when input at the beginning of MAINPGM, ENTRCHAR is called to handle the input of this character. But if no message is currently being entered, and if a control switch was depressed indicating a request to enter a message, ENTMESS is called to set up for message entry (assuming there is no conflicting pending request for message entry). If the system is in a message entry mode, the console display is updated to show what is being entered into the message buffer rather than what is on the main display. Finally, the time and temperature buffers are updated (if an "update time" input was asserted), and MAINPGM (and the processor) are put into a halt state awaiting the next RST7.5 interrupt.

Message selection and display. The display timer's countdown to zero causes the main program to update the display (UPDTDSP).

Figure 7 shows how UPDTDSP determines the status of the displays and updates them accordingly. If the display done flag (DISPDON) is set, the routine looks for the next message to display and sets the system pointers to display that message—any of three possible messages or the time and temperature, as determined by the positions of the console switches. After it is determined whether the display is segmented or scrolled, INCSEG or INCSCR (respectively) is called to update the display.

The increment-segmented display, INCSEG (Figure 8), is the most complex routine in the system. Before each display is changed, the message storage area is searched beginning at the point where the last segment ended. INCSEG looks for the longest segment—preferably of whole words—shorter than 64 columns and centers it on the display.

This segment is found by incrementing the display pointer to the first nonblank column of the message not yet displayed. The end pointer is set 64 columns past the display pointer and backed toward the display pointer until a word boundary (multiple blank columns) is found. If no word boundary is found, the first blank column separating two characters is used.

When the beginning and end of the next display segment are found, the entire segment is moved to the work area and centered. If the segment represents the end of a message, INCSEG increments the display counter and begins the message again. After the message has been displayed the maximum number of times allowed (three), the DISPDON flag is set.

Compared to the INCSEG routine, INCSCR is a relatively simple operation (Figure 9). The display shows the same message as in the previous display, with one exception: it is shifted one column to the left, with the column at right filled in by the next blank in the message storage area. Leading or trailing blanks are supplied as needed. The end of message is handled in a manner similar to that used in the INCSEG routine.

The two output routines DSPMAIN (Figure 10) and DSPCONS are nearly identical, differing only in the I/O ports which they address (the main and console display, respectively). Each routine polls the USART status register until the USART indicates it is ready to accept a data byte for transmission. Bytes are then transferred from the message work area to the USART until all 64 bytes (plus the display mode command byte) are sent.

Message input and storage. When MAINPGM has determined a valid "enter message" request, routine ENTMESS (Figure 11) is called. ENTMESS sets the message entry pointers to the appropriate message area. System status flags are then set to indicate the entry of a message, followed by an update of the console display lamps. Before returning control to the caller, ENTMESS clears the message work area on the console display for the next message.

When the message entry procedure is initiated, the keyboard handler (Figure 12) performs the work of message entry. ENTRCHAR is entered each time the main program detects an input from the keyboard. The routine first checks to see if the character is an "end-of-message" character; if so, the message entry is terminated. The message is also terminated if it has reached the end of its storage area. By pressing the backspace key, the programmer can delete the last character entered; pressing the space bar inserts a space. If the character is none of these special characters, it either is a legal character or is disregarded as illegal.

The 8-bit code input from the keyboard makes possible a character set of 256 characters. Since there are only 64 allowed in this system, a translation to an internal character code of smaller scope (6 bits) is desirable to minimize character table storage requirements. This internal code combines four low-order bits from the keyboard code with two high-order bits from CHARTAB1. The four low-order bits of the keyboard code are used as an index into CHARTAB1. Once the internal character representation is found, CHARTAB2 is consulted to find the index into CHARTAB3. CHARTAB3 contains the column-by-column 6-bit encoding of the display pattern for every character in the

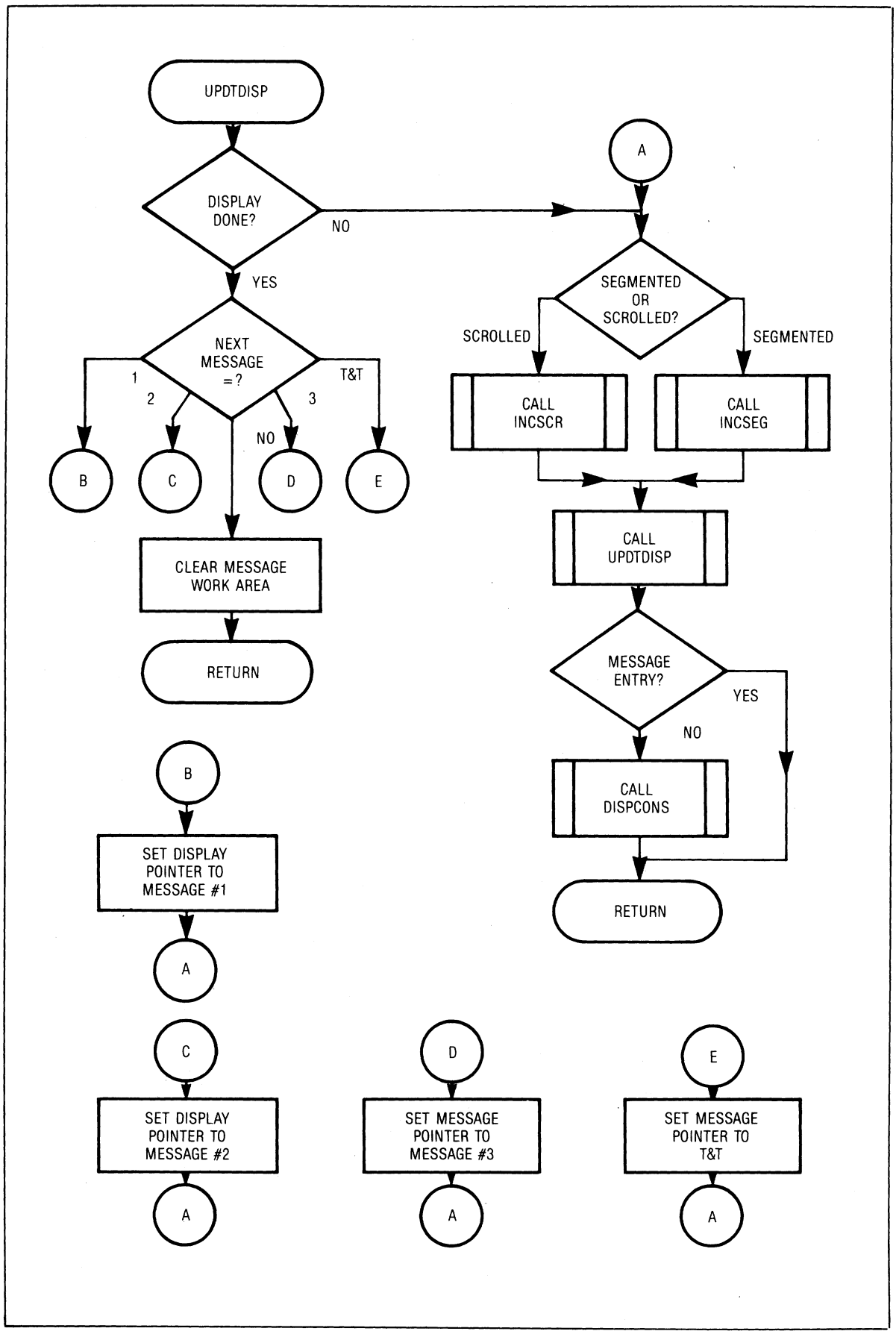


Figure 7. UPDTDISP flowchart.

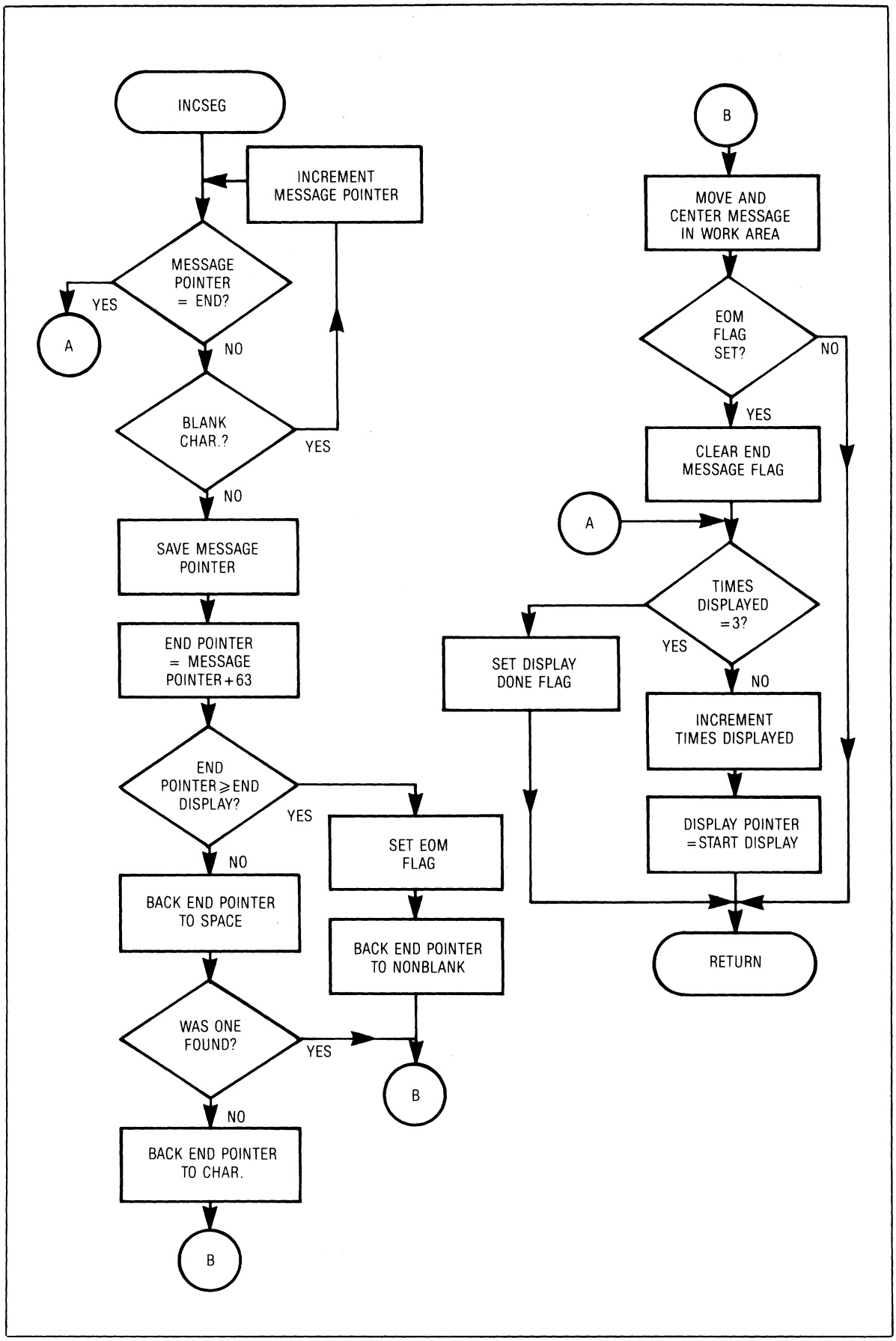


Figure 8. INCSEG flowchart.

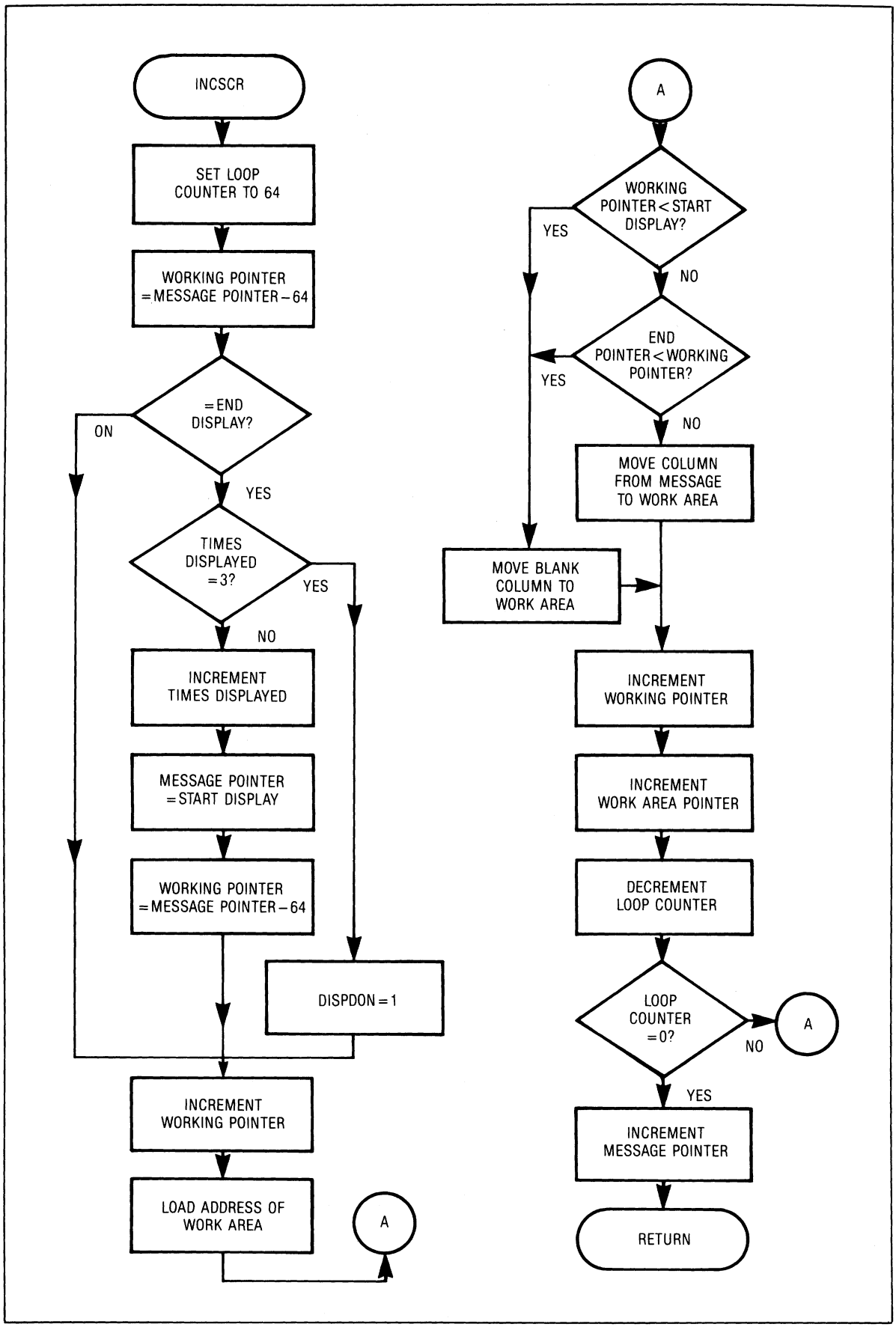


Figure 9. INCSCR flowchart.

message (up to five columns per character). These codes are transferred to the message storage area, where the message is assembled.

Time and temperature input handlers. Every time the main program finds the time-of-day clock input set, it calls the update-time subprogram (UPDTTIME). As the flowchart in Figure 13 shows, this subprogram uses four counters to store the current time. (CLKA, CLKB, CLKC, and CLKD represent tens of hours, hours, tens of minutes, and minutes, respectively.) The counters are incremented using modulo-six, -ten, or -twelve addition as appropriate. With the aid of routine MTSUB, UPDTTIME takes the current time data from CHARTAB3 and assembles it in the time-and-temperature message storage area.

When the time update is completed, another subprogram is called to update the temperature on the display (Figure 14). This routine, called UPDTTEMP, retrieves the 8-bit output of the temperature A/D converter from temporary storage in INPUT01. If the temperature has changed since the last temperature update, the routine updates the display accordingly. If there has been no change in temperature, control is returned to the main program.

To update the temperature display, the program must first translate the value from the A/D converter to a value representing the current temperature in degrees Fahrenheit and Celsius. This is done with a table (TEMPTAB) containing a 3-byte encoding of the temperature in degrees Fahrenheit and Celsius for each value output by the converter. These values are handled in a manner similar to UPDTTIME: calls to MTSUB move the display representation of the numbers to the message area.

Control software for the display processor. Compared to the main processor's software routines, the display processor's software is simple (Figure 15). The display processor performs only three tasks:

- inputting a string of 65 six-bit characters (64 encoded display column values, plus a display mode control character) from the USART,
- translating the encoded values into the actual display representation, and
- updating the screen display.

When execution of the display processor is initiated by the turn-on reset of the CPU, system memory, the display, and the USART are initialized by the program. Because the display is connected to the system as 64 I/O ports, with the display columns having different addresses, the output routine is transferred to read/write memory. This allows the program to change the I/O address specified by the output instruction.

The display processor next begins execution of the main program loop, synchronizing the USART to the main processor's transmitter. The USART inputs characters until a valid display mode control character is found. It then inputs 64 display "characters," which are decoded (using the character translation table TRANTB) and output to the display. This sequence repeats as long as the display processor is operating.

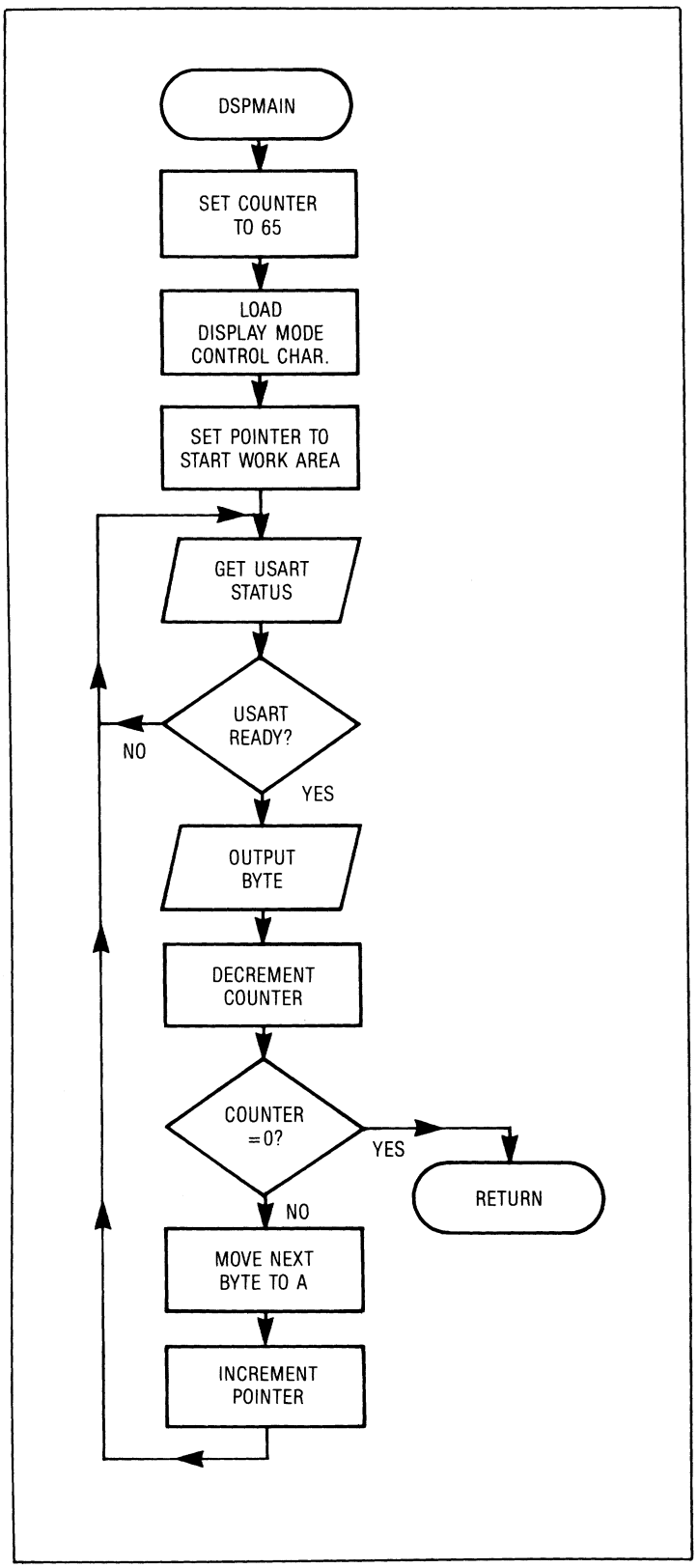


Figure 10. Display output routine flowchart.

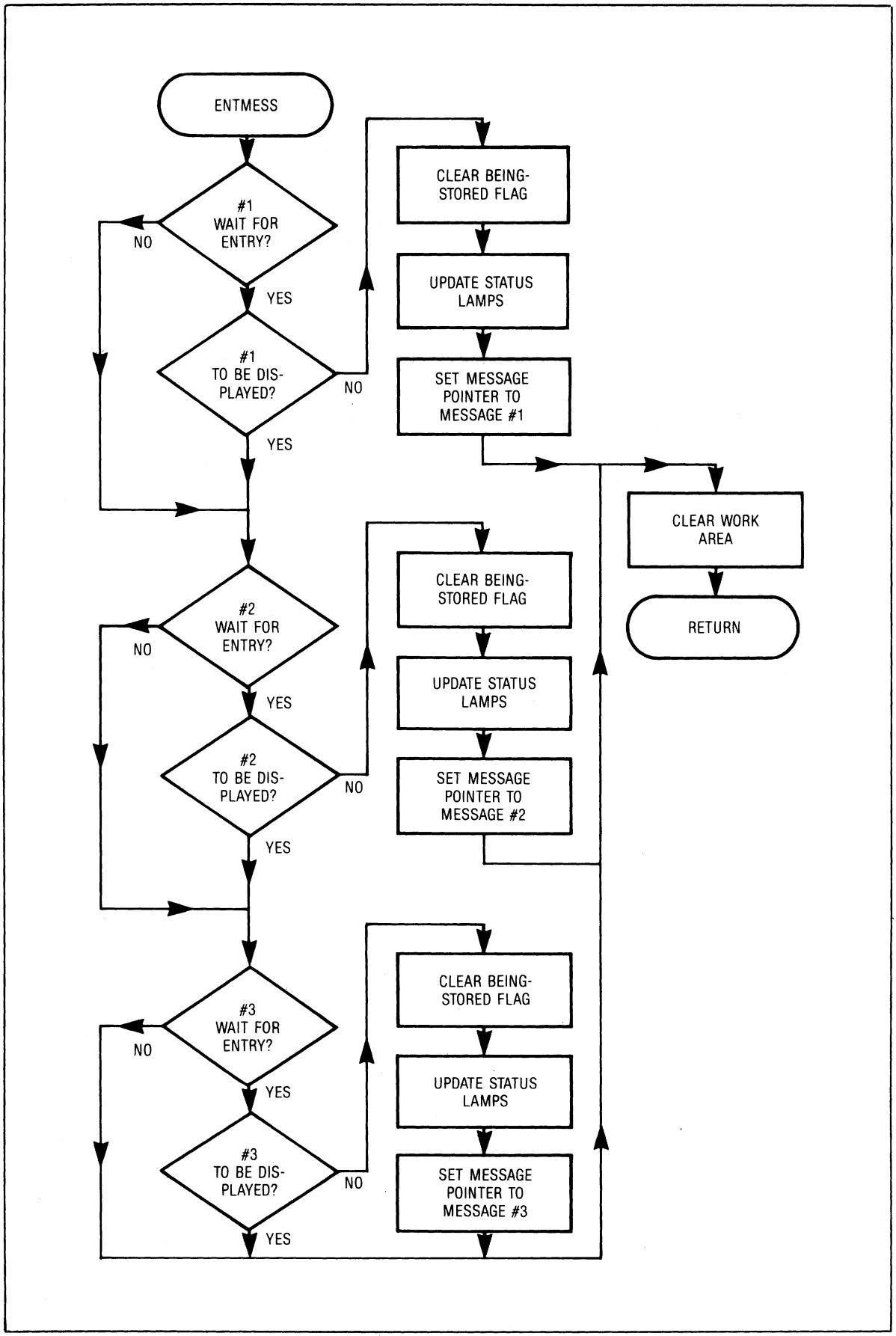


Figure 11. Enter message request handler flowchart.

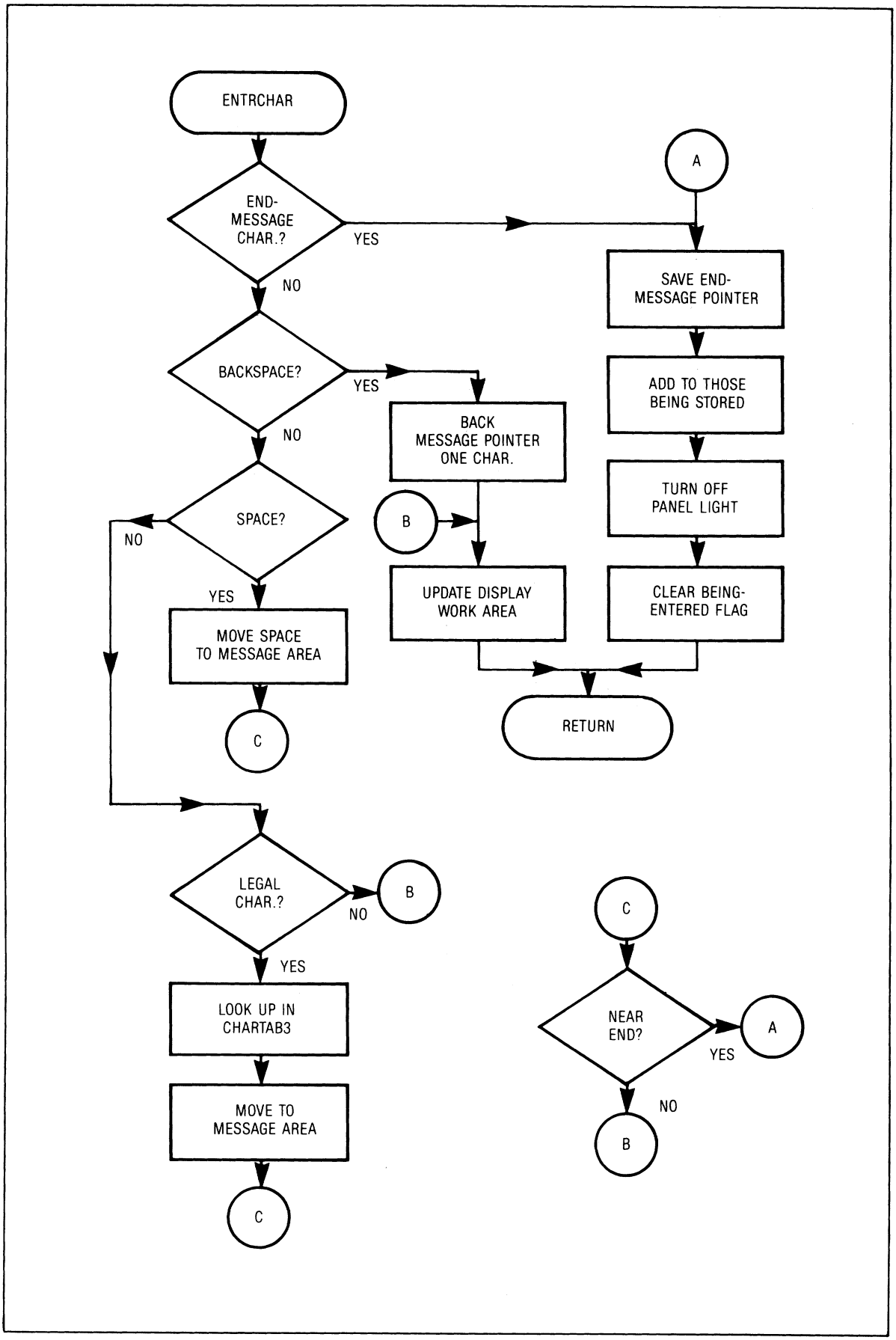


Figure 12. Flowchart of the ENTRCHAR routine.

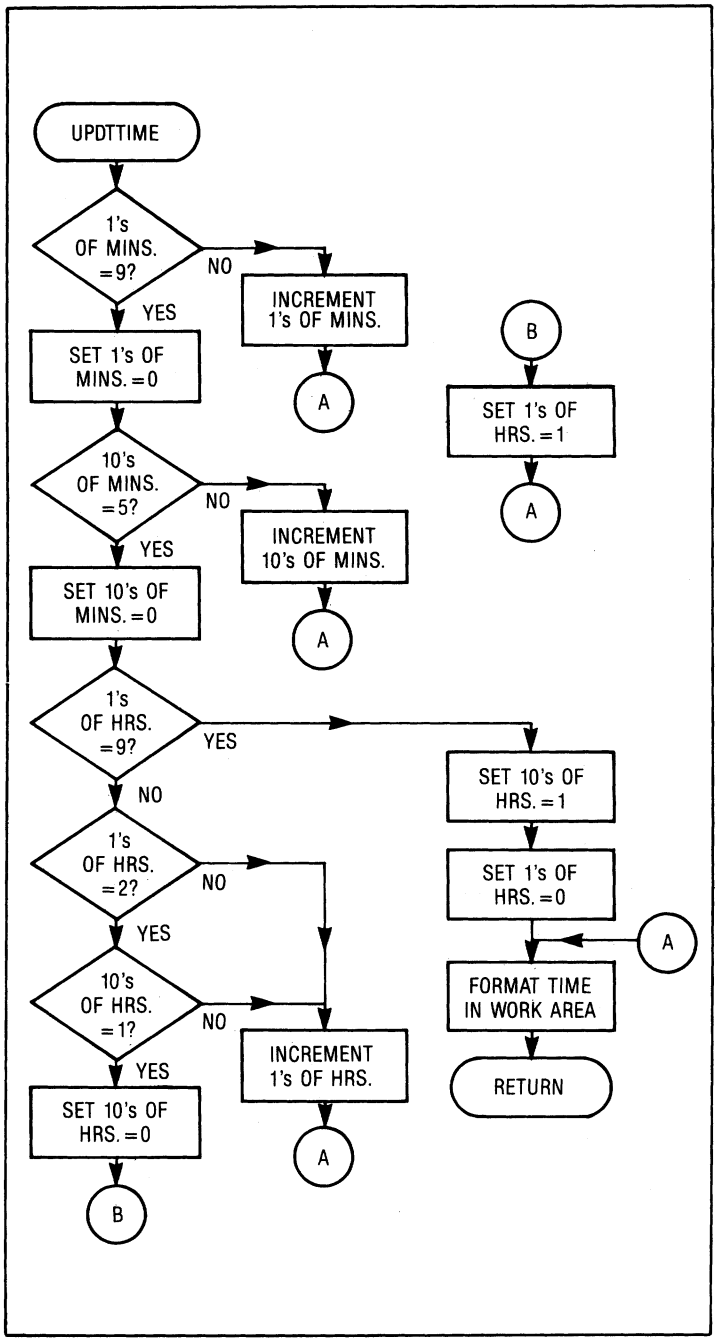


Figure 13. UPDTTIME flowchart.

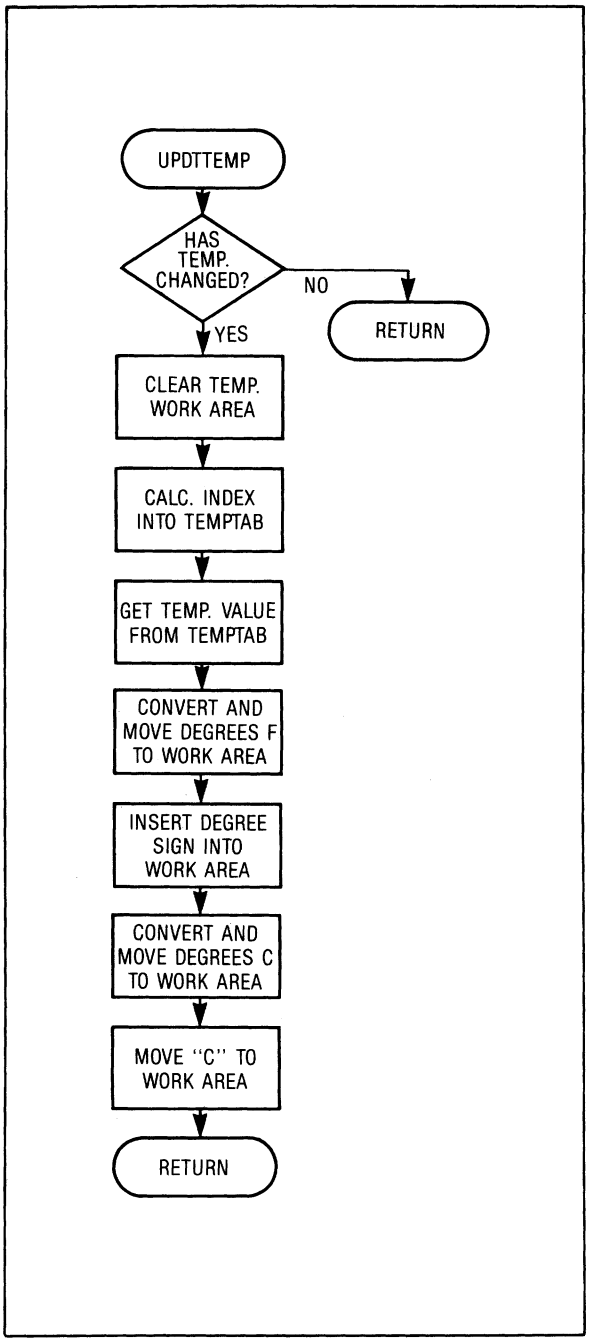


Figure 14. UPDTTEMP flowchart.

MCMDS: efficient and inexpensive

The Microprocessor-controlled Message Display System shows that microprocessors can be an efficient and cost-effective alternative to the electromechanical components normally used in message displays. The low-cost prototype will be even less expensive to produce in volume, giving it a substantial edge over similar message display systems currently in production.

The MCMDS is simple to operate and requires virtually no operator training. By eliminating mechanical components and adjustments, it should provide long-term reliability. ■

References

1. Adolph Mangirei, "Wire-Wrapping and Proto-System Techniques," *Byte,* Vol. 6, No. 5, May 1981, pp. 152-170.
2. *AD590 Two-Terminal IC Temperature Transducer,* Analog Devices Corporation, Norwood, MA, 1979.
3. *Linear Databook,* National Semiconductor Corporation, Santa Clara, CA, 1982.
4. *Optoelectronics Designer's Catalog,* Hewlett-Packard Corporation, Palo Alto, CA, 1981.
5. *Motorola Semiconductor Master Selection Guide,* Motorola, Inc., Phoenix, AZ, 1981.

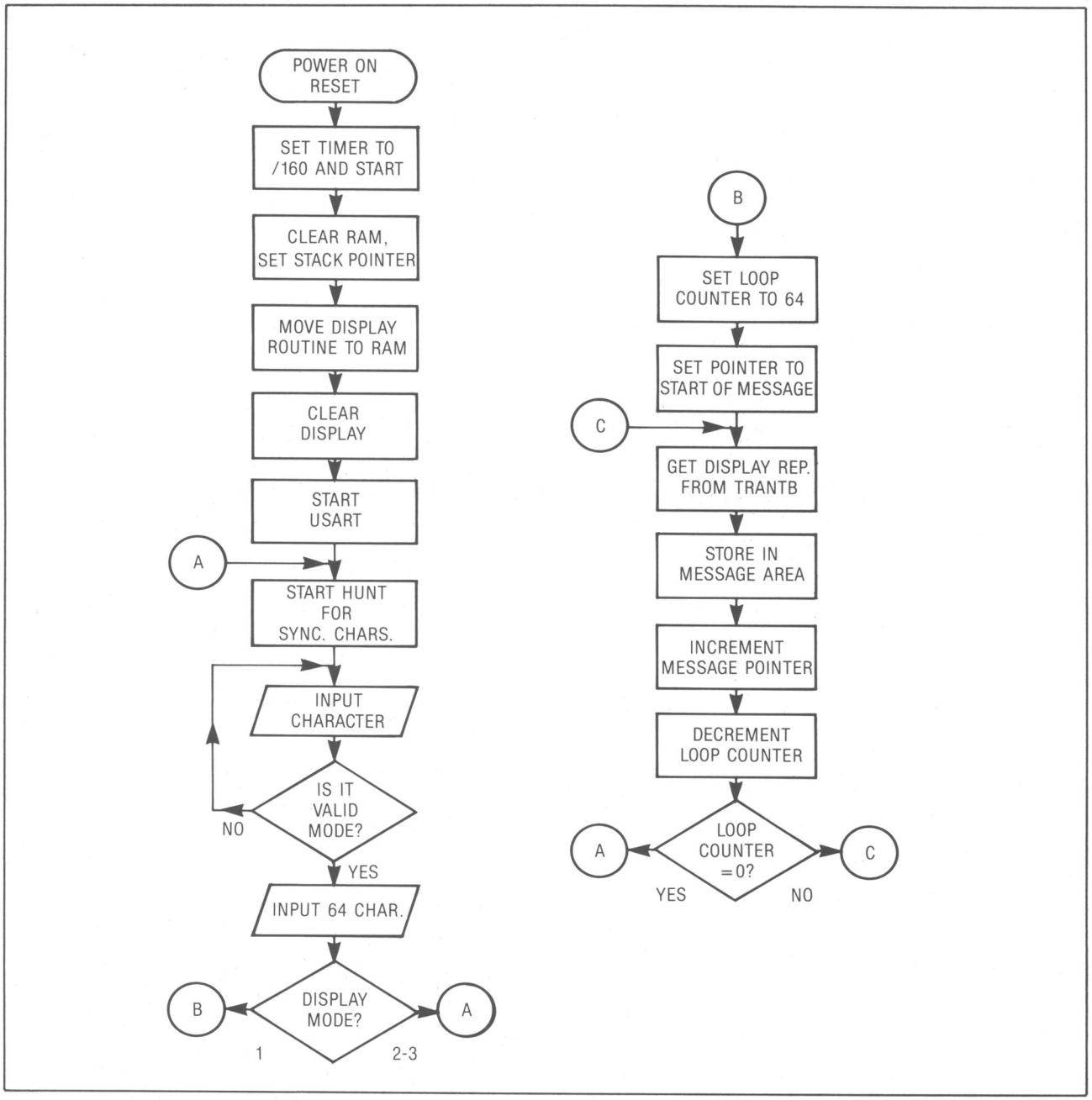
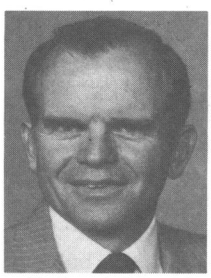


Figure 15. Flowchart of the display processor program.

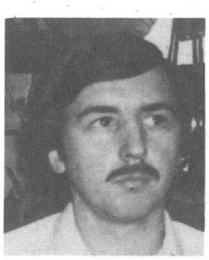


Paul D. Stigall is a professor of electrical engineering and computer science at the University of Missouri—Rolla, which he joined in 1970. His research interests include computer architecture and systems, digital signal processing, microprocessor and minicomputer applications, digital circuits, and fault-tolerant computing. He is the author of several technical publications and the principal investigator on several research projects.

Before joining the university, Stigall worked for McDonnell Douglas, the Navy Electronics Laboratory, and the Collins Radio Company. He was also an instructor at the University of Wyoming.

Stigall received the BSEE from the University of Missouri—Rolla in 1962 and the MS and PhD in electrical engineering from the University of Wyoming in 1965 and 1968. A registered professional engineer in Missouri, he is a member of ACM, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi, and a senior member of the IEEE.

Stigall's address is the Department of Electrical Engineering, University of Missouri—Rolla, Rolla, MO 65401.



Brian Lenharth has recently accepted a position with the Lake Stevens Division of Hewlett-Packard. While completing his MSEE requirements at the University of Missouri—Rolla, he was an engineer with the Department of the Army. He is interested in practical applications of microprocessors and in instrumentation development.