DESIGN OF A MICROCOMPUTER-BASED MICROPOROUS MEMBRANE PROCESS CONTROLLER

by

Douglas R. Browning

Thesis submitted to the Graduate Faculty

of the Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

APPROVED:

W. J. Arp, Chairman

D. G. Larsen

H. H. Robertshaw

March, 1985

Blacksburg, Virginia

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to the following people for their help and support in this research:

Dr. Leon Arp, for serving as my committee chairman, for contribution of his skills, ideas, and resourcefulness, and for his countless hours spent teaching, guiding and working with me to successfully complete this project.

Dr. Harry Robertshaw and Professor David Larsen for serving on my graduate committee.

for her friendship and support during the research and writing of this thesis.

Finally, my parents, , for their love, encouragement, and support which made my college education possible.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

I. INTRODUCTION

Problem Overview

A microcomputer-based controller has been developed to produce porous membrane material. The production process involves stretching the material in a temperature-controlled solvent bath. The materials and solvents usable in such a process are specified in the confidential industrial report "Microporous Membrane Production Process Outline" written by L. J. Arp, Professor of Mechanical Engineering, Virginia Polytechnic Institute and State University [1]. Preliminary tests have shown that the membrane produced by the process exhibits tiny "pores" in the material, the size of which is determined by the controlled variables. Quantifying the pore size under varying operating conditions is to be the subject of further research. Specific applications using the microporous material are expected to be blood oxygenation, kidney dialysis, particle filtration, and salt water desalination.

The variables to be controlled for this process are the rate of stretch of the material in terms of output length divided by input length over a finite time interval (stretch-ratio), the output feed rate, and the temperature of the solvent. The process is to be continuous-input, continous-output. The stretch-ratio and feed rate parameters can be controlled using two motors operating at two known but different speeds. For future research purposes, a wide control range for all controlled variables is desired.

Review of Control Strategies

Electronic control of dynamic systems is possible using dedicated circuitry, computer algorithms, or a combination of both. In such a combination, the computer is used to direct and monitor a process, while the dedicated circuitry frees the computer's processing unit of unnecessary calculations. On the other hand, an isolated dedicated circuit has no way of receiving computer-generated set-points.

Computer algorithms have some serious disadvantages. A computer control algorithm introduces a delay between sampling a system variable and delivering the control signal because of computer calculation time. This delay will undesirably affect the behavior of a quickly responding system. A computer algorithm also wastes processor time when calculating a control signal output. In addition, and most important to this project, most first-order computer control algorithms require knowledge of system gain and time constant [2]. The variables of temperature and motor velocity can both be modeled as first-order systems. Since the temperature time constant of the processing system may vary due to construction alternatives of the processor enclosure, an algorithm may not be suitable. Therefore, a combination of dedicated circuitry under computer control solves the problems associated with control algorithms and isolated dedicated circuitry.

Principle of Problem Solution

An interactive computer controller was selected as the means to control the process because it is a safe and effective way to direct and

monitor process variables. A dedicated motor controller receiving velocity set-point information from the computer was considered to be the fastest responding method to load disturbances requiring the least processor time for speed control. A hardware/software approach which bases a control signal only on error from desired set-point (proportional control) value was selected as the method for controlling system temperature. This temperature control scheme was chosen for its ability to be "tuned" to almost any control volume by varying the heater power gain. In the standard feedback control loop, this is equivalent to choosing a forward loop controller gain (heater power) which gives an acceptable steady-state response. Higher gains introduce more oscillation around the steady state value but reduce steady state error while lower gains tend to reduce steady state oscillations and increase steady state error. The objective of this research was to design the process controller only, not the entire system.

II. DESIGN OVERVIEW

The flow diagram for the microcomputer-based process controller is presented in Fig. 1.

Material to be processed is fed through an enclosed control volume which contains the saturated solvent mixture, temperature sensors, and heater coils. Conditions within the enclosure are monitored, displayed and controlled by the computer. As shown in the figure, temperature sensors relay temperature information to the computer and the computer control signals drive the heaters.

Two motors, each one rotating at a different speed, determine the output feed rate and resultant stretch-ratio of the processed material. The two motor controllers are proprietary dedicated circuits which maintain constant motor shaft velocity determined from a given 8-bit input from the computer. With motor 2 rotating at a selectable, faster rate than motor 1, a desired feed rate and stretch-ratio for the process can be obtained. Worm gear speed reducers connected to each motor shaft allow for low motor speed operation as well as preventing motor 2 from "pulling" motor 1 up to its speed via the material being processed. Thus, tension is maintained in the material drive system through the worm gears. In addition, high friction pulley systems attached to the output shafts of the gear reducers prevent slippage of material during the stretching process.

An assembly language computer program controls the entire processing sequence. A keyboard is utilized for user selection of operating parameters as well as independent control of individual

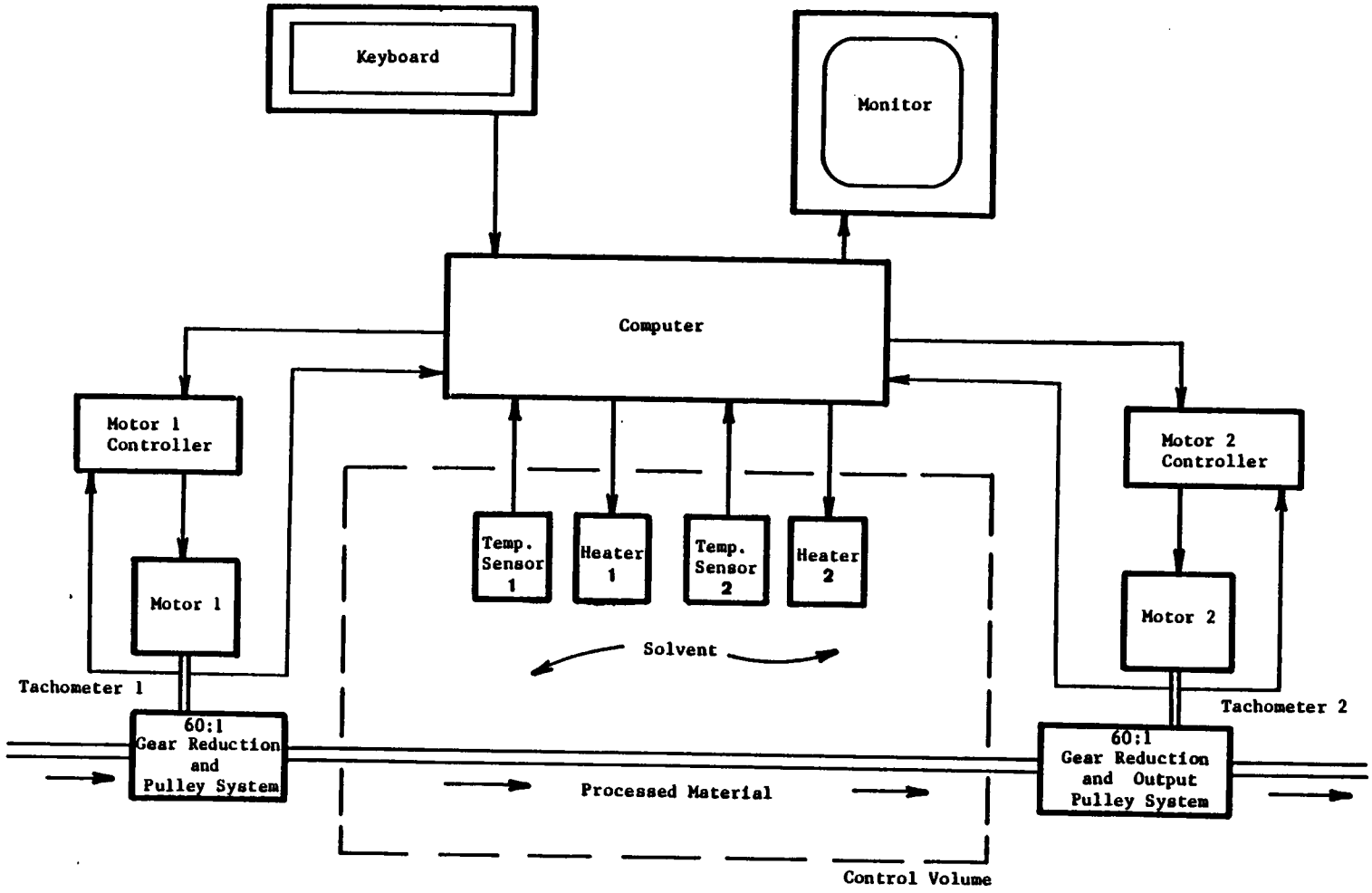


Figure 1. Microporous Membrane Process Controller

heaters and motors. During a process run, material feed rate and stretch-ratio are measured and displayed on the system monitor. Temperatures within the controlled system are also monitored and displayed. An audible and visual alarm occurs whenever system temperatures become 10° C greater than the desired set points. At this time, both motors and heaters are turned off. The process continues to run until either the operator shuts it down or an alarm condition occurs.

III. HARDWARE DESIGN

Microcomputer Selection

The central electronic component of the process controller is the computer. A Commodore VIC 20 microcomputer was chosen for this purpose since it consolidates several features necessary for development of the controller. A schematic of the VIC 20 is included in Appendix E.

The first and most important feature of the VIC 20 is that its central processing unit (CPU) is the same as the one in the AIM 65 microcomputer used for software development and program storage on integrated circuit chips. The CPU in both of these computers is the 6502 microprocessor. Other microcomputers such as APPLE, PET, KIM, and TIM, use this device. Since the AIM 65 microcomputer was used in this project for software development and erasable-programmable read-only memory (EPROM) programming, the VIC 20 can directly run an AIM 65-programmed algorithm.

Besides processor-compatibility, the VIC 20 features a built-in keyboard. This allows implementation of an interactive program to guide the user through proper process preparation channels. Also, the VIC 20 contains user-callable assembly language subroutines which are already part of the operating system for input/output (I/O), access to the system clock, and memory management. These routines greatly simplify the programming task of keyboard I/O.

The VIC 20 includes a video interface chip (6560) designed for color video graphics applications, such as for low cost cathode ray tube

(CRT) terminals. This feature enables the computer to prompt the user for keyboard input, display error or alarm messages, and display measured control system variables during process operation.

Interfacing the VIC 20 to external control circuitry is straightforward with the aid of the 44 pin expansion connector. All necessary address, data, and control bus lines are available there.

A powerful programming tool is also available for the VIC 20, known as the VIC MON.® VIC MON is Commodore's 6502 Machine Language Monitor cartridge for the VIC 20. With this cartridge the programmer can write and run assembly language programs on the VIC 20. The length of these programs, however, are limited to the amount of random access memory (RAM) available in the computer.

In spite of this limitation, the VIC 20 is quite a versatile microcomputer, and is one of the least expensive microcomputer systems available at this time. This was a major consideration when the decision was made to buy a computer rather than build one for the project.

Microcomputer Overview

The basic organization of the microcomputer system is presented in Fig. 2. Dotted lines separate the VIC 20 system from added interface and control circuitry.

The Motorola 6502A microprocessor serves as the central processing unit (CPU) of the system. This particular microprocessor incorporates a 16-bit address bus, an 8-bit bidirectional data bus, two types of

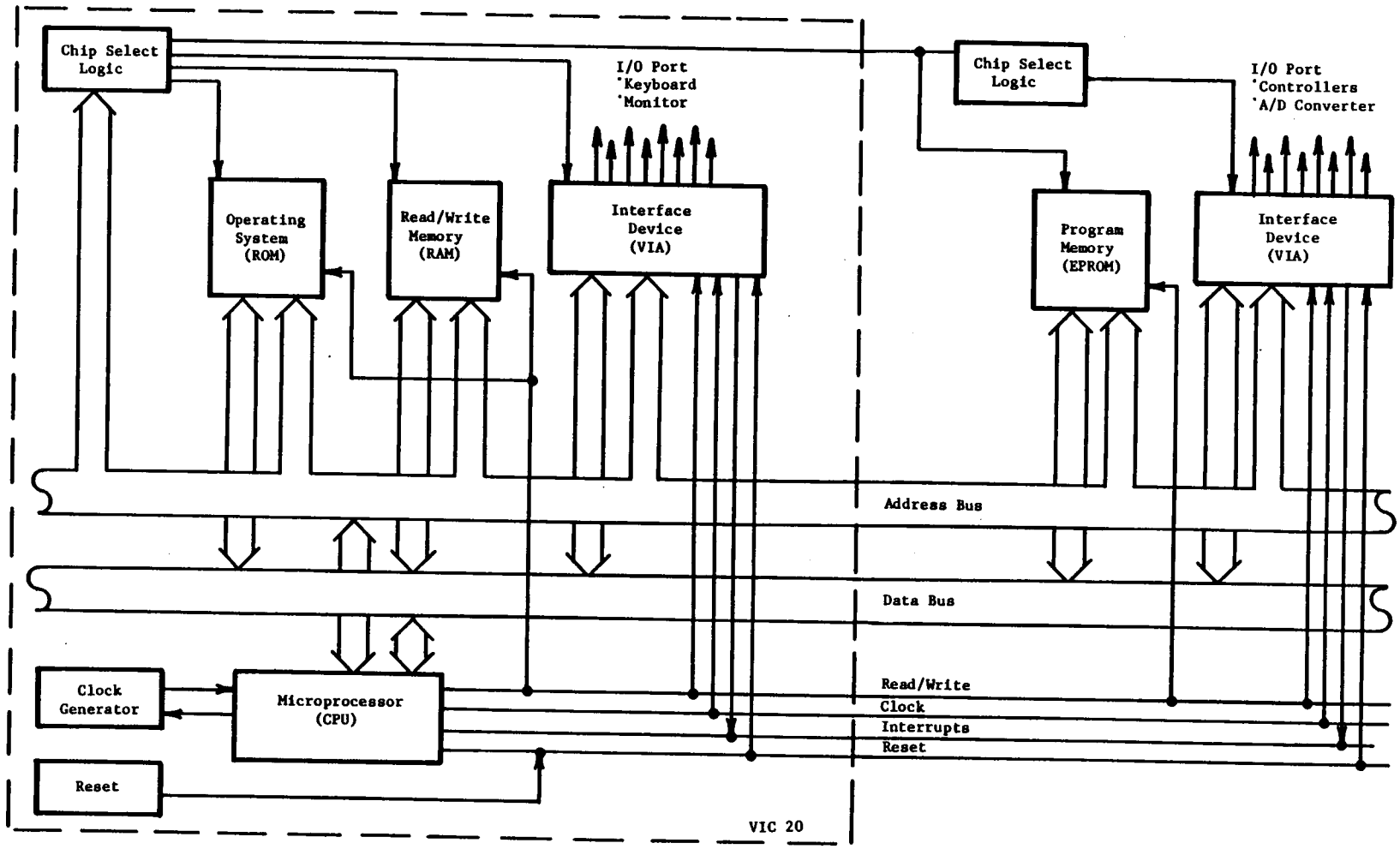


Figure 2. Basic Microcomputer Organization

interrupts, and operates on a single 5-volt power supply. The 6502A microprocessor is a member of the R6500 family of microprocessors and support devices, and all R6500 products are compatible with the Motorola M6800 bus structure [3].

As shown in Fig. 2, the microprocessor interacts with many other devices. Most of this communication occurs on the address and data buses. The address bus is unidirectional and is 16 bits wide. It is used to transfer addresses generated by the microprocessor to the memory and interface devices. A 16-bit binary address permits up to 65, 536 memory cells or words to be accessed. Each memory cell or word consists of 8 bits, called a "byte". The data bus transfers these bytes bi-directionally to and from the microprocessor. The READ/WRITE control signal from the microprocessor specifies the direction of transfer for the support devices.

Data transfer is enabled by three hardware subsystems which are the reset hardware, the clock generator and the chip select logic. Figure 2 shows that the VIC 20 contains all three. This not surprising since the VIC 20 can be operated independent of the interface and control hardware. A schematic of the VIC 20 is included in Appendix E.

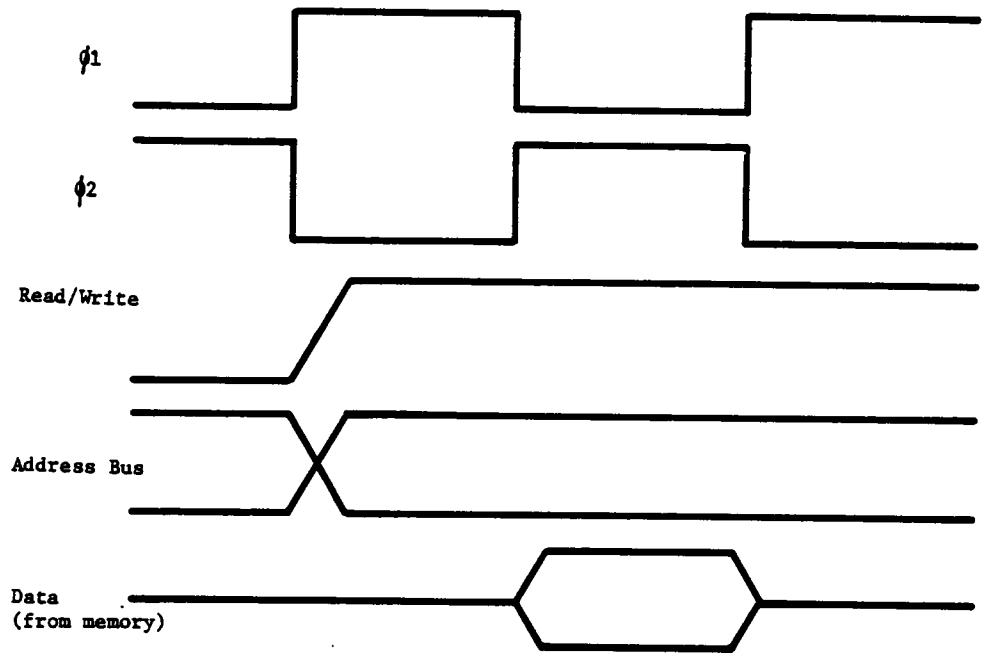
The reset hardware delays operation of the microprocessor and all interface devices during power-up until supply voltages and clock signals have stabilized. This ensures that computer operation does not begin until the microprocessor's first instruction is ready to be executed.

The clock generator produces two constant frequency waveforms used

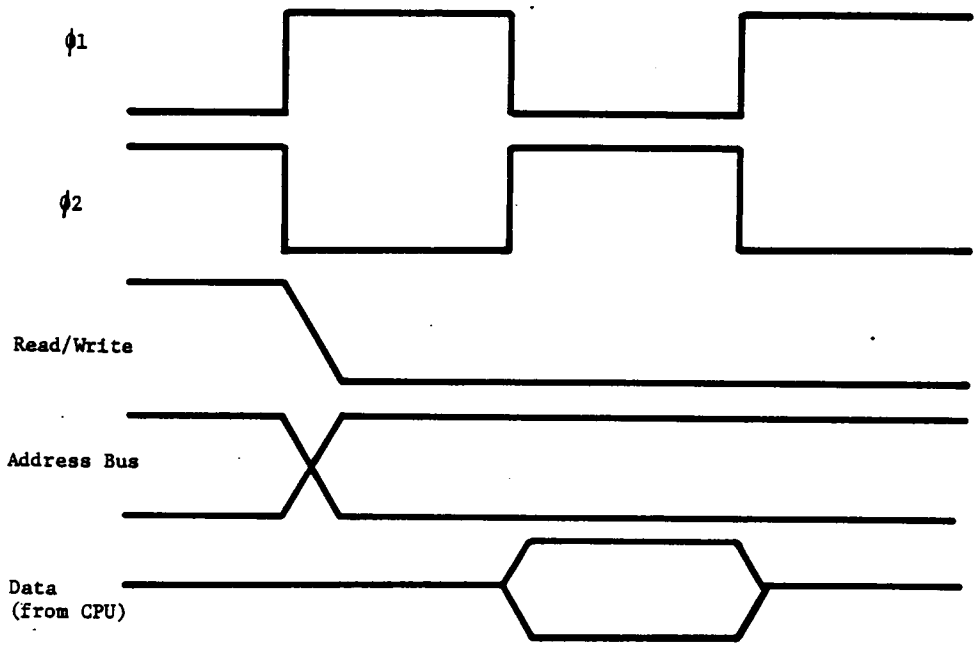
by the CPU and peripheral devices as a time base for controlling all signal transitions. Address lines change and the READ/WRITE signal stabilizes during the positive half cycle of the "phase 1" (ϕ1) square wave, and all data transfers occur during the positive half cycle of the "phase 2" (ϕ2) square wave. Therefore, these non-overlapping square waves ensure that all READ/WRITE and address lines are stable during data transfer. Figure 3 illustrates the timing relationships between the clock, buses, and READ/WRITE signals for memory read and memory write instructions.

Chip select logic determines which memory or interface chip is accessed during the execution of an instruction. The VIC 20 contains all the necessary logic to select or "turn on" its own chips at the appropriate time. However, aditional logic is required to access the added interface, control, and peripheral hardware. This is illustrated in Fig. 2. The chip select logic decodes the 16-bit address bus and enables only the appropriately addressed chips. Using this fact, the circuit designer can create the chip select logic to assign device addresses as he chooses.

During operation, the microprocessor accesses three groups of devices--program memory, READ/WRITE memory, and interface devices. Chip select logic determines when these devices are accessed. The program memory is usually either a pre-programmed read-only memory (ROM) or an erasable-programmable read-only memory (EPROM). In either case, the program is retained in the device when power is turned off. Read/Write or random-access memory (RAM) will not retain program memory when power



a) Memory Read Operation



b) Memory Write Operation

Figure 3. Timing Diagrams for Memory Read and Memory Write Operations

is turned off. It is typically used by the microprocessor as a "scratchpad" for calculations and as a temporary storage area. Interface devices are usually integrated circuit chips or electronic packages which allow the microcomputer to communicate with or control other hardware.

Interrupt processing is another important feature of the microcomputer system. Interrupt requests are signals sent to the microprocessor which are generated by an external component, usually an interface device. These requests are processed immediately after the microprocessor recognizes it has received an interrupt signal. The computer usually responds to an interrupt request by executing a short routine and then returning to the original program. With this feature, events requiring immediate attention may be attended to only when necessary [4].

VIC 20 System Review

Before discussing the control and interface circuitry used in this design, a description of the VIC 20 from a hardware as well as software approach is necessary.

The VIC 20 is a versatile, complete microcomputer. The system schematic is included in Appendix E. During power-up, the 6502A microcomputer is reset by a 555 timer chip. The 555 active-low reset signal is also used to initialize all interface chips.

The clock generator for the microcomputer is a quartz crystal oscillator. A 14.31818 MHz quartz crystal is used to clock the 6560

video interface chip which controls CRT screen I/O. This particular chip also supplies the 6502A microprocessor with a 1.000 MHz signal to produce the phase 1 (ϕ1) and phase 2 (ϕ2) square waves.

System Memory

Three 74LS138 integrated circuit (IC) chips provide the chip select logic for RAM, ROM, and the interface devices. Included in the VIC 20's ROM is the computer's operating system, or Kernal. The Kernal is comprised of routines which control all communications between the processor and other interface devices, i.e., the keyboard, the CRT, cassette tape, etc. In addition, the Kernal includes over 25 machine language subroutines which may be used by the programmer for keyboard or cassette tape I/O, access to the system clock, and memory management [5].

A Basic interpreter is also in the VIC 20 as read-only memory. This interpreter translates a high level Basic program into a series of machine code routines which perform the function required by each Basic command [6].

The third section of read-only memory in the VIC 20 is the character generator ROM. The character generator is a 4K ROM (1K = 1024 bytes) which contains the pattern of dots used to display each of the valid ASCII characters on the CRT screen [6].

The VIC 20 also includes a limited amount of random access memory (RAM) for temporary storage of user programs. There is 4K (4096 bytes) of RAM available on the unexpanded VIC 20.

A memory map of the VIC 20 is shown in Fig. 4. This figure presents the addressable locations of memory relative to each other. Since there are 16 lines comprising the address bus, the 6502A microprocessor is capable of accessing 64K (65,536 bytes) of memory. As shown in Fig. 4, only 29K of the available 64K is utilized: 8K for Kernal ROM, 8K for Basic ROM, 4K for character generator ROM, 4K RAM for user memory, 1K RAM for storage of Kernal and Basic variables, 1K RAM for storage of screen charater and character color information, and 3K for various interface devices such as the 6560 video interface chip and 6522 versatile interface adapters. The hexadecimal addresses of each section of memory are symbolized with the dollar sign prefix, $.

Power-up and Initialization

When the VIC 20 is switched on, a pre-defined initialization sequence stored in the Kernal ROM is executed. This initialization sequence is included in Appendix C.

On power-up, the microprocessor's program counter is loaded with the contents of $FFFC and $FFFD, which is the address of the initialization sequence, $FD22. Program execution then continues from this location in the Kernal ROM. First, interrupt processing is disabled and the stack pointer is set to $00FF. The stack pointer is an area of RAM where the CPU temporarily stores processor values, such as the current value in the program counter or the contents of a register. After the stack pointer is initialized, the Kernal checks for the presence of ROM at $A000. The Kernal looks in memory beginning at

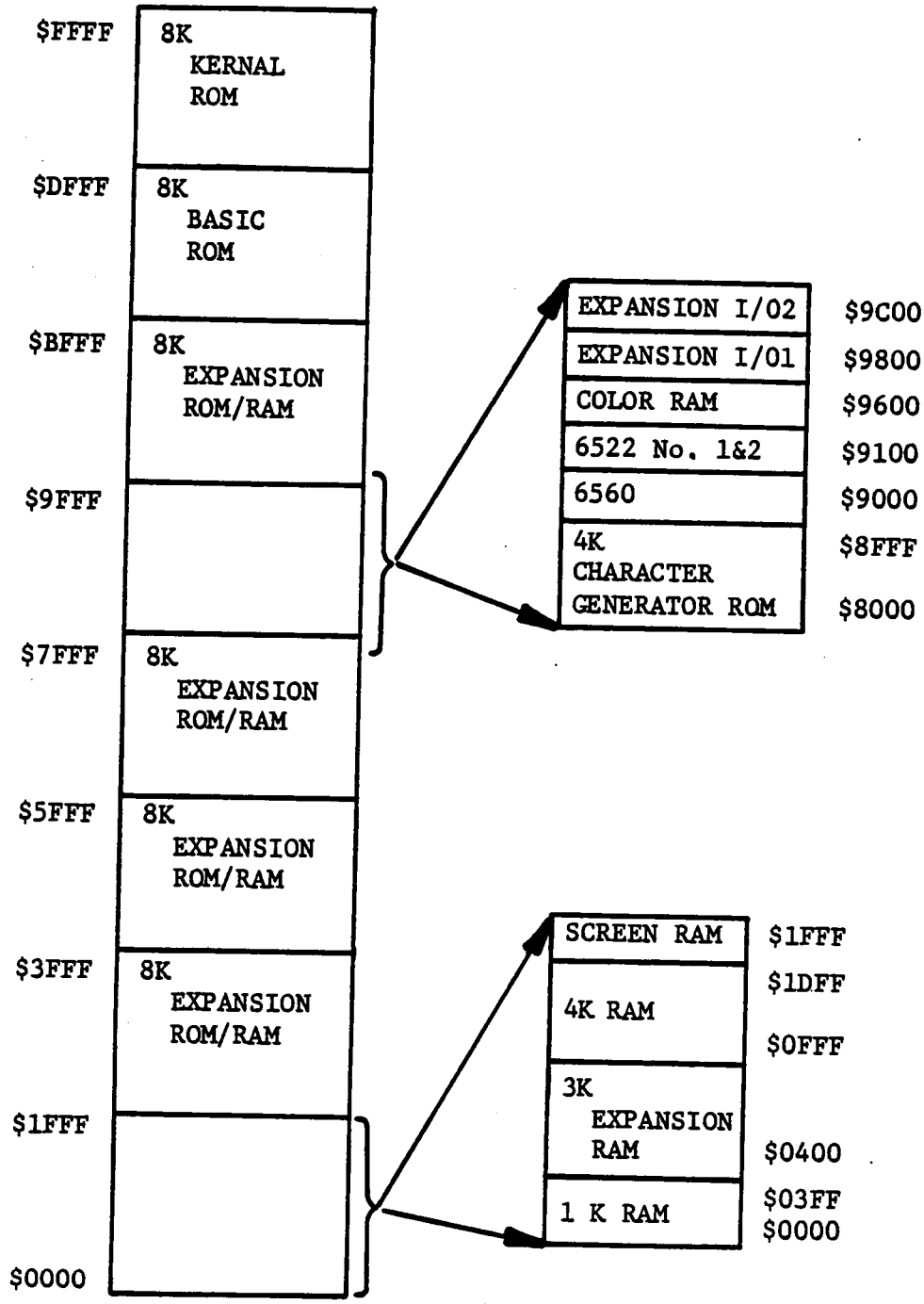


Figure 4. VIC 20 Memory Map

$A004 for the auto-start ROM sequence:

Address -	$A004	Contents -	$41
	$A005		$30
	$A006		$C3
	$A007		$C2
	$A008		$CD.

If this sequence is present, the 16-bit program counter (PC) is loaded with the contents of locations $A001 (PC high byte) and $A000 (PC low byte). Program execution then continues from the new location in the program counter. If the auto-start sequence is not found, the normal initialization routine (located at $FD2F) is executed. This routine sets up all interrupt and jump vectors, initializes the 6522 I/O devices and the video chip (6560), and then jumps to the start of Basic at $C000 [6].

This start routine is typical of the flexibility of the VIC 20. It allows the programmer to power-up into his own routine simply by wiring a ROM or EPROM, programmed with the auto start sequence, to the VIC at $A000. Note, however, that the normal initialization routine has been bypassed with this method. The programmer must therefore initialize all devices and vectors by calling the appropriate subroutines (see Appendix C) if he plans to use the VIC 20 I/O hardware or Kernal software.

Memory Expansion

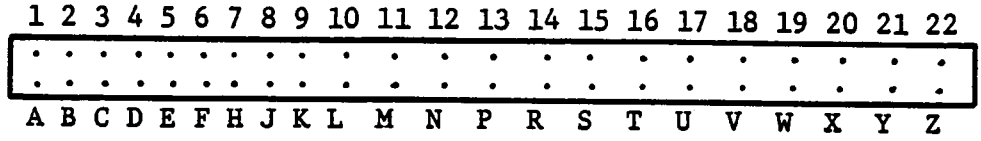
In order to connect a ROM, EPROM, RAM or any other addressable device to the VIC 20, the memory expansion connector must be used. This 44-pin (22/22) female edge connector is located on the back of the VIC 20. Figure 5 shows the location and type of each pin on the connector. Table 1, which follows, describes the function of each pin.

A brief discussion of the kinds of lines available at the expansion connector will help explain their usefullness for interfacing other devices to the VIC 20. The memory expansion port can be divided into five groups: the data bus, address bus, control bus, the block select, and power lines.

The eight data lines (CD0-CD7) are used to transfer data between the processor and memory. The address bus controls which memory location or I/O device receives or sends the data from or to the processor. Only 14 address lines (CA0-CA13) appear on the connector because the two most significant lines are decoded into the block select signals.

The block select lines include all $\overline{\text{RAM}}$, $\overline{\text{BLK}}$, and $\overline{\text{I/O}}$ signals. These are the chip select logic signals available to the system designer for direct use or for further decoding into smaller memory blocks. Each group of block select signals has been decoded by a separate 74LS138 3-to-8 line decoder chip. See Appendix E for further details.

The $\overline{\text{RAM}}$ signals, $\overline{\text{RAM}}$ 1, 2, and 3, are active-low signals which decode 1K of memory each. These lines will usually be used to access RAM to provide a larger block of continuous memory for the VIC 20.



Pin #	Type
1	GND
2	CD0
3	CD1
4	CD2
5	CD3
6	CD4
7	CD5
8	CD6
9	CD7
10	$\overline{BLK1}$
11	$\overline{BLK2}$

Pin #	Type
12	$\overline{BLK3}$
13	$\overline{BLK5}$
14	$\overline{RAM1}$
15	$\overline{RAM2}$
16	$\overline{RAM3}$
17	VR/W
18	CR/W
19	$\overline{IRQ}$
20	NC
21	+5V
22	GND

Pin #	Type
A	GND
B	CA0
C	CA1
D	CA2
E	CA3
F	CA4
H	CA5
J	CA6
K	CA7
L	CA8
M	CA9

Pin #	Type
N	CA10
P	CA11
R	CA12
S	CA13
T	$\overline{I/O2}$
U	$\overline{I/O3}$
V	SO2
W	$\overline{NMI}$
X	RESET
Y	NC
Z	GND

Figure 5. Location and Type of Pins on the Memory Expansion Connector

Table 1. Memory Expansion Connector Pin Functions

Type	Pin #	Description
GND	1	System Ground
(D0-CD7)	2-9	Data bus lines 0-7
$\overline{\text{BLK1}}$	10	8 k decoded RAM/ROM block 1, starting at $2000 (active low)
$\overline{\text{BLK2}}$	11	8 K decoded RAM/ROM block 2, starting at $4000 (active Low)
$\overline{\text{BLK3}}$	12	8 K decoded RAM/ROM block 3, starting at $6000 (active low)
$\overline{\text{BLK5}}$	13	8 K decoded RAM/ROM block 5, starting at $A000 (active low)
$\overline{\text{RAM1}}$	14	1 K decoded RAM at $0400 (active low)
$\overline{\text{RAM2}}$	15	1 K decoded RAM at $0800 (active low)
$\overline{\text{RAM3}}$	16	1 K decoded RAM at $0C00 (active low)
VR/W	17	Read/Write line from buffer chip, (high=read, low=write).
CR/W	18	Read/Write line from CPU, (high=read, low=write)
$\overline{\text{IRQ}}$	19	6502 A Interrupt request line, (active low)
(NC)	20	
+5V	21	+5 volt power line
GND	22	System Ground
GND	A	System Ground
CA0-CA13	B-S	Address bus lines 0-13
$\overline{\text{I/O1}}$	T	Decoded I/O block 2, starting at $9800
$\overline{\text{I/O3}}$	U	Decoded I/O block 3, starting at $9C00
S02	V	Phase 2 system clock
$\overline{\text{NMI}}$	W	6502A non maskable interrupt, (active low)
$\overline{\text{RESET}}$	X	6502A reset line (active low)
(NC)	Y	
GND	Z	System Ground

The $\overline{\text{BLK}}$ signals, $\overline{\text{BLK}}$ 1, 2, 3, and 5, are also active-low signals. They decode memory into 4 blocks of 8K which can be used for ROM or RAM. Notice that $\overline{\text{BLK 5}}$ begins at $A000. This line is used by the designer to power up into his own program.

The $\overline{\text{I/O}}$ signals, $\overline{\text{I/O}}$ 2 and 3, can be used to add additional I/O devices to the VIC 20, as well as access smaller sections of added memory.

The expansion connector also includes six control lines; VR/W, CR/W, $\overline{\text{IRQ}}$, S02, $\overline{\text{NMI}}$, and $\overline{\text{RESET}}$. Five of these lines connect to or originate directly from the 6502A microprocessor. They are CR/W, $\overline{\text{IRQ}}$, S02, $\overline{\text{NMI}}$, and $\overline{\text{RESET}}$. The $\overline{\text{RESET}}$, $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ lines are inputs to the 6502A. The $\overline{\text{RESET}}$ line is an active-low pulse from a 555 timer IC which is used to reset the 6502A and interface devices. The $\overline{\text{NMI}}$ line, when grounded, signals the microprocessor to execute an interrupt routine. In the VIC 20, this routine acts similarly to a power-up condition; all I/O are reinitialized. The $\overline{\text{IRQ}}$ line is also used for interrupt requests. However, unlike the $\overline{\text{NMI}}$, the $\overline{\text{IRQ}}$ line can be programmed to be ignored by the microprocessor. In the VIC 20, the $\overline{\text{IRQ}}$ servicing routine scans the keyboard for a pressed key and updates the system clock. The other two lines directly connected to the CPU are CR/W and S02. Both of these are outputs of the microprocessor. The CR/W signal informs the memory or the device being addressed whether the microprocessor wants to write data or read data. The S02 signal is the 1.000 MHz system clock. The only control line not directly attached to the 6502A is the VR/W signal. This signal is the

same as the CR/W signal except it is buffered by the system clock and is enabled only during a positive pulse of S02. Memory expansion will normally use the VR/W signal. Other devices such as the 6522 IC's need the CR/W signal.

The final group of signals available at the memory expansion connector is the power. The power available is +5 volts and ground; the current rating is approximately 750 ma.

Interrupt Processing

Interrupt processing on the VIC 20 is the last topic to be discussed before describing the interface and control circuitry. After system initialization, one of the 6522 versatile interface adapter (VIA) IC's produces interrupt requests ($\overline{IRQ}$) at a rate of 60 per second. If the $\overline{IRQ}$ enable flag is low, the 6502A will respond to the request by servicing the $\overline{IRQ}$ interrupt routine. During this routine, the VIC 20 scans the keyboard for a pressed key and updates the system clock. How the processor actually accesses this routine is what is important to the programmer or system designer. Once this process is understood, the designer may alter the interrupt servicing sequence.

When an interrupt request occurs on $\overline{IRQ}$, the instruction currently being executed by the CPU is completed. If the $\overline{IRQ}$ disable flag (bit 2) in the processor status register is set, the interrupt request is ignored and program execution continues. If the $\overline{IRQ}$ disable flag is a zero, then it is immediately set to prevent further interrupts while the existing one is being processed. Next, the current value in the program

counter is stored on the stack, and then the contents of the processor status register are saved on the stack. The processor then reads the contents of locations $FFFE and $FFFF for the value of the new program counter. Program execution then begins at this new location in the program counter, which is the interrupt servicing routine [7]. On the VIC 20, the address contained in $FFFE and $FFFF is $FF72, which is part of the Kernal ROM. At $FF72, a short routine saves the x,y and accumulator registers on the stack. Program flow then jumps to the address contained in location $0314 (new PC low) and $0315 (new PC high). These two locations are in system RAM which allows the designer to alter the interrupt processing sequence to suit his own needs. All the designer need do is write his own interrupt routine starting addresses into locations $0314 and $0315.

VIC 20 Interface Circuitry

The interface circuitry for this research project connects the EPROM circuitry and the temperature and motor control devices to the VIC 20. This circuitry buffers all lines used from the memory expansion connector as well as provides the chip select logic for the EPROM chips and interface devices.

Figure 6 shows the arrangement of the 74245 buffers. These chips are bi-directional bus transceivers with 3-state outputs. Typical propogation delay time, port-to-port, is 8 ns. Notice that 74245-3 is enabled only when $\overline{BLK5}$ or $\overline{BLK2}$ or $\overline{BLK1}$ or $\overline{I/O2}$ is low and S02 is high. This logic ensures that the data bus to/from the memory and interface

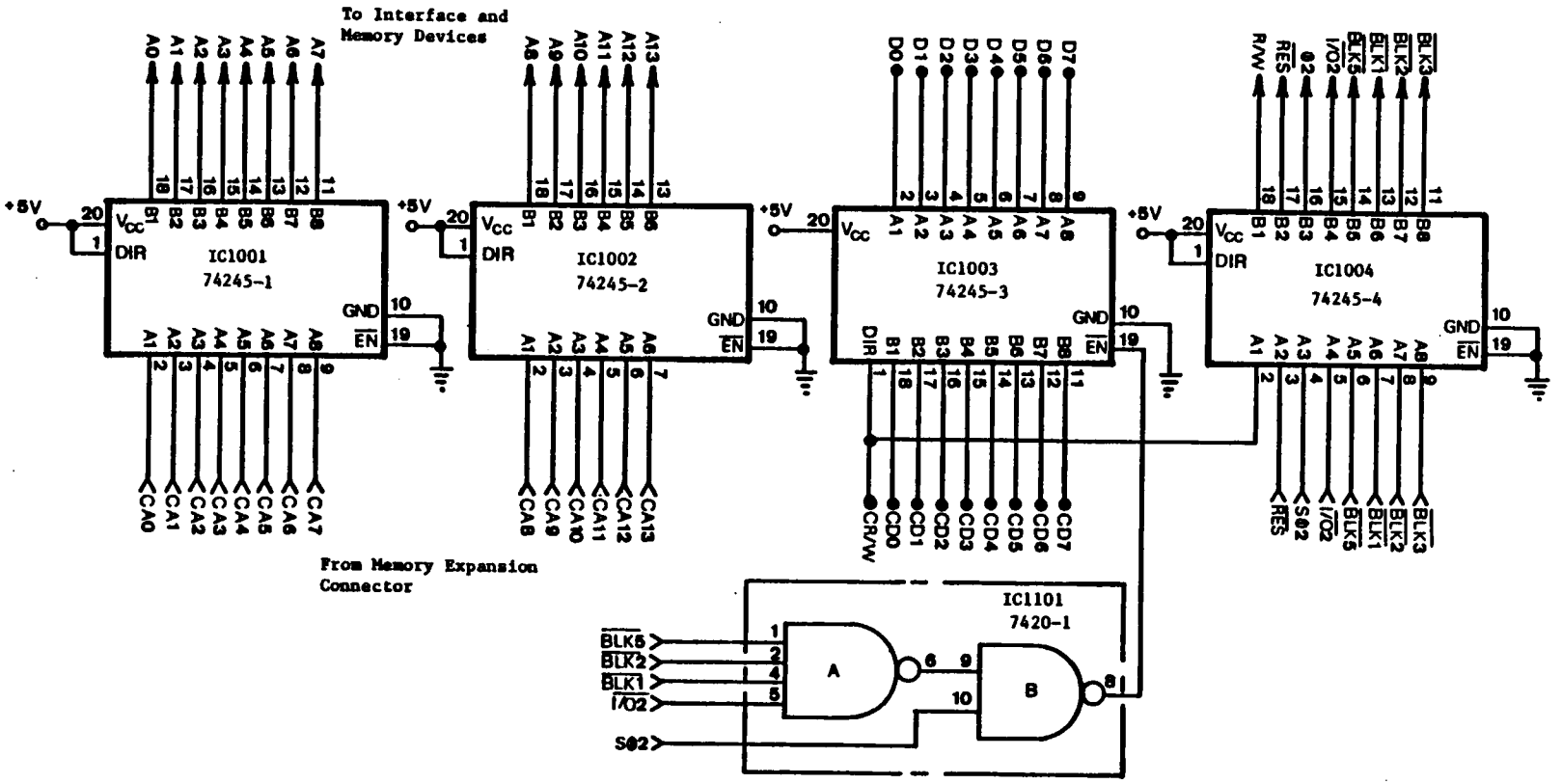


Figure 6. Memory Expansion Connector Buffer Circuitry

devices is enabled only when an expanded memory or interface device is selected. The 7420 dual quad-input NAND gate provides this data bus isolation logic.

Figure 7 illustrates the chip select logic used to access the interface devices. The 74154 is a 4 line-to-16 line decoder with 2 enable inputs. As shown, the circuit decodes address lines A5-A15 into 16 separate active-low pulses. Note that $\overline{I/O2}$ already has been decoded by the VIC 20 using address lines A10-A15. The five low order address bits are left undecoded since the interface devices (6522's) use these for chip enable and register access purposes.

EPROM Circuitry

The chip select logic for accessing program memory in EPROM is available directly from the VIC 20 expansion connector. As shown in Fig. 8, the $\overline{BLK1}$, $\overline{BLK2}$, and $\overline{BLK5}$ signals enable a separate 2716 EPROM device. The program for the process controller is stored in these three chips. Each one contains 2048 (2K) bytes of software in a 2048-bit by 8-bit configuration. The chips are electrically programmable and ultraviolet light erasable. Since each chip contains only 2K of memory and the $\overline{BLK}$ enable pulse decodes an 8K block of memory, there will be 6K of inaccessable memory for each $\overline{BLK}$ signal. The programmer must consider this when writing a program over 2048 bytes long. Table 2 illustrates this by listing the hexadecimal address range of each EPROM chip and its associated $\overline{BLK}$ enabling range.

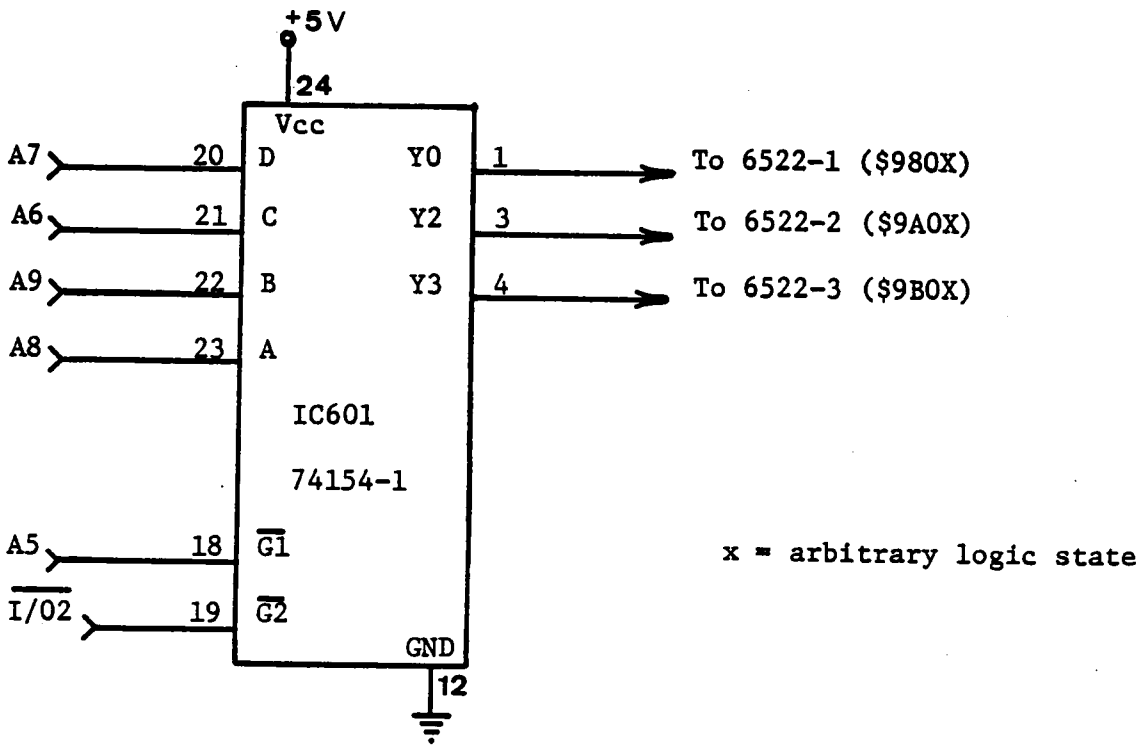


Figure 7. Chip Select Logic for Interface Devices

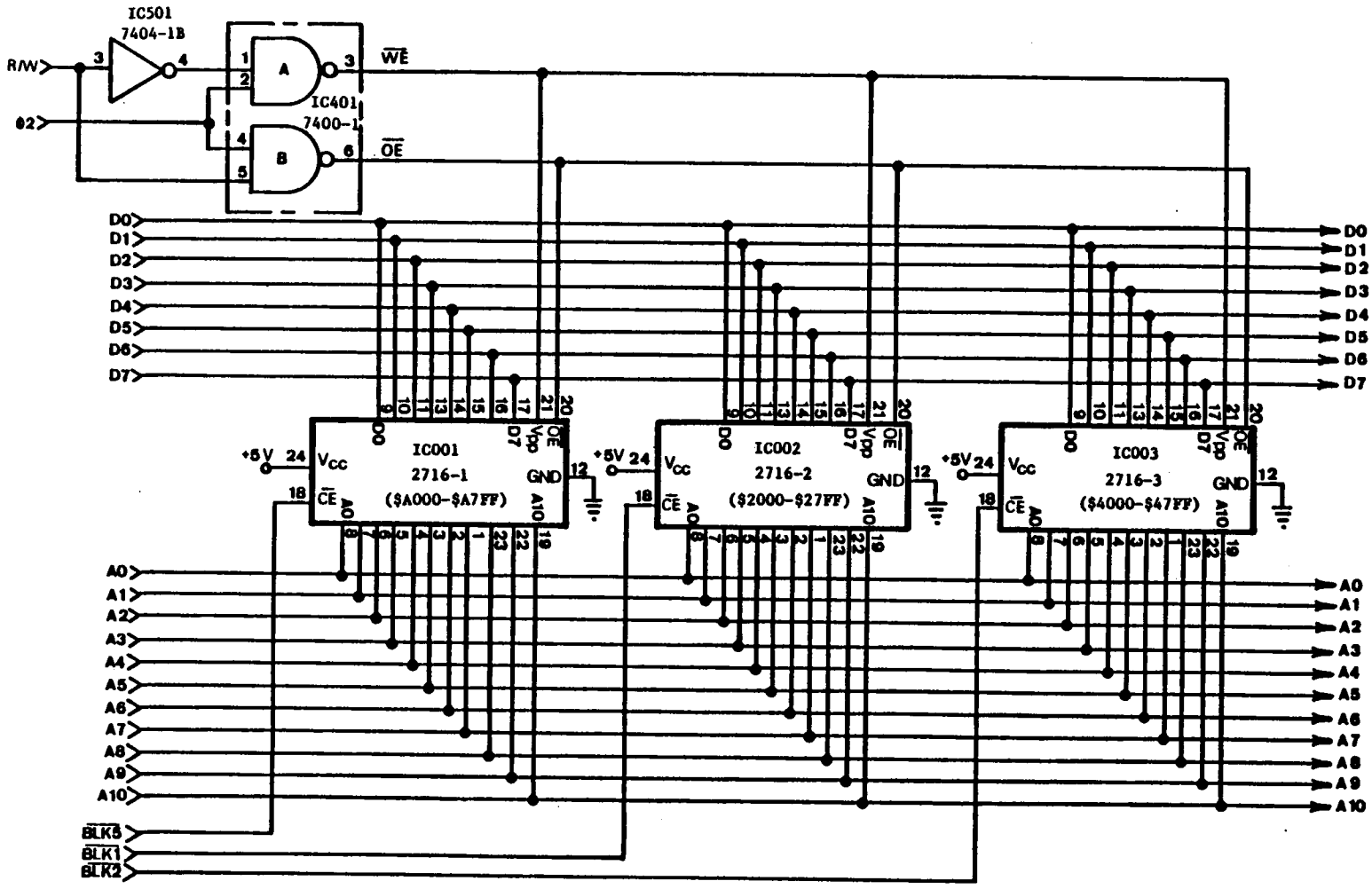


Figure 8. EPROM Interface Circuitry

Table 2. EPROM Address Range versus $\overline{\text{BLK}}$ Enabling Range

EPROM	EPROM Range	$\overline{\text{BLK}}$	$\overline{\text{BLK}}$ Range
2716 (IC 002)	$2000-$27FF	$\overline{\text{BLK1}}$	$2000-$3FFF
2716 (IC 003)	$4000-$47FF	$\overline{\text{BLK2}}$	$4000-$5FFF
2716 (IC 001)	$A000-$A7FF	$\overline{\text{BLK5}}$	$A000-$BFFF

Figure 8 also shows how the EPROM devices are enabled synchronously with the system clock. The two NAND gates in IC 401 produce a write enable ($\overline{WE}$) and output enable ($\overline{OE}$) pulse which change with the system clock. This ensures the chips are placing their instructions on the data bus at the proper time. All memory and interface devices which do not have a clock input for synchronization should be enabled with a pulse which has been synchronized with the system clock.

Interface Devices

Three 6522 Versatile Interface Adapters (VIA's) are used for temperature and motor controller I/O. These interface chips each contain two bi-directional peripheral data ports, a pair of very powerful interval timers, and a seriel-to-parallel/parallel-to-seriel shift register. Figures 9 and 10 depict the wiring of the chips. Table 3 summarizes pin assignments for all three VIA's.

The VIA (IC 101) in Fig. 9 performs several functions. Port A acts as an input port for converted temperature probe data. Port B is an output port whose individual lines control temperature probe and motor controller switches. Bits 0 and 1 of Port B determine which temperature probe is connected to the MC 14433 analog-to-digital (A/D) converter. Bits 2 and 3 turn motors 1 and 2 on and off. Input line CB1 is used to detect an end of conversion pulse from the A/D converter.

The other two VIA's (IC's 102 and 103) in Fig. 10 perform identical functions. On both, port A is an output port which delivers an 8-bit set-point to a motor controller. Port B is used to access the two

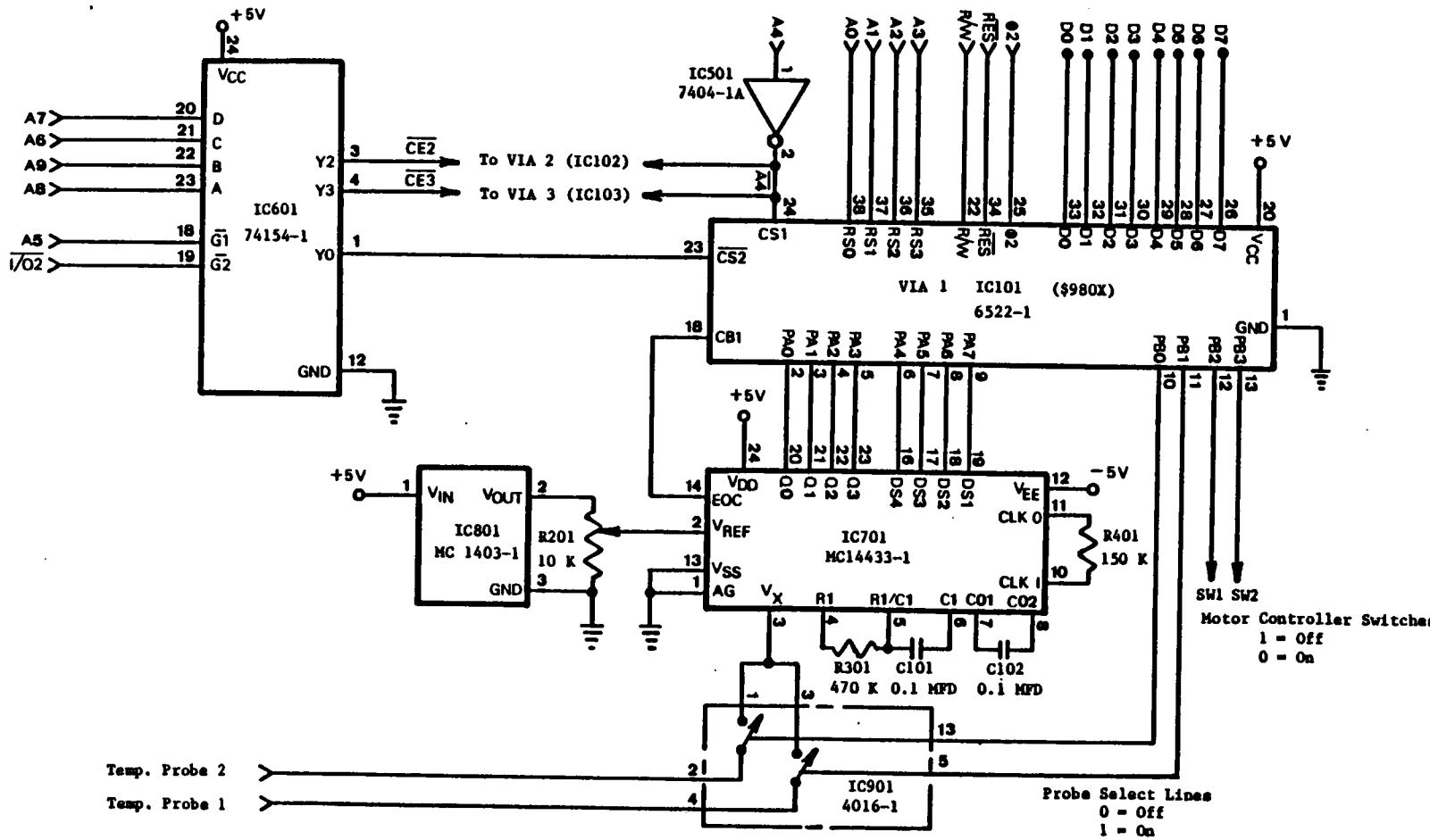


Figure 9. Interface Circuitry on VIA 1 (IC101)

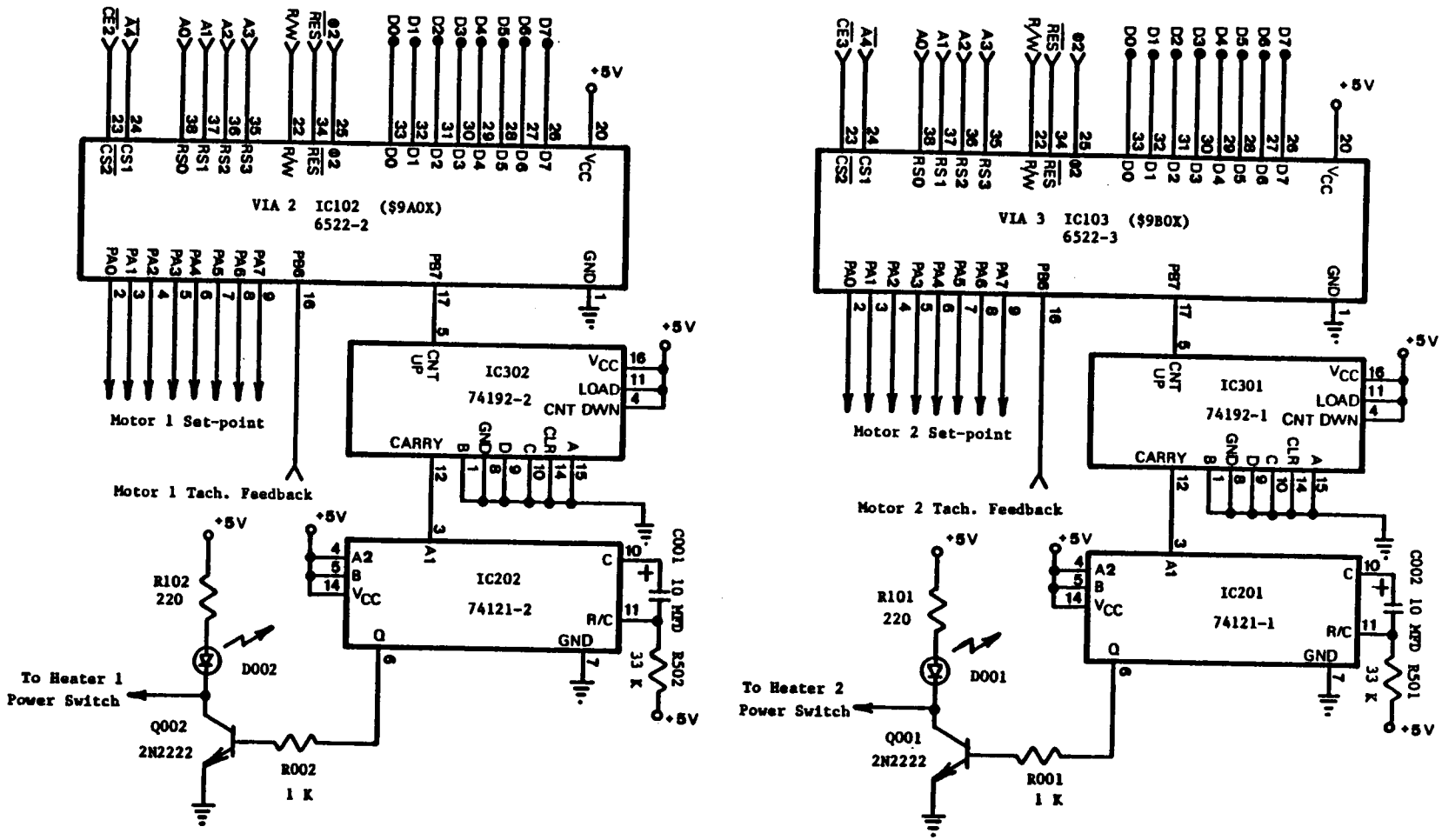


Figure 10. Interface Circuitry on VIA 2 (IC102) and VIA 3 (IC103)

Table 3. VIA Line Assignments

	Pin(s)	Functions	Description
	PA0-PA3	Input	A/D Converter Q0-Q3
VIA 1	PA4-PA7	Input	A/D Converter DS4-DS1
IC101	PB0-PB1	Output	Temperature Device Select
	PB2-PB2	Output	Motor Controller Switches
	PA0-PA7	Output	Motor 1 set point
VIA 2	PB6	Input	Motor 1 Pulse Counter
IC102	PB7	Output	Heater 1 Power Control Square Wave
	PA0-PA7	Output	Motor 2 set point
VIA 3	PB6	Input	Motor 2 Pulse Counter
IC103	PB7	Output	Heater 2 Power Control Square Wave

interval timers on each chip. Pin 6 of port B is used to count pulses from one of the motor tachometers. This count is used to calculate process material feed rate or stretch-ratio, depending on the VIA.

Motor Controllers

The proprietary motor control circuits were designed to regulate motor speed. The first objective of the design was to implement a programmable, dedicated circuit which controlled motor shaft velocity when given an 8-bit motor speed set-point. This frees the microprocessor from having to calculate new control signals for the motor via a digital algorithm. Another objective was that the controller respond quickly (in approximately 10 milliseconds) to changing shaft speed conditions, thus maintaining a constant velocity under any load condition. This rapid response requirement prevents the designer from using a digital control algorithm with the 6502.

A dedicated velocity feedback control system was selected to satisfy these objectives. Figure 11 illustrates the control loop. A digital encoder disk which provides 128 pulses per revolution is used to detect shaft velocity. This velocity information is then used by the controller to send a control signal to the motor to maintain constant shaft velocity. The controller, power supply, and feedback gain circuitry are proprietary information.

As shown in Fig. 10 and Table 3, two VIA's are used to send velocity set-point information to the motor controllers. Port A ($9A01) of VIA 2 and Port A ($9B01) of VIA 1 supply the controllers with the 8-

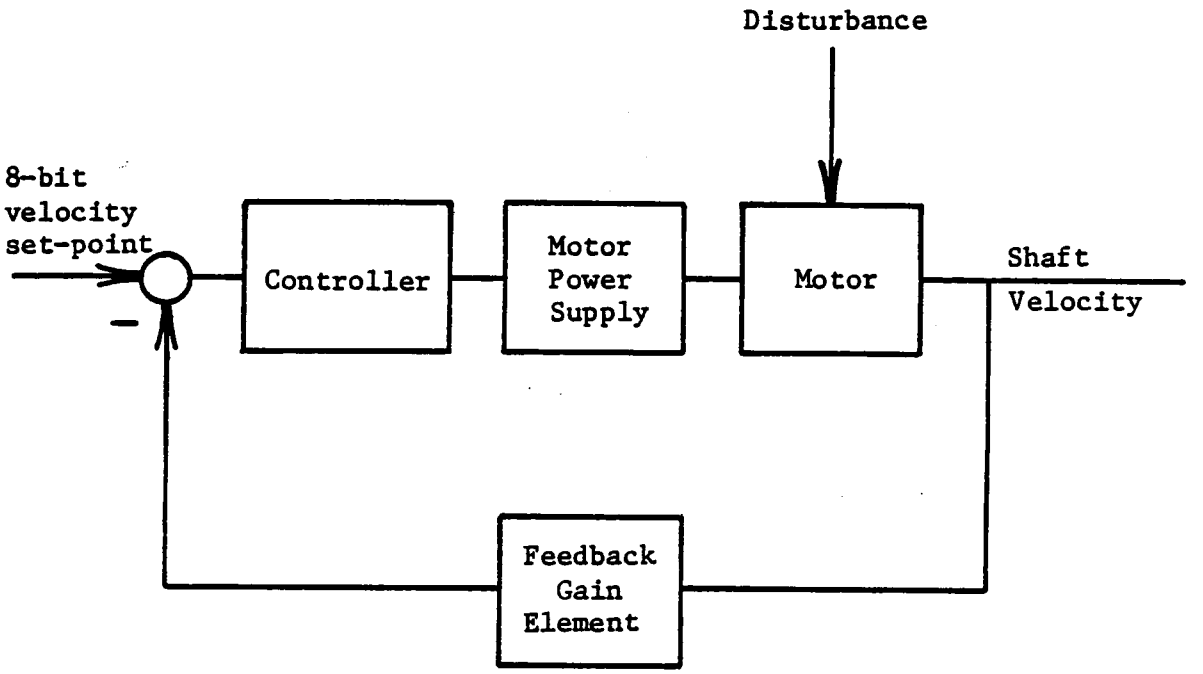


Figure 11. Motor Speed Control System

bit speed data. Bit 6 of port B on each VIA transmits shaft velocity information to the computer to calculate material feed rate and stretch-ratio for screen display.

Temperature Detection

Detection and processing of temperature information in this project is based on a design published in Mears' thesis "A Microcomputer-Based Temperature and Humidity Control System for Respiratory Therapy" [4].

Temperature signals are supplied to the microcomputer interface circuit board in the form of analog voltages in the range 0.000 to 1.999 volts. Two signals are supplied such that each 0.010 volt is equivalent to 1.0°C. Figure 9 illustrates how these signals are multiplexed to the analog-to-digital (A/D) converter, and in turn sent to the VIA.

A 4016 solid state switch (IC 901) is used to switch the two temperature signals by software control. The computer selects which signal is to be sent to the A/D converter by setting PB0 or PB1 on VIA 1 to a logical 1.

Analog-to-digital conversion is performed by a Motorola MC 14433 3 1/2-digit A/D converter (IC 701). This particular chip is a high-performance, low-power, low-cost converter designed to minimize use of external components. It performs dual slope A/D conversion with input voltage ranges of 199.9 millivolts or 1.999 volts at a maximum rate of 25 conversions per second. The accuracy of the MC 14433 is ±0.05 percent of the reading, ±1 count. As shown in Fig. 9, the A/D converter makes approximately eight conversions per second in the 0.000 to 1.999

volt range. The MC 14433 contains an oscillator system clock, and resistor R401 sets the frequency. The number of conversions per second equals the clock frequency divided by 16,400 [8].

The voltage reference for the A/D converter is provided by an Analog Devices AD580K low drift 2.50 volt voltage reference (IC 801) and a potentiometer (R201). The AD580K has a maximum output voltage change of 7 millivolts over the temperature range 32°F to 150°F (0° to 70°C). The potentiometer is used to provide a 2.00 volt reference for the A/D converter [9].

The A/D converter sends information to the computer on pins PA0-PA7 and pin CB1 of the 6522 VIA 1 (IC 101). Pin CB1 is an interrupt line of the 6522 and is used by the A/D converter to signal end of conversion (EOC) at the end of each conversion cycle. The A/D converter produces its converted data in binary-coded decimal (BCD) form. Four digits of BCD data contain the digitally-encoded equivalent of the analog voltage input. These 4-bit BCD digits are presented to the microcomputer one at a time on VIA 1 pins PA0 through PA3. The most significant digit is presented immediately following an end-of-conversion pulse and is followed by the remaining three digits in order of decreasing significance. The most significant digit is the "1/2" digit and is also coded with overrange, underrange, and polarity information. Pins PA4 through PA7 of VIA 1 are used by the A/D converter to indicate which digit is being presented. Each of these A/D converter digit select outputs is a logical 1 when its corresponding BCD digit is selected. Note that the four BCD data lines and the four digit select lines share

the same input port on VIA 1. This permits them to be read by the microprocessor as a single byte, thus avoiding any confusion which could arise if changes occur between the time the BCD data lines are read and the time the digit select lines are read [4].

Temperature Control

The temperature detection circuitry allows for control of up to two separate material processing area temperatures. Control is achieved through a software/hardware approach.

Figure 10 shows the hardware used to implement the control signals. Each VIA (IC 102 and IC 103) is programmed to produce a "free-running" square wave on pin PB7. The period and thus the frequency of this wave can be altered at any time by writing to the 6522 Timer 1 counters. Since the maximum period of the square waves is only 0.131 seconds, a longer period wave is produced using a decade counter (IC's 301 and 302). The decade counters feed one-shots (IC's 201 and 202) which provide pulses of fixed duration to switch on transistors Q001 and Q002. The transistors, when switched on, provide a signal of known duty cycle to the heating elements. Thus, the rate at which the one-shot is fired controls the average power to the heater.

Programming the 6522 to produce square waves on pin PB7 is relatively easy. Once the chip is initialized to produce the continuous square wave, the period of the wave can be programmed by writing data to the internal 16-bit counter. The maximum count is 65,536 and the period of the square wave is 2N + 3.5 cycles of the 02 clock signal. "N" is

the value written to the 16-bit counter. Thus, a 1.000 MHz clock limits the maximum period of the wave to 0.131 seconds. The decade counter divides by 10, producing a maximum period of 1.31 seconds. This yields evenly-spaced triggering pulses for the one-shot. The duration of the output pulse from the one-shot is fixed by the external timing components of the one-shot. Thus, the duty cycle of the heater element control signal can be altered through software as well as hardware (changing the timing components of the one-shot) [4]. The exact duty cycle can be calculated from

$$\text{Duty Cycle (\% on)} = \frac{\text{Duration of One-Shot Output Pulse}}{(2N + 3.5)(1 \times 10^{-7})} \qquad (1)$$

Temperature Controller Analysis

The described proportional temperature controller can be analyzed using z-transform techniques to determine the range of stable behavior. The forward loop of the sampled data control system consists of a zero-order hold device (6522), the controller (heater power element), and the first-order plant (the control volume). The feedback loop has unity gain because of the characteristics of the temperature probe and A/D converter.

The transfer function of the entire closed loop system is given by

$$TF = \frac{K_c K_p (1 - e^{-T/\tau})}{z - e^{-T/\tau} + K_c K_p (1 - e^{-T/\tau})} \qquad (2)$$

where

K_c = Controller gain, watts,

K_p = Plant gain, °C/watts

T = Sample interval, 0.25 seconds,

τ = Plant time constant, seconds.

The controller gain, K_c, is determined from the equation

$$K_c = (\text{Duty Cycle})(K) \tag{3}$$

where

Duty Cycle = Percent on-time as calculated from equation 1.

K = Heater power setting, watts.

Since all poles of the closed loop transfer function must be within the unit circle for the system to remain stable, the following relationship determines the stability range of the temperature controller:

$$T < \tau \ln \left[\frac{K_c K_p + 1}{K_c K_p - 1}\right] . \tag{4}$$

This inequality shows that the controller will remain stable for the described application because the control volumes to be used will have very long time constant (10 minutes, minimum) compared to the sample interval and the product $K_c K_p$ will not be high enough to drive the system unstable.

Note that higher gain, K_c, will increase steady state oscillation but will reduce steady state error and lower gain will have the opposite effect. The operator will therefore be required to "fine-tune" the system response by selecting a desired constant-level heater power setting, K.

The temperature controller may also be modeled and analyzed as a linear system for quick but less accurate response simulations. This may be done since the sample interval will be much smaller than the open loop system time constant.

Power Supply

A commercially available power supply external to the VIC 20 is used to drive all interface and control hardware connected to the VIC 20 through the memory expansion connector. The GND lines from the VIC 20 are used to supply a common ground for all electrical connections.

IV. SOFTWARE DESIGN

Overview of Process Control Sequence

The processing sequence begins with preparing the system for operation. First, the material to be processed must be hand-fed through the system. A very low motor speed may be used to aid the user in feeding the material through the pulley system connected to each motor shaft. Next, the operator should be able to enter the processing control parameters of feed rate, stretch-ratio, and temperature. After this, he should be able to independently turn the heating elements on and off. The system is now prepared for operation.

Once prepared, the system begins controlling the temperature. If system temperature ever rises more than 10°C above desired set-point, then an alarm sounds and the system shuts down. When the temperature reaches the desired set-point value, the motors can be turned on by the operator to run at their selected rates. At this point material processing starts. Throughout the controlling section of the processing sequence the operator must be able to terminate the process, stop motors, or change parameters at any time.

Program Details

An assembly language program controls the entire process. Each subroutine was written and tested on either the VIC 20 using the VIC MON® assembler cartridge or the AIM 65 Advanced Interactive Microcomputer. The final version of the whole program was assembled on

the AIM 65 and burned into EPROM chips with the DRAM PLUS® EPROM programmer. Table 4 presents the address and contents of each EPROM chip.

The program is an interactive routine which allows the user to select all the control parameters used in the process. It "steps" the user through the preparatory and control sections by displaying instructions, codes, and error signals. The assembly language program listing is included in Appendix A

Initialization

When the VIC 20 is turned on, the initialization sequence in Appendix C is executed. During this initialization the VIC 20 searches for the auto-start sequence, as discussed in the VIC 20 review section. After finding this sequence at $A004, program execution begins at $A009, which is the beginning of the process control initialization routine.

The process initialization routine is flowcharted in Fig. 12. The first part of this routine turns off the motors and disconnects temperature probe inputs to the A/D converter. To do this, the VIA interface devices must be configured.

The first step in configuring the interface devices is to specify whether the peripheral port pins are to act as inputs or outputs. This is done by writing the appropriate binary values into the Data Direction Registers (DDR's). Once the DDR's are initialized, the chip's output pins can be brought to initial operating state by writing to the output

Table 4. Location and Contents of EPROM Chips

Location	Contents
$A000-$A7FF	Initialization Routine, Main Program, Subroutines
$2000-$27FF	Subroutines
$4000-$47FF	Interrupt Routine, Messages, Look-up Table

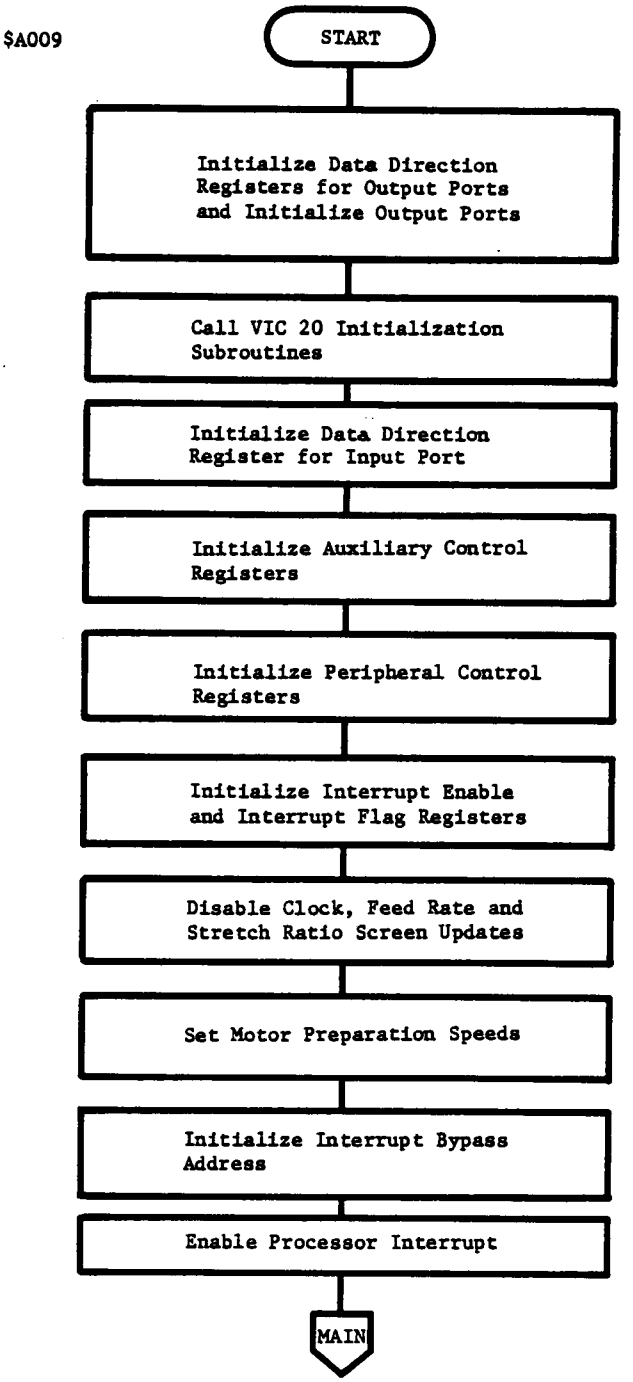


Figure 12. Initialization Routine Flowchart

ports. All motor and temperature probe switches are initialized in this way.

After motor and temperature probe initialization, four VIC 20 Kernal subroutines are called. These subroutines clear RAM locations $0000-$03FF, set all jump vectors, initialize I/O chips, and initialize the screen.

Next, the remaining steps in configuring the interface devices are performed. The DDR of the input port is set and the rest of the initial operating states of the output ports are specified.

The Auxiliary Control Registers (ACR's) in each interface chip are then initialized. The ACR selects the operating mode for the two interval timers (T1,T2) in the device. Another control register initialized at this time is the Peripheral Control Register (PCR). This register is used primarily to select the operating mode of the four peripheral control pins which affect the interrupt flags in the chip.

Another step in configuring the VIAs is to enable the interrupts which will be used by the programmer. The Interrupt Enable Registers determine which interrupt flags the microprocessor responds to. The final configuration step is to initialize the Interrupt Flag Registers (IFRs), which indicate which interrupt lines have been active.

Once all the interface devices are initialized, the initialization routine disables the clock, feed rate and stretch-ratio screen updates by setting flags in memory. Then the routine selects the motor speeds for process preparation and initializes the interrupt bypass address. The microprocessor interrupt flag is then enabled so interrupt requests can be processed.

Main Program

After initialization, the main program loop shown in Fig. 13 is executed. This loop controls the entire process. Subroutines are used extensively.

The main sequence begins by displaying the title screen (SCRN1) on the system CRT monitor. Appendix B contains the ten display screens used in the main loop. SCRN1 remains on the CRT for approximately 12 seconds before the program continues.

After the title screen is displayed, screen character colors for the rest of the main program are specified with the COLOR subroutine. SCRN2 then follows, which prompts the user to enter the time on the keyboard for the programmed 24-hour clock. The FETCH1 subroutine accepts and echos the operator's input on the screen. CHECK1 tests this input for valid data and sets an error flag in memory if it is invalid. If invalid, an error message is displayed with the ERROR subroutine and program flow jumps back to the beginning of the FETCH1 subroutine to get a new time. If the entered data is valid the program continues by displaying the next screen.

The next screen is SCRN2A which prompts the operator to enter the diameter of the drive wheel connected to motor 2. This value will determine the output feed rate of the processed material. FETCH2 accepts, echos on the screen, and checks the entered data for validity. As before, if the data is invalid an error flag is set in memory, ERROR is called, and the program jumps back to accept new

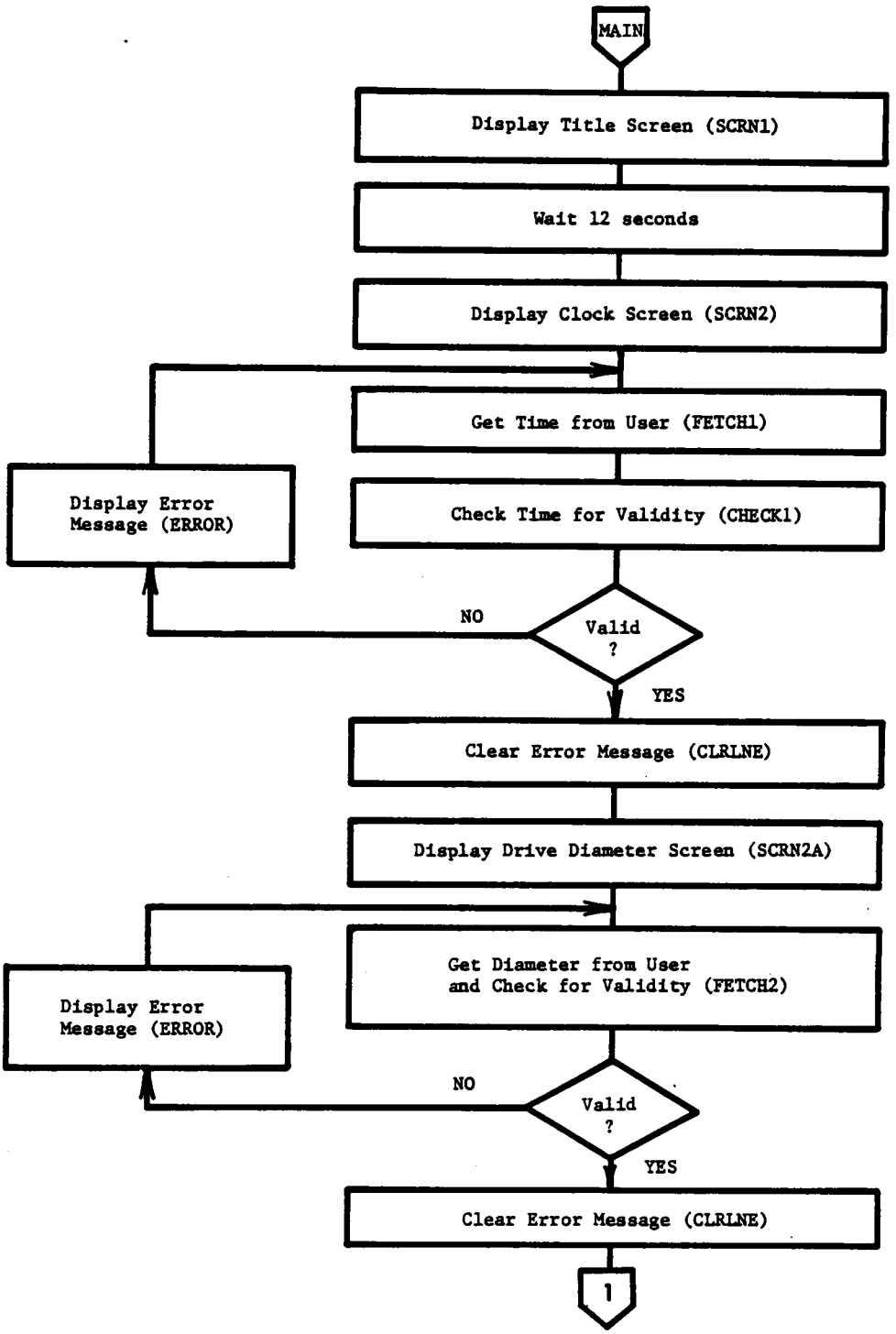


Figure 13 a. Main Program Sequence

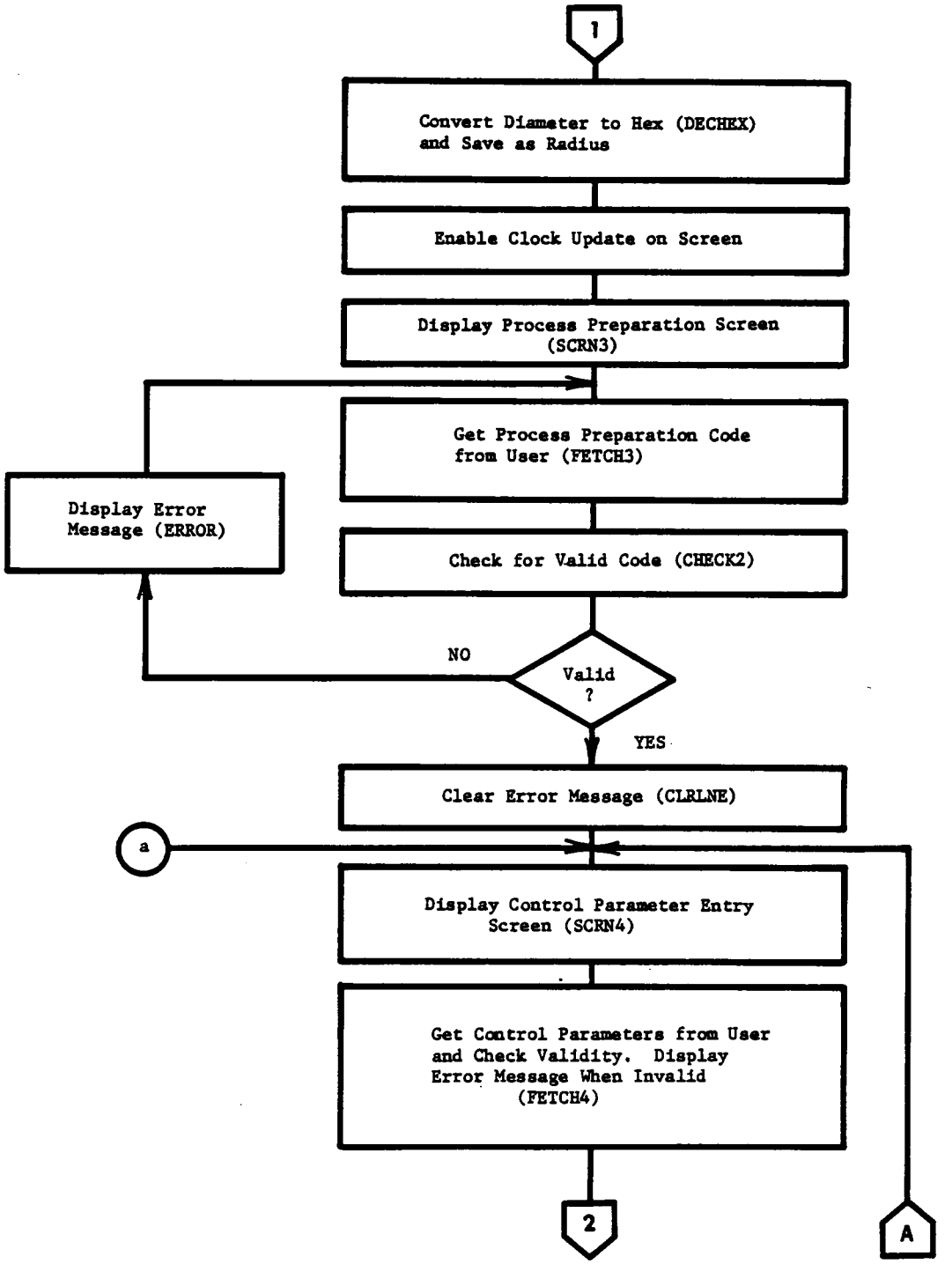


Figure 13 b. Main Program Sequence

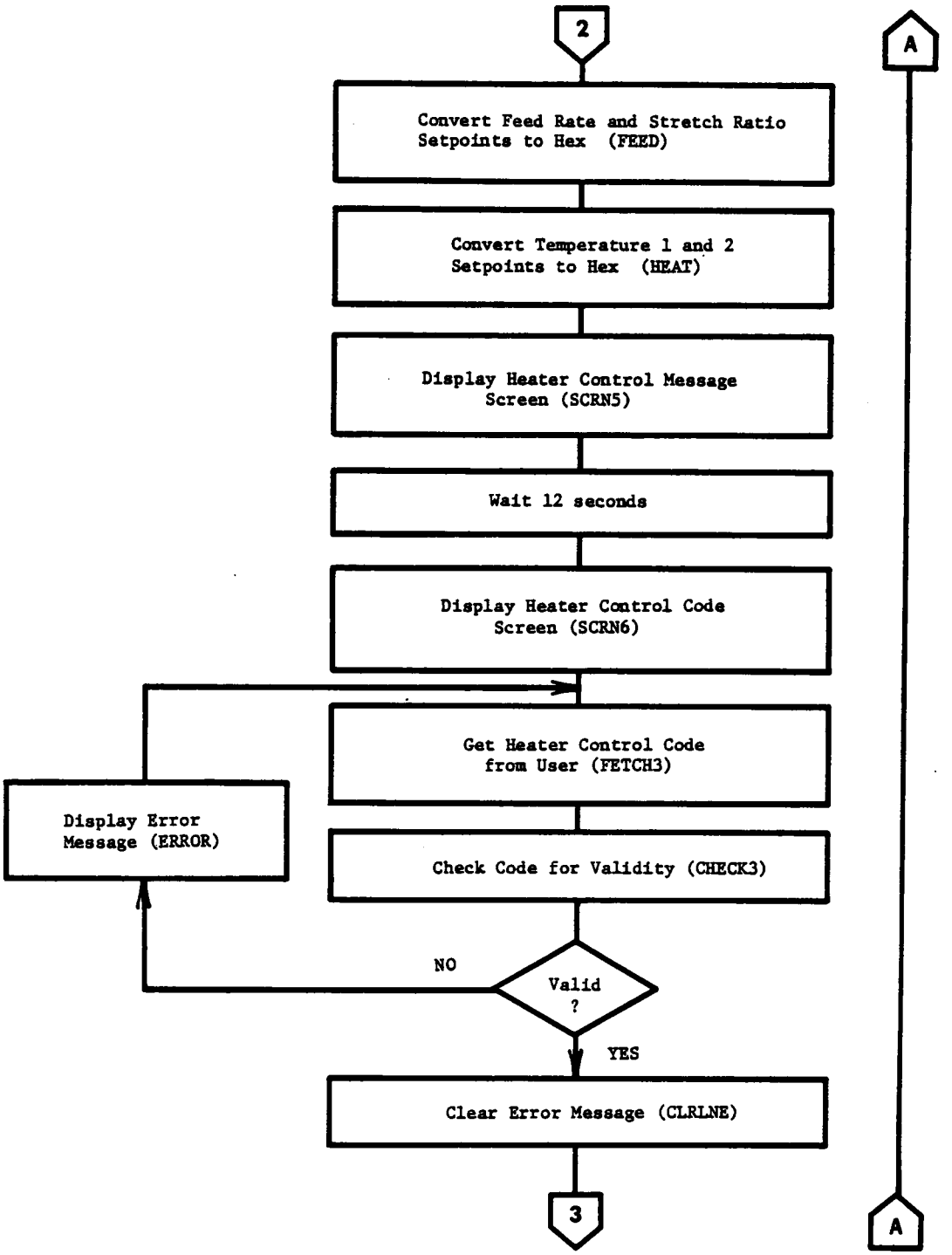


Figure 13 c. Main Program Sequence

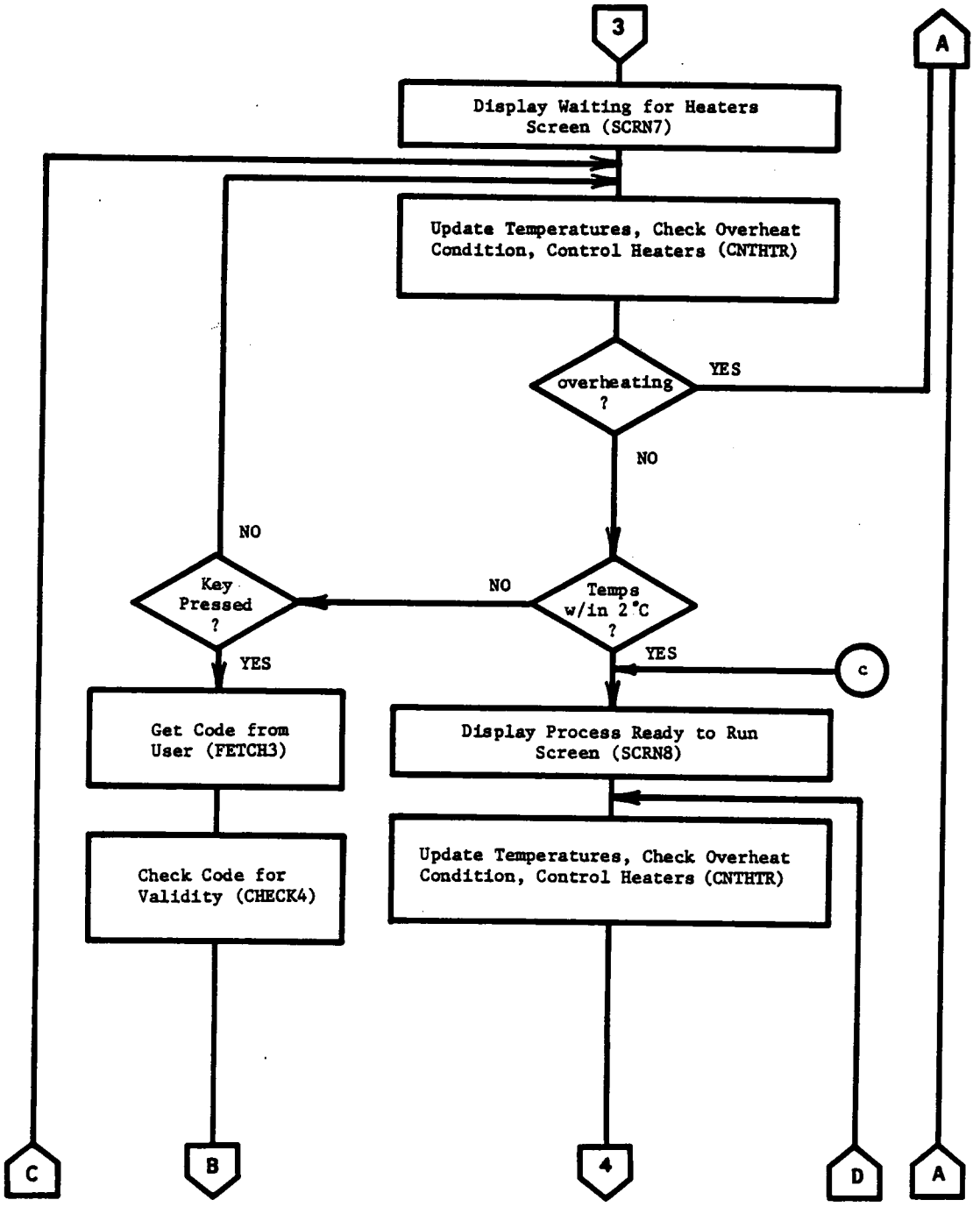


Figure 13 d. Main Program Sequence

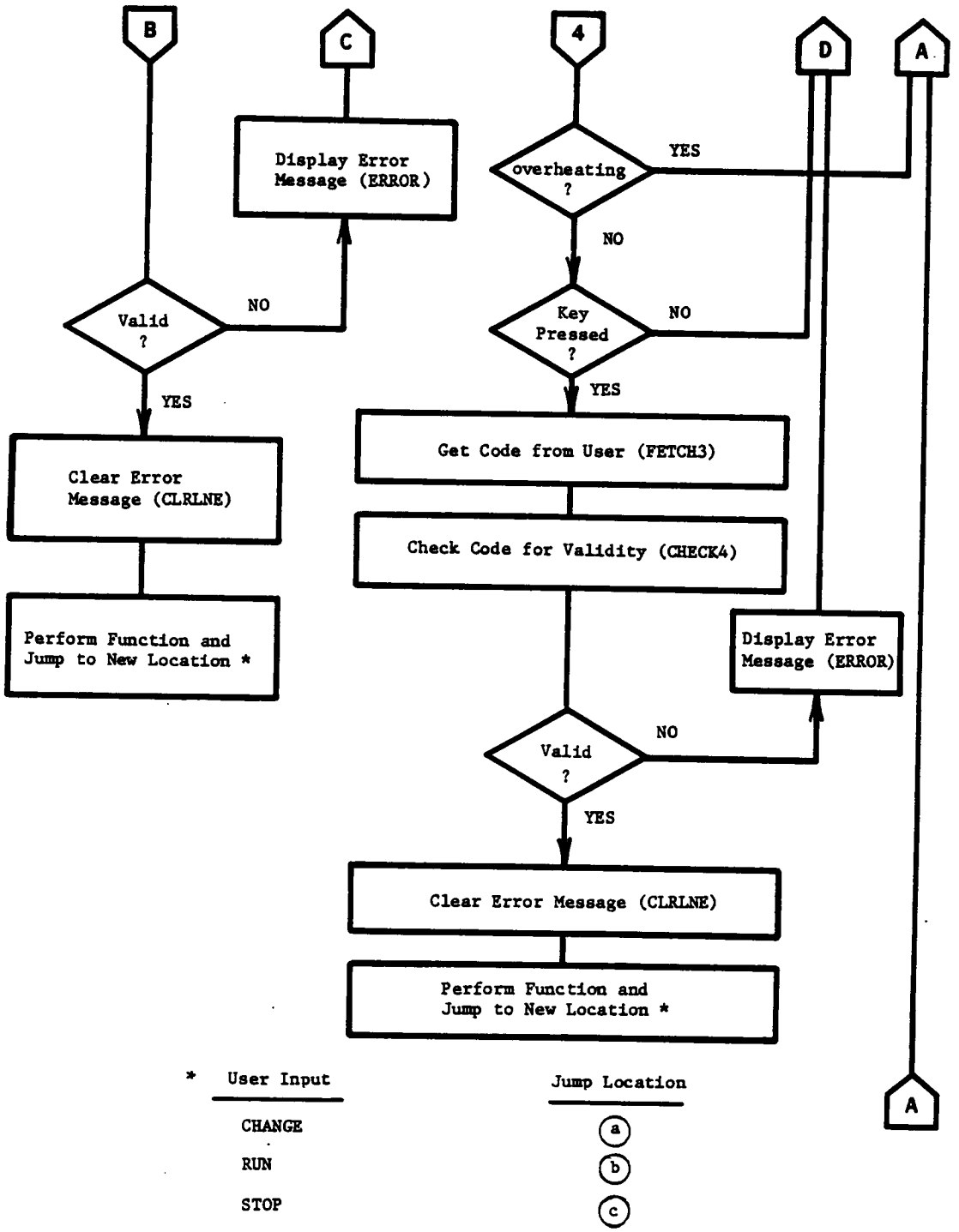
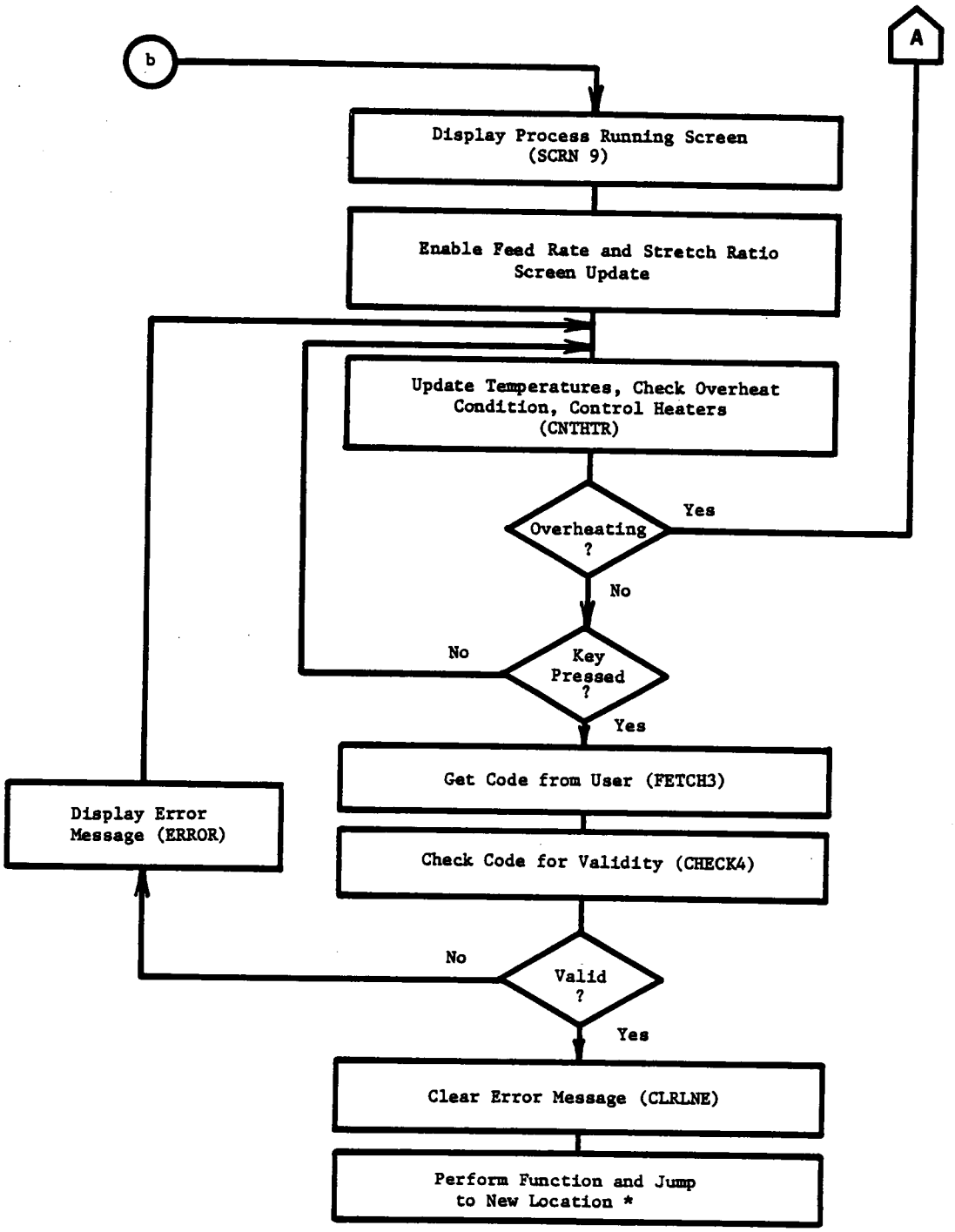


Figure 13 e. Main Program Sequence



*See Fig. 13e.

Figure 13 f. Main Program Sequence

data. If the data is valid, the program converts it to a hexadecimal number and saves this value in memory for future calculations.

Next, the clock update flag in memory is cleared. This allows the time on the screen to be updated by a routine in the interrupt bypass program.

The process preparation screen, SCRN3, is then displayed. SCRN3 instructs the operator how to independently turn either process motor on or off for the process preparation sequence. The operator enters the desired code on the keyboard followed by a carriage return. FETCH3 accepts the keyboard entry and CHECK2 determines whether a valid code was entered. If the entry is invalid, an error message is displayed with the subroutine ERROR. The only way to exit the process preparation sequence is to enter the code CONT for "continue with process."

After the system has been prepared for operation, SCRN4 displays the control parameter input screen. This screen allows the user to enter process control parameters of feed rate, stretch-ratio and temperature. The subroutine FETCH4 accepts the control parameters from the keyboard and checks them for validity. If an invalid parameter is entered, ERROR is called and program flow jumps back to get a new value. When all parameters are entered, the subroutine FEED is called to convert the feed rate and stretch-ratio set-points to 8-bit hexadecimal numbers. The main program then calls the subroutine HEAT to convert the temperature set-points to hexadecimal values.

As soon as the control parameter conversion is complete, SCRN5 is called. The subroutine SCRN5 displays an instructional message on the

screen for the user to enter a heater operation code. This meassage remains on the screen for approximately 12 seconds. After the 12 second delay, SCRN6 is called which displays the actual heater operation codes. These codes inform the operator how to turn on and off each heater. Subroutine FETCH3 accepts the keyboard entries and CHECK3 determines validity. Again, if the data is invalid, ERROR is called to display an error message and the program jumps back to accept a new entry. The only way to exit this loop is to enter CONT as explained for the process preparation sequence.

After exiting the heater initialization loop, SCRN7 subroutine is called. This subroutine displays a message informing the operator that the system is waiting for the temperature to reach within 2°C of set-point value. In addition, SCRN7 displays the codes that allow the operator to change parameters, terminate the process or perform an emergency stop of heater control signals.

The main program enters a temperature control loop immediately following the SCRN7 subroutine call. This control loop first calls the subroutine CNTHTR which samples each temperature probe output, checks for an overheat condition, and if overheating didn't occur, implements the heater control signals by writing a 16-bit counter value to Timer 1 of VIA 2 and 3. If an overheat condition occurs, an alarm sounds, heaters are turned off, the screen turns red, and program flow jumps to the control parameter input screen (SCRN4). If overheating did not occur, the control loop continues by implementing the heater control signals. Next, the sampled temperatures are tested to determine if they

are within 2 degrees Celsius of set-point value. If both temperatures are within this range then program flow jumps out of the loop to display SCRN8. If the temperatures do not fall in this range, the loop continues by checking for keyboard input. If a key was pressed, FETCH3 is called to accept the operator's code and CHECK4 determines if the entry is valid. For a valid entry, the program performs the desired operation. For example, if the operator wants to change parameters, the program jumps back to the control parameter input screen (SCRN4). If the operator wants to terminate processing then both heaters are turned off, the screen is blanked, and the microprocessor enters an endless loop. For an emergency stop input, both heaters are turned off and program flow jumps to the heater initialization screen (SCRN5). If the entry was invalid, an error message is displayed and the control loop starts over by calling CNTHTR. If a key was never pressed after the temperature range test, then the temperature control loop begins again by calling the subroutine CNTHTR. Thus, the only way to continue with the process is for both temperature probes to read within 2°C of the desired values.

When both probes read within 2°C of their set-points, the SCRN8 subroutine is called. SCRN8 displays a message informing the operator that the process is ready to run. Also, SCRN8 displays the codes that allow the operator to run the process motors, change control parameters, terminate process, or perform an emergency stop of motors and heaters.

After SCRN8 is displayed, the program enters a temperature control loop similar to the one which follows SCRN7. First, CNTHTR is called.

If CNTHTR senses an overheat condition, the ALARM subroutine is called which sounds an alarm, turns off motors and heaters and makes the screen red. Upon returning to the calling routine, program execution jumps to the control parameter input screen (SCRN4). If overheating did not occur, heater control signals are implemented by writing counter values to the Timer 1 registers of VIA 2 and 3 and the control loop continues. Next, the keyboard is checked to determine if a key has been pressed. As before, if a key was pressed, FETCH3 is called to accept the operator's input and CHECK4 determines if the entry is valid. If the entry is valid, the desired function is executed. Depending on the input, the motors are turned on, program flow jumps back to control parameter input screen (SCRN4), the process is terminated, or an emergency stop of motors and heaters occurs. If a key was never pressed, then the temperature control loop begins again by calling CNTHTR. The only way to continue with the process is for the operator to choose to run the process motors.

When the operator chooses to run the process motors, the subroutine SCRN9 is called. SCRN9 displays a message on the screen informing the operator that the process is running. Immediately following SCRN9, a flag in memory is set to allow the measured values of feed rate and stretch-ratio to be updated on the screen.

Next, the program enters a temperature control loop nearly identical to the one following SCRN8. The only difference between the two is that SCRN9 displays a "Process Running" message and allows the operator to stop the process motors instead of starting them. If the

operator stops the motors, the program jumps back to SCRN8 to allow him to start the motors again. The process continues running until the operator terminates it or an alarm occurs.

Subroutines

The subroutines in the process control program will be presented as four separate groups. They are keyboard, screen, calculation, and control. Some subroutines call other subroutines. Table 5 lists both the calling and called subroutines used in the main program sequence.

Keyboard

The subroutines classified as "keyboard" are those which accept and check user input from the keyboard. The keyboard subroutines are FETCH1, FETCH2, FETCH3, FETCH4, CHECK1, CHECK2, CHECK3, CHECK4, CHKSP, and TEMPS.

The subroutine FETCH1 accepts, saves in memory, and displays on the screen the operator's input for the time. FETCH1 will start over if six characters are entered and not followed by a carriage return. A carriage return signifies the end of operator input. Program execution then returns to the calling routine.

The subroutine FETCH2 is similar to FETCH1. FETCH2 accepts, saves, and displays keyboard entry for the drive wheel diameter of motor 2. FETCH 2 will start again if two characters are entered and not followed by a carriage return. A carriage return signifies the end of input. Program flow then returns to the calling routine.

Table 5. Subroutines that Call Other Subroutines

Calling Subroutine	Called Subroutines
SCRN1	DISPLY
SCRN2	DISPLY
ERROR	DISPLY
CLRLNE	DISPLY
SCRN2A	DISPLY
SCRN3	DISPLY
CHECK2	CHKSP, R1,R2,RB, S1,S2, CONT
SCRN4	DISPLY
FETCH4	ERROR, CLRLNE, TEMPS
TEMPS	CLRLNE
FEED	DECHEX, M8BY8,D16BY8
HEAT	DECHEX
SCRN5	DISPLY
SCRN6	DISPLY
CHKSP	SPBAR
SCRN7	DISPLY
CNTHTR	TNEW,DECHEX,ALARM
SCRN8	DISPLY
CHECK4	CHKSP, RUN, STOP, QUIT
SCRN9	DISPLY

FETCH3 is a generalized form of FETCH1 and FETCH2. It is called in the main program to accept, save in memory, and display user input on the screen. Like FETCH1, FETCH3 will start over if six characters are entered and not followed by a carriage return. A carriage return signifies end of input. FETCH3 is called during screen display of SCRN3, SCRN6, SCRN7, SCRN8 and SCRN9.

The subroutine FETCH4 is the most complicated "FETCH" routine. It accepts, displays, and checks for valid keyboard input for all control parameter set-points, all of which are three characters long. The first parameter is feed rate. If the feed rate set-point is valid (000-199 in/min), then the next set-point, stretch-ratio, is accepted from the keyboard. Appendix F presents a copy of the operator's procedure to select a valid feed rate/stretch-ratio combination. Controllable motor speed range determines the combination validity. If feed rate is invalid, an error message is displayed and the operator must re-enter a new set-point. After a valid feed rate is entered, the error message line is cleared and the stretch-ratio parameter is accepted. If it is invalid, an error message is displayed, and the operator must re-enter a new set-point. If the stretch-ratio is valid (1.00-3.99), then the error message is cleared and the process temperature set-points are accepted and checked in succession. The temperature set-points must be in the range 000-159°C. If one does not, an error message is displayed and the operator must re-enter that temperature. When all parameters are successfully entered, program flow returns to the calling routine.

The "CHECK" subroutines are used to test for valid input accepted by the "FETCH" subroutines. The CHECK1 subroutine determines if the operator's entry for the time is valid. If not, a flag is set in memory to indicate such.

The CHECK2 subroutine tests for valid process preparation codes, which are R1, R2, RB, S1, S2, SB, and CONT. If a valid code is detected, the subroutine of the same name is called. These subroutines will be discussed in the section on control subroutines. If a valid code is not found, an error flag is set in memory.

CHECK3 is similar to CHECK2. CHECK3 tests for valid heater operation codes. These codes are H1, H2, HB, 01, 02 and CONT. If a valid code is found, a subroutine of the same name is called. These subroutines, too, will be covered in the section on control subroutines. If a valid code is not detected, an error flag is set.

The subroutine CHECK4 is used to test the input accepted by FETCH3. CHECK4 tests for valid process operation codes of RUN, STOP, QUIT, and CHANGE. If a valid code is detected, a control subroutine of the same name is called. If a valid code is not found, an error flag is set in memory.

Subroutines CHECK2, 3, and 4 call the subroutine CHKSP to check for a space bar character. A space bar indicates an emergency stop of all motors and heaters is desired. CHKSP tests the first three characters accepted by CHECK2, 3, or 4 for a space bar. If found, the subroutine SPBAR is called to stop the motors and heaters, and a flag is set to indicate this. If a space bar is not found, the flag is cleared.

The last keyboard subroutine is TEMPS. TEMPS is called by FETCH4 to test a temperature set-point entered by the operator. If the set-point is invalid, an error flag is set in memory. FETCH4 calls TEMPS twice, once for temperature 1 and once for temperature 2.

Screen

The screen subroutines are those which affect the video display screen. These subroutines include SCRN1, SCRN2, SCRN2A, SCRN3, SCRN4, SCRN5, SCRN6, SCRN7, SCRN8, SCRN9, ERROR, CLRLNE, DISPLY, and COLOR.

All of the "SCRN" subroutines display full CRT screens which lead the operator through the process sequence. SCRN1 first selects the VIC 20 character generator set 2, and then sets the screen and border colors to light cyan and blue, respectively. Next, the project title is displayed with the DISPLY subroutine. Messages 1 and 2 contain the characters which produce the title.

SCRN2 displays messages 1, 2, and 3. Message 1 and 2 restore the project title to the top of the screen. Message 3 instructs the operator to enter the time. As for all "SCRN" subroutines, the DISPLY subroutine is used to put each message on the screen in the desired location.

SCRN2A displays message 2A. This message instructs the operator to enter the diameter of the output drive wheel.

SCRN3 displays process preparation information. Messages 5, 6, and 7 contain the screen title, instructions, and keyboard codes, respectively.

Control parameter selection is made possible with SCRN4. This subroutine uses messages 8, 9 and 10 to display the parameters, instructions, and screen title, respectively.

SCRN5 displays a short instructional message to the operator to enter an upcoming heater operation code. Message 11 contains the characters which produce this instruction.

SCRN6 displays the heater opertion codes via message 12. These codes are used by the operator to turn the heaters on or off before processing begins.

Processing can begin when the heaters bring the system temperature to within 2°C of set-point value. SCRN7 displays message 14 until the system reaches this 2°C range. Message 14 contains the keyboard codes the operator may use while waiting for the heaters.

SCRN8 displays a message that the process is ready to run, i.e., the system temperature is up to the desired level. Operator keyboard codes are also displayed with this message (Message 15).

SCRN9 notifies the operator via message 16 that the process is running. Message 16 also displays keyboard codes for altering the present system status.

Appendix B illustrates the video screen displays after each "SCRN" subroutine call. Each message number is noted at the right of the "screen."

Two other subroutines that affect the video screen are ERROR and CLRLNE. ERROR simply displays message 4 which notifies the operator that an invalid keyboard entry was made. CLRLNE clears message 4 by printing spaces on line 15 of the screen.

The subroutine DISPLY is used to display all messages. The calling routine must specify the beginning address of the message in memory, the number of characters in the message, and the location on the screen where the message will begin. DISPLY prints a character at a time, until all characters are displayed. The screen location of the character is determined by its address in screen RAM. Since the CRT is capable of displaying 506 characters (23 rows by 22 columns), there are 506 bytes of memory which define the entire screen. The area of screen RAM for the VIC 20 is $1E00-$1FF9.

Each character on the screen also has a corresponding location in memory which determines its color. This section of memory is called color RAM and covers $9600-$97F9. The subroutine COLOR, called just after SCRN1 in the main sequence, specifies all character colors for the rest of the main program. COLOR writes a color code (0-7) to color RAM for each character on the screen. This subroutine makes the characters in lines 1-4 blue, line 5 black, line 6 blue, lines 7-14 black, line 15 red, line 16 blue, and lines 17-23 black.

Calculation

Calculation subroutines are defined as those which either convert or actually calculate data. The calculation subroutines include DECHEX, HEXDEC, FEED, HEAT, M8BY8, and D16BY8.

The subroutine DECHEX converts a three-digit decimal number less than 200 into a two-digit (1 byte) hexadecimal number. The decimal number to be converted must be in ASCII format, which lends itself for

use after a "FECTH" subroutine because keyboard input is ASCII. The hexadecimal result is left in the accumulator and memory [7].

HEXDEC converts a one byte hexadecimal number to a packed, two-byte decimal number. Location $1035 contains the most significant decimal digit and $1036 contains the least significant two digits. This subroutine is capable of obtaining a decimal number in the range 0-255 [7].

The two subroutines which calculate the hexadecimal control parameter setpoints use the DECHEX subroutine. These subroutines, FEED and HEAT, convert the operator's ASCII format setpoints to hexadecimal values.

FEED calculates the hexadecimal setpoint for motor 2 from the operator's 3-digit ASCII feed rate input. After calling DECHEX, the routine calculates the setpoint for motor 2 from the equation below.

$$x = 4602 \frac{r}{FR} \tag{5}$$

where r = output drive wheel radius,

FR = desired feed rate,

x = calculated motor 2 setpoint.

FEED also calculates the hexadecimal setpoint for motor 1 using the operator's 3-digit ASCII entry for desired stretch-ratio. The ASCII setpoint is first converted to hexadecimal. The equation below then determines the set-point for motor 1.

$$y = (SR)(X) \tag{6}$$

where SR = desired stretch-ratio,

x = calculated motor 2 setpoint,

y = calculated motor 1 setpoint.

The subroutine HEAT performs no calculations. It simply converts the 3-digit ASCII set-points for temperatures 1 and 2 to one-byte hexadecimal values. DECHEX is used to perform the conversions.

The subroutines M8BY8 and D16BY8 perform all the multiplicatons and divisions in the process controller program. M8BY8 multiplies two 8-bit numbers to get a 16-bit product. The 8-bit multiplier and multiplicand must be in $1025 and $1026, respectively. The product is left in $1023 (MSB) and $1024 (LSB). The subroutine D16BY8 divides a 16-bit dividend by an 8-bit divisor. The dividend must be located in $1020 (MSB) and $1021 (LSB). In addition, the dividend and divisor must be unsigned hexadecimal numbers. This restricts the range of the divisor to 1-127, decimal. The divisor must be placed in location $1022. After D16BY8 is called, the quotient is left in $1021 and the remainder in $1020 [11].

Control

The control subroutines are those which affect the motors and heaters. These subroutines include R1, R2, RB, S1, S2, H1, H2, HB, O1, O2, CONT, RUN, STOP, QUIT, SPBAR, TNEW, CNTHTR, and ALARM.

The subroutines R1, R2 and RB switch on motor 1, motor 2, and both motors, respectively. This is accomplished by writing to port B of VIA 1. Bits 2 and 3 of this port determine which motor(s) will be on. A "zero" written to the appropriate bit(s) will allow the motor(s) to run; a "one" will not. The subroutines S1 and S2 turn off motor 1 and motor 2, respectively. By writing a "one" to bit 2 or 3, motor 1 or motor 2 (or both) can be switched off. These motor-switching subroutines are used only during the process preparation sequence.

The subroutines H1, H2, HB, 01 and 02 are called only during the heater preparation sequence. They are similar to the motor-switching routines. H1, H2 and HB turn on heater 1, heater 2, and both heaters, respectively. This is done by enabling Timer 1 of VIA 2, 3 or both to produce a square wave on pin PB7. The enabled heater(s) is sent a 100 percent duty cycle control signal by writing a very small counter value to the Timer 1 registers. The subroutines 01 and 02 turn off heaters 1 and 2, respectively, by disabling Timer 1 of the appropriate VIA.

The subroutine CONT affects only the process motors and is used to advance the operator to the processing sequence following SCRN3 and SCRN6. CONT first turns off motors 1 and 2. Then it sets a flag in memory to indicate CONT was entered from the keyboard and returns to the calling program.

Four other subroutines (RUN, STOP, QUIT, and SPBAR) are also called only when the operator enters the appropriate code on the keyboard. The RUN subroutine switches on both motors when the operator enters the code RUN during display of SCRN8. STOP turns off both motors if the operator

enters STOP during display of SCRN9. The subroutine QUIT turns off both the motors and heaters if the operator enters QUIT on the keyboard during display of SCRN 7, 8 or 9. QUIT terminates the process by blanking the screen with the color blue and then entering an endless loop. The subroutine SPBAR also turns off the motors and heaters. SPBAR is called if the operator presses the space bar key during display of SCRN3, 6, 7, 8 or 9. Program execution will then resume rather than terminate.

Two subroutines which affect only the heaters are TNEW and CNTHTR. TNEW is called by CNTHTR to update a temperature on the screen. CNTHTR specifies which temperature will be updated by allowing one of the temperature probes to transmit its analog signal through the 4016 switch (IC 901) to the A/D converter. TNEW accepts the converted data by means of a hand-shaking sequence between the microprocessor and the A/D converter. First, TNEW initializes a digit select test byte, "DIGSEL," which is used to test for the desired digit selection. This is necessary because the A/D converter presents four BCD (Binary Coded Decimal) digits to the processor in the order most significant digit (MSD) to least significant digit (LSD). "DIGSEL" must be shifted to test for the desired digit since the A/D converter also sends a digit select bit to the processor with the corresponding BCD digit. After "DIGSEL" is initialized, the End of Conversion (EOC) interrupt flag is cleared by TNEW. The processor then waits until this flag is reset. This ensures that all converted data will be taken from the same conversion cycle. When the EOC flag is reset, digits are presented,

read, and displayed on the screen. Only three digits are displayed on the screen because the fourth digit gives the temperature to the closest tenth of a degree Celsius, which is not critical for this application. After all digits have been presented and accepted, program execution returns to CNTHTR.

CNTHTR calls TNEW twice, once to update temperature 1 and once for temperature 2. Each temperature is updated four times per second since the conversion rate of the A/D converter is eight per second. For each temperature reading, CNTHTR checks for an overheat condition, and if overheating has not occurred, implements a heater control signal based on the error between desired and actual temperature. An overheat condition is detected if a temperature is more than 10°C above the desired level. After detection, the subroutine ALARM is called to notify the operator of the overheat condition. If overheating did not occur, CNTHTR sends the control signal to the appropriate heater after computing the difference between desired and actual temperature. If the actual temperature is more than 10°C below desired, a signal of 100 percent duty cuycle (full on) is sent to the heater. If the actual is between 1 and 10°C above desired temperature, a 17.5 percent duty cycle square wave is sent. If the error between desired and actual temperature is in the range 1 to 10°C below desired, then a signal based on the magnitude of the error is sent to the heater. This is achieved by using a look-up table in EPROM which contains the counter values to be placed in the appropriate VIA Timer 1 register. All heater control signals are implemented by writing data to the appropriate Timer 1

registers. Table 6 summarizes the relationship between error signal and resultant duty cycle of the heater control signal. Notice that this control technique uses a proportional band where the duty cycle varies linearly with the error signal.

The last "control" subroutine is ALARM. This subroutine notifies the operator of an overheat condition and is called by CNTHTR. First, ALARM turns off both motors and both heaters. It then displays a message indicating the occurrence of an overheating condition and makes the entire screen red. Next, an audible alarm is programmed by sending three frequencies to the system monitor speaker. This alarm remains on, and the screen remains red, until the operator presses a key on the keyboard. Program flow returns to the calling routine (CNTHTR) after a key is pressed.

Interrupt Processing

As discussed in the VIC 20 review section, the VIC 20 produces an interrupt request at a rate of 60 per second. Each time an $\overline{\text{IRQ}}$ interrupt request occurs, program flow jumps to the location contained in $0314 and $0315. The EPROM initialization sequence specifies this location to be $4000. Thus, the interrupt bypass routine begins at $4000.

The interrupt bypass routine, CLOCK, performs two functions. It updates the system clock and presents the actual feed rate and stretch-ratio parameters on the screen.

Table 6. Relationship Between Error Signal and Heater Power Control Signal

Error*	Hexadecimal Entry		Duty Cycle (% on)
	MSB	LSB	
10 to 0	FF	FF	17%
1	FF	FF	17%
2	E0	00	20%
3	B3	00	25%
4	80	00	35%
5	5A	00	50%
6	4A	00	60%
7	40	00	70%
8	38	00	80%
9	32	00	90%
Greater than 10	01	01	100%

*Error = Desired Temperature - Actual Temperature

The 24-hour clock will be updated every five seconds if a flag, "FLAG," in memory is set. If the flag is not set, the bypass routine immediately jumps to the VIC 20 interrupt routine located at $EABF. In the main program "FLAG" is set just prior to display of SCRN3. This enables the system clock to be updated. A 16-bit counter in memory is used to detect when five seconds has elapsed, since three-hundred interrupt requests occur in a five second period. The time on the screen is updated in a cascaded manner. If the least significant seconds digit (SEC2) equals 10, then the most significant seconds digit (SEC1) is incremented by one and SEC2 is zeroed. Otherwise program flow jumps to the feed rate (FR) and stretch-ratio (SR) section of CLOCK. If, after being incremented, SEC1 equals 60, the least significant minutes digit (MIN2) is incremented by one and SEC1 is zeroed. Otherwise, program flow jumps to the FR/SR section of CLOCK. The rest of the clock update routine follows a similar pattern when updating the most significant minutes digit (MIN1), the least significant hours digit (HR2), and the most significant hours digit (HR1). When the clock time reaches 23:59:55, the next time displayed is 01:00:00.

The FR/SR section of the interrupt bypass routine follows the clock update section. The feed rate and stretch-ratio parameters will be updated on the screen if a flag is set in memory. Otherwise program flow returns to the VIC 20 interrupt routine. The main program sets this flag prior to calling SCRN9 which notifies the operator that the process is running. After the FR/SR flag is enabled, the interrupt bypass routine calculates the feed rate and stretch-ratio of the material being processed.

The feed rate is calculated from the speed of motor 2. Motor 2 speed is detected by Timer 2 of VIA 3. Each digital tachometer feedback pulse decrements the 16-bit Timer 2 counter by one, thus providing a means to calculate feed rate. The actual number of feedback pulses over a five second period is used to calculate average feed rate from the equation

$$FR = (COUNT)(R)\ \frac{2\pi}{640} \tag{7}$$

where COUNT = number of pulses over 5 second period

R = output drive radius, in.,

FR = feed rate, in./min.

The interrupt bypass routine obtains the count information by reading the Timer 2 registers. After calculating the feed rate using M8BY8 and D16BY8, the hexadecimal result must be converted to decimal, and then to ASCII for screen display. The subroutine HEXDEC is used to convert to decimal, and CLOCK does the ASCII conversion and display.

Stretch-ratio is updated in a similar fashion. The stretch-ratio calculation involves the speed of motor 1. Motor 1 speed is detected by Timer 2 of VIA 2. The Timer 2 registers give the necessary information to calculate the stretch-ratio, which is given by

$$SR = \frac{COUNT2}{COUNT1} \tag{8}$$

where COUNT2 = number of pulses from motor 2,

COUNT1 = number of pulses from motor 1,

SR = stretch-ratio.

After CLOCK obtains the count from motor 1, it calculates stretch-ratio by using D16BY8. Before calling D16BY8, however, CLOCK normalizes both motor counts by shifting them right until COUNT1 is the maximum value obtainable less than 127. (Remember, D16BY8 requires the divisor to be 127 or less.) Also, before calling D16BY8, CLOCK multiplies COUNT2 by 10 to get stretch-ratio significance to the closest tenth. After calling D16BY8, the subroutine HEXDEC is used to convert the "stretch-ratio times ten" to decimal. CLOCK then converts this number to ASCII and displays it with the decimal point shifted left once to obtain the true stretch-ratio. Program flow then jumps to the VIC 20 interrupt routine at $EABF to complete the interrupt request processing.

V. SYSTEM EVALUATION

The described electrical and mechanical systems performed exactly as designed and expected.

Electrical System

When turned on, the computer jumps directly to the process control program. Also, all interface and memory devices function properly. While designing the electrical systems two computer interfacing problems were encountered and quickly solved. One involved enabling the data bus at the proper time for communication with the memory and interface chips. Figure 6 in the VIC 20 Interface Circuitry section shows how this problem was solved using a 7420 dual 4-input NAND gate. The other problem encountered also dealt with timing. It was discovered that all 6522 Versatile Interface Adapters (VIA's) require the R/W control pulse to be active for the entire clock cycle. The VR/W signal from the VIC 20 memory expansion connector is only active during the positive pulse of the clock cycle whereas the CR/W signal is active during the entire clock cycle. Therefore, the CR/W signal was used with all VIA's.

The heater control signals were tested under simulated changes in temperature. A power supply was connected to both temperature probe inputs and the voltage was varied. Since the temperature signals from the probes will be delivered such that each 0.010 volt equals 1°C, a voltage in the range 0 to 1.59 could be supplied by the power supply without exceeding the range specified by the control program. Thus, the voltage level was varied around several equivalent temperature set-

points and the duty cycle of the heater control signals was observed. The duty cycle of the heater control signals did, in fact, vary linearly with the error signal. As the error decreased, so did the duty cycle. The duty cycle of the heater control signals can also be changed by altering the hardware, which may need to be done to "tune" the controller to a specific control volume if varying the heater power rating does not give an acceptable response. Altering the hardware is not expected to be necessary since Mears [4] used the same approach and achieved results within 5 percent of set-point temperature while sampling each probe at a rate of once per second. The described process controller samples each probe four times per second and has a wider proportional band than Mears' controller. This would allow the control volume, once designed, to be analyzed using linear control system theory in the proportional band.

Mechanical System

The motor controllers performed exactly as desired. While being controlled, no measurable sag in shaft velocity was observed for applied loads up to the specified maximum load over a wide control range, 50 RPM to 1500 RPM. In this range, each motor reached its steady state operating speed well within 0.5 seconds after switching the motor on. This delay is due to the inertial response of the motor because the maximum delay time of the controller is 50 microseconds. Both motors are rated at 1/4 H.P. at 2500 RPM with specified maximum ratings of 24 volts and 13 amps dc. Each motor has a time constant of approximately

1.4 seconds, which means each motor, driven open-loop, reaches final velocity for a step change in voltage input in about 6 seconds.

Software

No uncommon problems were encountered during software development. The VIC 20 VICMON[R] assembler proved invaluable for testing all keyboard and screen display subroutines on the VIC 20 before final assembly on the AIM 65.

Only one inconsistancy was observed in the software. For two cases of the many tested, the updated feedrate on the screen would include a non-numerical character. The subroutine HEXDEC most likely caused this problem because the decimal equivalent of the hexadecimal number is displayed on the screen. For these two cases, HEXDEC obviously did not obtain a decimal value.

VI. RECOMMENDATIONS AND CONCLUSIONS

The controller designed and developed for this material processing application is essentially error-free. No revisions are necessary to begin processing material. A sealed control volume, a material-handling pulley system, and appropriate heating coils are the remaining components to be designed and built in order to perform an actual process run.

The controlling program also requires no changes or revisions prior to a process run. If, at a later time, it is necessary to alter the program, the following recommendations may be helpful.

1. Use prioritized interrupts to update the system clock, update feed rate and stretch-ratio, and sample and control system temperature.
2. Program a velocity curve for each motor on start-up to prevent any initial "jerk" which may occur due to the quickness of the motor controllers.
3. Include a routine to trap invalid feed rate and stretch-ratio combinations.

REFERENCES

1. Arp, L. J., "Microporous Membrane Production Process Outline," Confidential Industrial Report to Aequitron Medical, Inc., January 1984.

2. Deshpande, P. B., and R. H. Ash, Elements of Computer Process Control with Advanced Control Applications, Instrument Society of America, Research Triangle Park, North Carolina, 1982, pp. 143-157.

3. R6500 Microcomputer System Hardware Manual, Rockwell International Corp., Document No. 29650, N31, Rev. 1, Anaheim, California, 1978.

4. Mears, D. T., "A Microcomputer-Based Temperature and Humidity Control System for Respiratory Therapy," Master Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1980.

5. Finkel, A., N. Harris, P. Higginbottom, and M. Tomczyk, VIC 20 Programmer's Reference Guide, Commodore Business Machines, Inc. and Howard W. Sams and Co., Inc., King of Prussia, Pennsylvania, 1983, pp. 182-211.

6. Hampshire, N, VIC Revealed, Hayden Book Company, Inc., Rochelle Park, New Jersey, 1982, pp. 44-102.

7. De Jong, M. L., Programming and Interfacing the 6502, with Experiments, Howard W. Sams and Co., Inc., Indianapolis, Indiana, 1980, pp. 132-200.

8. MC14433, Motorola Semi Conductor Products Inc., Document O. DS-9423, Austin, Texas, 1976.

9. AD580 Integrated Circuit Low Drift Voltage Reference, Analog Devices Company, Norwood, Massachusetts, 1975.

10. Takahashi, Y., M. Rabins, and D. Auslander, Control and Dynamic Systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1972, pp. 467-485.

11. Leventhal, L. A., 6502 Assembly Language Programming, Osborne/McGraw-Hill, Berkeley, California, 1979, Section 8, pp. 12-16.

APPENDIX A

Software Listing

This appendix lists the program used for the process control system. The program was written in 6502 assembly language and was assembled on a Rockwell AIM 65 microcomputer. Comments were added after assembly to the right of the assembly code and are preceded by a semi-colon.

The software listing contains several symbols that need to be defined. The symbol "*=$XXXX" sets the program counter address to XXXX, thus specifying the starting address of assembly. The dollar sign, "$", specifies a hexadecimal number. Throughout the listing the symbol "==YYYY" specifies the hexadecimal address YYYY of the upcoming machine code byte.

The assembly code is organized as most assembly language programs are. From left to right there is the machine code, a mnemonic operation code, an operand, and a comment. Some lines do not contain comments.

Abbreviations used in the comments are defined below.

Abbreviation	Meaning
LSB	Least-Significant Byte
MSB	Most-Significant Byte
FR	Feed rate
SR	Stretchratio
CR	Carriage Return
SP	Space Bar
HR1	Most-Significant Hours digit
HR2	Least-Significant Hours digit
MIN1	Most-Significant Minutes digit
MIN2	Least-Significant Minutes digit
SEC1	Most-Significant Seconds digit
SEC2	Least-Significant Seconds digit
ACR	Auxiliary Control Register
DDR	Data Direction Register
PCR	Peripheral Control Register
IER	Interrupt Enable Register
IFR	Interrupt Flag Register

There are three EPROM memory locations noted with an asterisk [*] in the program. These values, listed below, determine the sytem's motor controller setpoints and feed rate/stretch-ratio update calculations. They may need to be altered for use with different motor controllers or system configurations.

Asterisked Location	Function
$2114	MSB Controller Constant
$2119	LSB Controller Constant
$40EE	FR Divisor Constant

The governing equations are given by:

Motor 1 Set Point = (Controller Constant x Drive Radius)/Desired FR)

Actual FR = (Motor 1 Count x Drive Radius)/(Divisor Constant)

```
==0000
        *=$A000
==A000
   EQUATE STATEMNTS
==A000    I01PB=$9800     ;  Port B Data Registers
==A000    I02PB=$9A00
==A000    I03PB=$9B00
==A000    I01PA=$9801     ;  Port A Data Registers
==A000    I02PA=$9A01
==A000    I03PA=$9B01
==A000    I01BDD=$9802    ;  Port B Data Direction Registers
==A000    I02BDD=$9A02
==A000    I03BDD=$9B02
==A000    I01ADD=$9803    ;  Port A Data Direction Registers
==A000    I02ADD=$9A03
==A000    I03ADD=$9B03
==A000    I02T1L=$9A04    ;  Timer Control Registers
==A000    I02T1H=$9A05
==A000    I02T2L=$9A08
==A000    I02T2H=$9A09
==A000    I03T1L=$9B04
==A000    I03T1H=$9B05
==A000    I03T2L=$9B08
==A000    I03T2H=$9B09
==A000    I01ACR=$980B    ;  Auxiliary Control Registers
==A000    I02ACR=$9A0B
==A000    I03ACR=$9B0B
==A000    I01PCR=$980C    ;  Peripheral Control Registers
==A000    I02PCR=$9A0C
```

```
==A000   I03PCR=$9B0C
==A000   I01FR=$980D     ;  Interrupt Flag Register
==A000   I01IER=$980E    ;  Interrupt Enable Register
==A000   ALRM=$1028      ;  Alarm condition flag
==A000   LAST=$100C      ;  24-hour clock jiffy detector
==A000   FLAG=$100D      ;  Clock update flag
==A000   CNTR1=$1010     ;  Clock update counter, MSB
==A000   CNTR2=$1011     ;  Clock update Counter, LSB
==A000   HR1=$1F52       ;  Hours digit
==A000   HR2=$1F53       ;  Hours digit
==A000   MIN1=$1F55      ;  Minutes digit
==A000   MIN2=$1F56      ;  Minutes digit
==A000   SEC1=$1F58      ;  Seconds digit
==A000   SEC2=$1F59      ;  Seconds digit
==A000   DIGSEL=$1017    ;  Digit select byte for A/D converter
==A000   HEX=$1013       ;  Temporary storage for hexadecimal conversion
==A000   GETIN=$FFE4     ;  Keyboard Character-accept subroutine
==A000   PNTR1=$1000     ;  Pointer for storage of keyboard input
==A000   TEMP=$100A      ;  Temporary storage byte
==A000   THR1=$1ED3      ;  Temporary hours digit
==A000   THR2=$1ED4      ;  Temporary hours ditit
==A000   TMIN1=$1ED6     ;  Temporary minutes digit
==A000   TMIN2=$1ED7     ;  Temporary minutes digit
==A000   TSEC1=$1ED9     ;  Temporary seconds digit
==A000   TSEC2=$1EDA     ;  Temporary seconds digit
==A000   COLON1=$1ED5    ;  Colon for clock display
==A000   COLON2=$1ED8    ;  Colon for clock display
==A000   ERRFLG=$100B    ;  Error flag
==A000   FRSR=$1027      ;  Feed rate/stretch-ratio update flag
==A000   TABLE=$46E3     ;  Start of heater control lookup table
==A000   INTH=$40        ;  Address of interrupt bypass routine, MSB
==A000   INTL=$00        ;  Address of interrupt bypass routine, LSB
==A000   M1L=$00         ;  Message 1 address , LSB
==A000   M1H=$42         ;                     , MSB
==A000   M2L=$14         ;  Message 2 address , LSB
==A000   M2H=$42         ;                     , MSB
==A000   M2AL=$A4        ;  Message 2A address, LSB
==A000   M2AH=$46        ;                     , MSB
==A000   M3L=$26         ;  Message 3 address , LSB
==A000   M3H=$42         ;                     , MSB
==A000   M4L=$9F         ;  Message 4 address , LSB
==A000   M4H=$42         ;                     , MSB
==A000   M5L=$B5         ;  Message 5 address , LSB
==A000   M5H=$42         ;                     , MSB
==A000   M6L=$C8         ;  Message 6 address , LSB
==A000   M6H=$42         ;                     , MSB
==A000   M7L=$08         ;  Message 7 address , LSB
==A000   M7H-$43         ;                     , MSB
==A000   M8L=$8C         ;  Message 8 address , LSB
==A000   M8H=$43         ;                     , MSB
==A000   M9L=$0E         ;  Message 9 address , LSB
==A000   M9H=$44         ;                     , MSB
```

```
==A000    M10L=$8D        ;  Message 10 address, LSB
==A000    M10H=$44        ;                      , MSB
==A000    M11L=$9F        ;  Message 11 address, LSB
==A000    M11H=$44        ;                      , MSB
==A000    M12L=$D8        ;  Message 12 address, LSB
==A000    M12H=$44        ;                      , MSB
==A000    M14L=$5C        ;  Message 14 address, LSB
==A000    M14H=$45        ;                      , MSB
==A000    M15L=$B4        ;  Message 15 address, LSB
==A000    M15H=$45        ;                      , MSB
==A000    M16L=$22        ;  Message 16 address, LSB
==A000    M16H=$46        ;                      , MSB
==A000    M17L=$90        ;  Message 17 address, LSB
==A000    M17H=$46        ;                      , MSB
==A000    RUNL=$C7        ;  Run motors jump address, LSB
==A000    RUNH=$A1        ;                         , MSB
==A000    STOPL=$94       ;  Stop motors jump address, LSB
==A000    STOPH=$A1       ;                          , MSB
==A000    QUITL=$00       ;  Terminate process jump address, LSB
==A000    QUITH=$A0       ;                                  , MSB
==A000    CHNGEL=$1A      ;  Change parameters jump address, LSB
==A000    CHNGEH=$A1      ;                                  , MSB
;MAIN PROGRAM
   (INITIALIZATION ROUTINE)
09        BYTE  $09, $A0  ;  Auto-start jump address
   $09
A0
09
A0        BYTE  $A0, $41  ;  Auto-start sequence
   $30
41
30
C3        BYTE  $C3, $02
   $CD
C2
CD
A0FF      LDY   #$FF      ;  Initialize output port DDR's
8C0298    STY   I01BDD
8C038A    STY   I02ADD
==A011
8C039B    STY   I03ADD
A20D      LDX   #$0D      ;  Turn off motors and
8E0098    STX   I01PB     ;  Open temperature probe switches
8E1C10    STX   $101C
EA        NOP
EA        NOP
208DFD    JSR   $FD8D     ;  RAM clear and memory check
==A021
2052FD    JSR   $FD52     ;  Set VIC 20 vectors
20F9FD    JSR   $FDF9     ;  Initialize I/O in VIC 20
2018E5    JSR   $E518     ;  Initialize screen
A900      LDA   #$00      ;  Initialize input port DDR's
```

```
8D0398  STA   IO1ADD
A980    LDA   #$80
==A031
8D029A  STA   IO2BDD
8D029B  STA   IO3BDD
A0FF    LDY   #$FF        ;  Initialize motor set points
8C009A  STY   IO2PB
8C009B  STY   IO3PB
A900    LDA   #$00        ;  Initialize ACR's
==A041
8D0B98  STA   IO1ACR
A220    LDX   #$20        ;  Disable heaters
8E0B9A  STX   IO2ACR
8E0B9B  STX   IO3ACR
8D0C98  STA   IO1PCR      ;  Initialize PCR's
8D0C9A  STA   IO2PCR
==A052
8D0C9B  STA   IO3PCR
A97F    LDA   #$7F        ;  Initialize IER's
8D0E98  STA   IO1IER
A990    LDA   #$90
8D0E98  STA   IO1IER
A97F    LDA   #$7F        ;  Initialize IFR
8D0D98  STA   IO1FR
==A064
A900    LDA   #$00        ;  Disable clock update
8D0D10  STA   FLAG
8D2710  STA   FRSR        ;  Disable feed rate/stretch-ratio update
A9C0    LDA   #$C0        ;  Set preparation motor speeds
8D019A  STA   IO2PA
8D019B  STA   IO3PA
==A074
A940    LDA   #INTH       ;  Initialize interrupt bypass address
8D1503  STA   #0315
A900    LDA   #INTL
8D1403  STA   #0314
58      CLI               ;  Clear interrupt disable flag

: (MAIN SEQUENCE)
200C23  JSR   SCRN1       ;  Display title screen
A926    LDA   #$26        ;  Wait 12 seconds
==A084
8D0A10  STA   $100A
==A087  DCRM
A2FF    LDX   #$FF
==A089  DCRX
A0FF    LDY   #$FF
==A08B  DCRY
88      DEY
D0FD    BNE   DCRY
CA      DEX
D0F8    BNE   DCRX
```

```
CE0A10   DEC    $100A
D0F1     BNE    DCRM
207127   JSR    COLOR       ;  Specify screen character colors
20C923   JSR    SCRN2       ;  Display 24-hour clock entry screen
==A090   PRINT2
20FCA1   JSR    FETCH1      ;  Get clock time from user
20AEA2   JSR    CHECK1      ;  Check for valid input
A900     LDA    #$00
CD0B10   CMP    ERRFLG      ;  Input Valid?
F006     BEQ    DIAM        ;  If yes, continue
20D125   JSR    ERROR       ;  Display error message
==A0AC
4C9CA0   JMP    PRINT2      ;  Get new clock time
==A0AF   DIAM
20EA25   JSR    CLRLNE      ;  Clear error message
202A24   JSR    SCRN2A      ;  Display drive wheel diameter entry screen
==A0B5   PRINT3
2021A3   JSR    FETCH2      ;  Accept and check user input
A900     LDA    #$00
CD0B10   CMP    ERRFLG      ;  Input valid?
F006     BEQ    CNVRT       ;  If yes, continue
20D125   JSR    ERROR       ;  Display error message
4CB5A0   JMP    PRINT3      ;  Get new diameter
==A0C5   CNVRT
20EA25   JSR    CLRLNE      ;  Clear error message
ADB91F   LDA    $1FB9       ;  Prepare to convert diameter to hex
8D0110   STA    $1001
ADBA1F   STA    $1FBA
8D0210   STA    $1002
A900     LDA    #$00
==A0D6
8D0010   STA    $1000
20A822   JSR    DECHEX      ;  Convert drive wheel diameter to hex
4A       LSR    A           ;  Get drive wheel radius
8D1410   STA    $1014       ;  Save radius
A000     LDY    #$00
==A0E2   CLK
B9D31E   LDA    $1ED3,Y     ;  Display clock time in correct place
99521F   STA    #1F52,Y     ;  on screen
A930     LDA    #$30
8D591F   STA    #1F59
C8       INY
C008     CPY    #$08
D0F0     BNE    CLK
==A0F2
A901     LDA    #$01        ;  Enable clock update on screen
8D0D10   STA    FLAG
204324   JSR    SCRN3       ;  Display process preparation screen
==A0FA   PREP
2095A3   JSR    FETCH3      ;  Get keyboard input
20E5A3   JSR    CHECK2      ;  Check for valid code
A900     LDA    #$00
```

```
CD0B10   CMP    ERRFLG      ;  Input valid?
F006     BEQ    CNTRL       ;  If yes, continue
20D125   JSR    ERROR       ;  If no, display error message
==A10A
4CFAA0   JMP    PREP        ;  Jump to get new input
==A10D   CNTRL
20EA25   JSR    CLRLNE      ;  Clear error message
A900     LDA    #$00
CD1D10   CMP    $101D       ;  Check for CONT code input
F0E3     BEQ    PREP        ;  If not, jump back for new code
204527   JSR    STOP        ;  If so, stop motors and continue
==A11A   PRMTR
20A424   JSR    SCRN4       ;  Display control parameter input screen
20A2A4   JSR    FECTH4      ;  Accept and check for valid entries
200021   JSR    FEED        ;  Convert and store feed rate/stretch-ratio
                            ;  inputs
209D21   JSR    HEAT        ;  Convert and store temperature inputs
20ED24   JSR    SCRN5       ;  Display heater control message
A926     LDA    #$26        ;  Wait 12 seconds
==A12B
8D0A10   STA    $100A
==A12E   DWNM
A2FF     LDX    #$FF
==A130   DWNX
A0FF     LDY    #$FF
==A132   DWNY
88       DEY
D0FD     BNE    DWNY
CA       DEX
D0F8     BNE    DWNX
CE0A10   DEC    $100A
D0F1     BNE    DWNM
201125   JSR    SCRN6       ;  Display heater control codes
==A140   HOT
2095A3   JSR    FECTH3      ;  Get code from keyboard
20E4A6   JSR    CHECK3      ;  Check for valid code
A900     LDA    #$00
CD0B10   DMP    ERRFLG      ;  Input valid?
F006     BEQ    HEATON      ;  If so, continue
20D125   JSR    ERROR       ;  If not, display error message
==A150
4C40A1   JMP    HOT         ;  Jump to get new code
==A153   HEATON
20EA25   JSR    CLRLNE      ;  Clear error message
A900     LDA    #$00
CD1D10   CMP    $101D       ;  CONT code entered?
F0E3     BEQ    HOT         ;  If not, jump back for new code
205825   JSR    SCRN7       ;  If so, display waiting for heaters screen
==A160   DOHEAT
20CE21   JSR    CNTHTR      ;  Control heaters
A902     LDA    #$02
CD1810   CMP    $1018       ;  Temp. 1 w/in 2 C of set-point?
```

```
1003     BPL    H2CHK       ;  If so, check temp. 2
4C74A1   JMP    KEY         ;  If not, check for keyboard input
==A16D   H2CHK
A902     LDA    #$02
CD1910   CMP    $1019       ;  Temp. 2 w/in 2 C of set-point?
1020     BPL    NEWSCN      ;  If so, continue with new screen
==A174   KEY
20E4FF   JSR    GETIN       ;  If not, check for keyboard input
C900     CMP    #$00
F0E5     BEQ    DOHEAT      ;  If no, jump back to control heaters
2095A3   JSR    FETCH3      ;  If yes, get user input code
200020   JSR    CHECK4      ;  Check for valid input
A900     LDA    #$00
CD0B10   CMP    ERRFLG      ;  Valid input?
==A186
F006     BEQ    PERFRM      ;  If yes, branch to perform function
20D125   JSR    ERROR       ;  If not, display error message
4C60A1   JMP    DOHEAT      ;  Jump back to control heaters
==A18E   PERFRM
20EA25   JSR    CLRLNE      ;  Clear error message
6C1510   JMP    ($1015)     ;  Jump to new location based on user input
==A194   NEWSON
20EA25   JSR    CLRLNE      ;  Clear error message line
207125   JSR    SCRN8       ;  Display process ready to run screen
==A19A   SETHTR
20CE21   JSR    CNTHTR      ;  Control heaters
A900     LDA    #$00
CD2810   CMP    ALRM        ;  Did an overheat alarm occur?
F003     BEQ    OKGO1       ;  If not, continue
4C1AA1   JMP    PRMTR       ;  If so, jump back to get new control
                            parameters
==A1A7   OKGO1
20E4FF   JSR    GETIN       ;  Check if a key was pressed
C900     CMP    #$00
F0EC     BEQ    SETHTR      ;  If not, keep controlling heaters
2095A3   JSR    FETCH3      ;  If so, get user's code from keyboard
200020   JSR    CHECK4      ;  Check for valid input
A900     LDA    #$00
CD0B10   CMP    ERRFLG      ;  Valid Input?
==A1B9
F006     BEQ    DOIT        ;  If yes, jump to perform function
20D125   JSR    ERROR       ;  If no, display error message
4C9AA1   JMP    SETHTR      ;  and keep controlling heaters
==A1C1   DOIT
20EA25   JSR    CLRLNE      ;  Clear error message
6C1510   JMP    ($1015)     ;  Jump to new location based on user input
==A1C7   RUNPRC
208A25   JSR    SCRN9       ;  Display process running screen
A901     LDA    #$01
8D2710   STA    FRSR        ;  Enable screen update of feed rate/stretch-
                            ratio
==A1CF   HTRCNT
```

```
20CE21   JSR    CNTHTR      ;  Control heaters
A900     LDA    #$00
CD2810   CMP    ALRM        ;  Did an overheat alarm occur?
F003     BEQ    OKGO2       ;  If not, continue
4C1AA1   JMP    PRMTR       ;  If so, jump back to get new parameters
==A1DC   OKGO2
20E4FF   JSR    GETIN       ;  Check if a key pressed
C900     CMP    #$00
F0EC     BEQ    HTRCNT      ;  If not, keep controlling heaters
2095A3   JSR    FETCH3      ;  If so, get user's code from keyboard
200020   JSR    CHECK4      ;  Check for valid input
A900     LDA    #$00
CD0B10   CMP    ERRFLG      ;  Valid input?
==A1EE
F006     BEQ    NEWLOC      ;  If yes, jump to perform function
20D125   JSR    ERROR       ;  If not, display error message
4CCFA1   JMP    HTRCNT      ;  and keep controlling heaters
==A1F6   NEWLOC
20EA25   JSR    CLRLNE      ;  Clear error message
6C1510   JMP    ($1015)     ;  Jump to new location based on user's input

;SUBROUTINES
; FETCH1                    ;  (Clock Input and Display)
==A1FC   FETCH1
48       PHA
==A1FD   AGAIN
A22D     LDX    #$2D        ;  Display underscore
8EE91E   STX    $1EE9
A000     LDY    #$00        ;  Initialize character counter
==A204   LOOP1
98       TYA
48       PHA                ;  Save character counter on stack
==A206   WAIT
20E4FF   JSR    GETIN       ;  Get character from keyboard que
0900     CMP    #$00
F0F9     BEQ    WAIT
8D0A10   STA    TEMP        ;  Store ASCII keyboard input temporarily
68       PLA
A8       TAY                ;  Restore character counter
AD0A10   LDA    TEMP        ;  Get ASCII character
990010   STA    PNTR1,Y     ;  Save it in memory
==A218
C8       INY                ;  Increment character counter
C90D     CMP    #$0D        ;  CR entered yet?
D003     BNE    GO          ;  If no CR, continue
4CA0A2   JMP    RET         ;  If CR, return from subroutine
==A220   GO
C001     CPY    #$01        ;  Is character entered HR1?
F017     BEQ    TIME1       ;  If yes, branch to display on screen
C002     CPY    #$02        ;  HR2?
F023     BEQ    TIME2       ;  If yes, branch to display
C003     CPY    #$03        ;  MIN1?
```

```
F02F     BEQ   TIME3      ;  If yes, branch to display
C004     CPY   #$04       ;  MIN2?
F040     BEQ   TIME4      ;  If yes, branch to display
==A230
C005     CPY   #$05       ;  SEC1?
F04C     BEQ   TIME5      ;  If yes, branch to display
C006     CPY   #$06       ;  SEC2?
F05D     BEQ   TIME6      ;  If yes, branch to display
4CFDA1   JMP   AGAIN      ;  Get new time inputs if counter >6
==A23B   TIME1
8DD31E   STA   THR1       ;  Display HR1
A220     LDX   #$20       ;  Remove underscore on screen
8EE91E   STX   $1EE9
A22D     LDX   #$2D       ;  Put in new underscore
8EEA1E   STX   $1EEA
4C04A2   JMP   LOOP1      ;  Get next character
==A24B   TIME2
8DD41E   STA   THR2       ;  Display HR2
A220     LDX   #$20       ;  Remove underscore
8EEA1E   STX   $1EEA
A22D     LDX   #$2D       ;  Put in new underscore
8EEC1E   LDX   $1EEC
4C04A2   JMP   LOOP1      ;  Get new character
==A25B   TIME3
A23A     LDX   #$3A       ;  Display colon
8ED51E   STX   COLON1
8DD61E   STA   TMIN1      ;  Display MIN1
A220     LDX   #$20       ;  Remove underscore
8EEC1E   STX   $1EEC
A22D     LDX   #$2D       ;  Put in new underscore
8EED1E   STX   $1EED
==A26D
4C04A2   JMP   LOOP1      ;  Get next character
==A270   TIME4
8DD71E   STA   TMIN2      ;  Display MIN2
A220     LDX   #$20       ;  Remove underscore
8EED1E   STX   $1EED
A22D     LDX   #$2D       ;  Put in new underscore
8EEF1E   STX   $1EEF
4C04A2   JMP   LOOP1      ;  Get next character
==A280   TIME5
A23A     LDX   #$3A       ;  Display colon
8ED81E   STX   COLON2
8DD91E   STA   TSEC1      ;  Display SEC1
A220     LDX   #$20       ;  Remove underscore
8EEF1E   STX   $1EEF
A22D     LDX   #$2D       ;  Put in new underscore
8EF01E   STX   $1EF0
==A292
4C04A2   JMP   LOOP1      ;  Get next character
==A295   TIME6
8DDA1E   STA   TSEC2      ;  Display SEC2
```

```
A220     LDX   #$20        ;  Remove underscore
8EF01E   STX   $1EF0
4C04A2   JMP   LOOP1       ;  Get next character
==A2A0   RET
A000     LDY   #$00        ;  Clear all underscores
A920     LDA   #$20
==A2A4   CONTIN
99E91E   STA   $1EE9,Y
C8       INY
C008     CPY   #$08
D0F8     BNE   CONTIN
68       PLA
60       RTS               ;  Return from subroutine

; CHECK1                  ;  (Check Clock Input)
==A2AE   CHECK1
48       PHA
A000     LDY   #$00        ;  Clear character counter
8C0B10   STY   ERRFLG      ;  Clear error flag
B90010   LDA   PNTR1,Y     ;  Get HR1 from memory
C930     CMP   #$30        ;  Is HR1>=0?
B003     BCS   NEXT1       ;  If yes, continue
4C1AA3   JMP   ERR         ;  If not, set error flag
==A2BE   NEXT1
C933     CMP   #$33        ;  Is HR1>=3?
B058     BCS   ERR         ;  If yes, set error flag
C932     CMP   #$32        ;  Is HR1 = 2?
F012     BEQ   CHAR2       ;  If yes, check HR1 for 0-3
C8       INY               ;  Increment character counter
B90010   LDA   PNTR1,Y     ;  Get HR2
C930     CMP   #$30        ;  Is HR2>=0?
B003     BCS   NEXT2       ;  If yes, continue
==A2CE
4C1AA3   JMP   ERR         ;  If not, set error flag
==A2D1   NEXT2
C93A     CMP   #$3A        ;  Is HR2>9?
B045     BCS   ERR         ;  If so, set error flag
4CE7A2   JMP   CHAR3       ;  If not, continue w/MIN1
==A2D8   CHAR2
C8       INY               ;  Increment character counter
B90010   LDA   PNTR1,Y     ;  Get HR2 and check for 0-3
C930     CMP   #$30        ;  Is HR2>=0?
B003     BCS   NEXT3       ;  If yes, continue
4C1AA3   JMP   ERR         ;  If not, set error flag
==A2E3   NEXT3
C935     CMP   #$35        ;  Is HR2>=4?
B033     BCS   ERR         ;  If yes, set error flag
==A2E7   CHAR3
C8       INY               ;  Increment character counter
B90010   LDA   PNTR1,Y     ;  Get next character
C002     CPY   #$02        ;  Is it MIN1?
F00F     BEQ   TSTMS1      ;  If so, test MIN1 validity
```

```
C003     CPY   #$03        ;  Is it MIN2?
F019     BEQ   TSTMS2      ;  If so, test MIN2 validity
C004     CPY   #$04        ;  Is it SEC1?
F007     BEQ   TSTMS1      ;  If so, test SEC1 validity
==A2F7
C005     CPY   #$05        ;  Is it SEC2?
F011     BEQ   TSTMS2      ;  If so, test SEC2 validity
4C1FA3   JMP   RETR        ;  Jump to return
==A2FE   TSTMS1
C930     CMP   #$30        ;  Test MIN1 or SEC1 for 0-5
B003     BCS   NEXT4       ;  If character >=0, continue
4C1AA3   JMP   ERR         ;  If not, set error flag
==A305   NEXT4
C936     CMP   #$36        ;  Is character >=6?
B011     BCS   ERR         ;  If so, set error flag
4CE7A2   JMP   CHAR3       ;  If not, continue w/MIN2 or SEC2
==A30C   TSTMS2
C930     CMP   #$30        ;  Is MIN2 or SEC2 >=0?
B003     BCS   NEXT5       ;  If so, continue
4C1AA3   JMP   ERR         ;  If not, set error flag
==A313   NEXT5
C93A     CMP   #$3A        ;  Is character > 9?
B003     BCS   ERR         ;  If so, set error flag
4CE7A2   JMP   CHAR3       ;  If not, get next character
==A31A   ERR
A201     LDX   #$01        ;  Set error flag in memory
8E0B10   STX   ERRFLG
==A31F   RETR
68       PLA
60       RTS               ;  Return from subroutine

 ; FETCH2
==A321   FETCH2            ;  (Get Drive Wheel Diameter)
A900     LDA   #$00        ;  Clear error flag
8D0B10   STA   ERRFLG
A22D     LDX   #$2D        ;  Display underscore
8ECF1F   STX   $1FCF
A000     LDY   #$00        ;  Initialize character counter
==A32D   CHAR
98       TYA
48       PHA               ;  Save character counter on stack
==A32F   WAIT2
20E4FF   JSR   GETIN       ;  Get character from keyboard que
C900     CMP   #$00
F0F9     BEQ   WAIT2
8D0A10   STA   TEMP        ;  Temporarily store ASCII character
68       PLA
A8       TAY               ;  Restore character counter
AD0A10   LDA   TEMP        ;  Get character
990110   STA   $1001,Y     ;  Save character in memory
==A341
C8       INY               ;  Increment character counter
```

```
C90D     CMP   #$0D        ;  CR entered yet?
D003     BNE   ANOTHR      ;  If not, continue
4C6CA3   JMP   EXIT        ;  If so, exit to check input
==A349   ANOTHR
C001     CPY   #$01        ;  First character entered?
D010     BNE   SECOND      ;  If not, check if second character
A220     LDX   #$20        ;  Remove underscore on screen
8ECF1F   STX   $1FCF
A22D     LDX   #$2D        ;  Put in new underscore
8ED01F   STX   $1FD0
8DB91F   STA   $1FB9       ;  Display first character
==A35A
4C2DA3   JMP   CHAR        ;  Get another character
==A35D   SECOND
C002     CPY   #$02        ;  Second character?
D0C0     BNE   FETCH2      ;  If not, start over
A220     LDX   #$20        ;  If so, remove underscore
8ED01F   STX   $1FD0
8DBA1F   STA   $1FBA       ;  Display second character
4C2DA3   JMP   CHAR        ;  Get next character
==A36C   EXIT
A000     LDY   #$00        ;  Clear character counter
==A36E   NEXTY
B9B91F   LDA   $1FB9,Y     ;  Get a character
C930     CMP   #$30        ;  Is it >=0?
B003     BCS   TSTHI       ;  If so, continue
4087A3   JMP   ERR2        ;  If not, set error flag
==A378   TSTHI
C93A     CMP   #$3A        ;  Is it >=9?
9003     BCC   NEXTCH      ;  If not, check next character
4C87A3   JMP   ERR2        ;  If so, set error flag
==A37F   NEXTCH
C8       INY               ;  Increment character counter
C002     CPY   #$02        ;  Third character?
F008     BEQ   RTRN        ;  If so, return
4C6EA3   JMP   NEXTY       ;  If not, get another character
==A387   ERR2
A901     LDA   #$01        ;  Set error flag in memory
8D0B10   STA   ERRFLG
==A38C   RTRN
A920     LDA   #$20        ;  Clear all underscores
8DCF1F   STA   $1FCF
8DD01F   STA   $1FD0
60       TRS               ;  Return from subroutine

; FETCH2
==A395   FETCH3            ;  (Get and Display Keyboard Input)
A22D     LDX   #$2D        ;  Display underscore
8E4B1F   STX   $1F4B
A000     LDY   #$00        ;  Initialize character counter
==A39C   LOOP3
98       TYA
```

```
48       PHA                ;  Save character counter on stack
==A39E   WATI3
20E4FF   JSR   GETIN        ;  Get character from keyboard que
C900     CMP   #$00
F0F9     BEQ   WAIT3
8D0A10   STA   TEMP         ;  Temorarily store ASCII character
68       PLA
A8       TAY                ;  Restore character counter
AD0A10   LDA   TEMP         ;  Retrieve character
990010   STA   $1000,Y      ;  Save it in memory
==A3B0
C920     CMP   #$20         ;  Check character for space bar, SP
F024     BEQ   RETN         ;  If SP, branch
C90D     CMP   #$0D         ;  If not SP, check for CR
D003     BNE   G03          ;  If not CR, continue
4CD8A3   JMP   RETN         ;  If CR, return
==A3BB   G03
C006     CPY   #$06         ;  More than 6 characters entered yet?
D008     BNE   STORE        ;  If not, store current characters
A220     LDX   #$20         ;  If so, clear underscore
8E511F   STX   $1F51
4C95A3   FETCH3             ;  Start subroutine again
==A3C7   STORE
99351F   STA   $1F35,Y      ;  Store character on screen
A920     LDA   #$20
994B1F   STA   $1F4B,Y      ;  Remove underscore
A92D     LDA   #$2D
994C1F   STA   $1F4C,Y      ;  Put in underscore
C8       INY
4C9CA3   JMP   LOOP3        ;  Get next character
==A3D8   RETN
A000     LDY   #$00         ;  Clear all underscores
A920     LDA   #$20
==A3DC   SPACE
994B1F   STA   $1F4B,Y
C8       INY
C007     CPY   #$07
D0F8     BNE   SPACE
60       RTS                ;  Return from subroutine

; CHECK2
==A3E5   CHECK2             ;  (Check for Process Preparation Codes)
A900     LDA   #$00         ;  Clear error and CONT flag
8D1D10   STA   $101D
8D0B10   STA   ERRFLG
208BA4   JSR   CHKSP        ;  Check if a space bar was entered
C900     CMP   #$00         ;  Was SP entered?
D014     BNE   FINI         ;  If so, return
AD0010   LDA   $1000        ;  If not, get first character
==A3F7
C952     CMP   #$52         ;  Is character = R?
F00E     BEQ   RCHAR        ;  If yes, check for valid code
```

```
C953     CMP    #$53        ;  Is character = S?
F03F     BEQ    SCHAR       ;  If yes, check for valid code
C943     CMP    #$43        ;  Is character = C?
F05C     BEQ    CCHAR       ;  If yes, check for valid code
==A403   BAD1
A901     LDA    #$01        ;  Set error flag
8D0B10   STA    ERRFLG
==A408   FINI
60       RTS                ;  Return from subroutine
==A409   RCHAR              ;  Test for R1,R2, or RB
AD0110   LDA    $1001       ;  Get second character
C931     CMP    #$31        ;  Is it >=1?
B003     BCS    GOOD1       ;  If yes, continue
4C03A4   JMP    BAD1        ;  If not, set error flag
==A413   GOOD1
D00B     BNE    GOOD2       ;  If second character not = 1, continue
AD0210   LDA    $1002       ;  Get third character
C90D     CMP    #$0D        ;  Is it CR?
D0E7     BNE    BAD1        ;  If not, set error flag
20BD26   JSR    R1          ;  If so, turn on motor 1
60       RTS                ;  Return
==A420   GOOD2
C932     CMP    #$32        ;  Check second character for 2 or B
D00B     BNE    GOOD3       ;  If it is not = 2, branch
AD0210   LDA    $1002       ;  Get third character
C90D     CMP    #$0D        ;  Is it CR?
D0D8     BNE    BAD1        ;  If not, set error flag
20C926   JSR    R2          ;  If yes, turn on motor 2
60       RTS                ;  Return
==A42F   GOOD3
C942     CMP    #$42        ;  Test second character for B
D0D0     BNE    BAD1        ;  If not B, set error flag
AD0210   LDA    $1002       ;  Get third character
C90D     CMP    #$0D        ;  Is it CR?
D0C9     BNE    BAD1        ;  If not, set error flag
20F126   JSR    RB          ;  If yes, turn on both motors
60       RTS                ;  Return
==A43E   SCHAR              ;  Test for, S1, or S2
AD0110   LDA    $1001       ;  Get second character
C931     CMP    #$31        ;  It is = 1?
D00B     BNE    GOOD4       ;  If not, continue
AD0210   LDA    $1002       ;  Get third character
C90D     CMP    #$0D        ;  It is CR?
D0B7     BNE    BAD1        ;  If not, set error flag
20D526   JSR    S1          ;  If yes, stop motor 1
==A44F
60       RTS                ;  Return
==A450   GOOD4
C932     CMP    #$32        ;  Is second character = 2?
D0AF     BNE    BAD1        ;  If not, set error flag
AD0210   LDA    $1002       ;  Get third character
C90D     CMP    #$0D        ;  Is it CR?
```

```
D0A8     BNE    BAD1        ;  If not, set error flag
20E326   JSR    S2          ;  If yes, stop motor 2
60       RTS                ;  Return
==A45F   CCHAR              ;  Test for CONT
AD0110   LDA    $1001       ;  Second character
C94F     CMP    #$4F        ;  Is it = O?
F003     BEQ    CHKN        ;  If yes, continue
4C03A4   JMP    BAD1        ;  If not, set error flag
==A469   CHKN
AD0210   LDA    $1002       ;  Get third character
C94E     CMP    #$4E        ;  It is = N?
F003     BEQ    CHKT        ;  If yes, continue
4C03AJ   JMP    BAD1        ;  If not, set error flat
==A473   CHKT
AD0310   LDA    $1002       ;  Get fourth character
C954     CMP    #$54        ;  Is it T?
F003     BEQ    CHKCR       ;  If yes, continue
4C03A4   JMP    BAD1        ;  If not, set error flag
==A47D   CHKCR
AD0410   LDA    $1004       ;  Get fifth character
C90D     CMP    #$0D        ;  Is it CR?
F003     BEQ    OUT         ;  If yes, continue
4C03A4   JMP    BAD1        ;  If not, set error flag
==A487   OUT
205327   JSR    CONT        ;  Process CONT subroutine
60       RTS                ;  Return
;  CHKSP                    ;  (Check for Space Bar)
==A48B   CHKSP
A000     LDY    #$00        ;  Clear character counter
==A48D   REPEAT
B90010   LDA    $1000,Y     ;  Get character
C8       INY                ;  Increment counter
C920     CMP    #$20        ;  Is character = SP?
F007     BEQ    JUMPSB      ;  If yes, jump
C003     CPY    #$03        ;  If no, keep checking first
D0F4     BNE    REPEAT      ;    three characters
A900     LDA    #$00        ;  Clear flag in accumulator
60       RTS                ;  Return
==A49C   JUMPSB
20AC26   JSR    SPBAR       ;  Turn off heaters and motors
A901     LDA    #$01        ;  Set SP flag in accumulator
60       RTS                ;  Return from subroutine
; FETCH4                    ;  (Get and Check Control Parameter Input)
==A4A2   FETCH4
A22D     LDX    #$2D        ;  Display underscore
8EC71E   STX    $1EC7
A000     LDY    #$00        ;  Initialize character counter
==A4A9   LOOP4
98       TYA
48       PHA                ;  Save character counter on stack
==A4AB   WAIT4
20E4FF   JSR    GETIN       ;  Get a character from keyboard que
```

```
C900     CMP   #$00
F0F9     BEQ   WAIT4
8D0A10   STA   TEMP        ;  Temporarily store character
68       PLA
A8       TAY               ;  Restore Character counter
AD0A10   LDA   TEMP        ;  Retrieve character
C90D     CMP   #$0D        ;  It is a CR?
==A4BC
D003     BNE   G04         ;  If not, continue
4CE9A4   JMP   CHKFR       ;  If so, check input for valid feedrate
==A4C1   G04
C003     CPY   #$03        ;  Check if 4 characters entered yet
D003     BNE   THIRD       ;  If not, continue
4CA2A4   JMP   FETCH4      ;  If yes, start over
==A4C8   THIRD
C002     CPY   #$02        ;  Third character entered yet?
D00C     BNE   SHOW        ;  If not, process first and second ones
8DB31E   STA   $1EB3       ;  If so, display character
A220     LDX   #$20        ;  Remove underscore
8EC91E   STX   $1EC9
C8       INY               ;  Increment character counter
4CA9A4   JMP   LOOP4       ;  Get fourth character
==A4D8   SHOW
99B11E   STA   $1EB1,Y     ;  save character in screen memory
A920     LDA   #$20        ;  clear underscore
99C71E   STA   $1EC7,Y
A92D     LDA   #$2D        ;  Display new underscore
99C81E   STA   $1EC8,Y     ;
C8       INY               ;  Increment character counter
4CA9A4   JMP   LOOP4       ;  Get next character
==A4E9   CHKFR             ;  Check for valid feedrate input
A000     LDY   #$00        ;  Initialize counter
==A4EB   NEXTFR
B9B11E   LDA   $1EB1,Y     ;  Get character
C930     CMP   #$30        ;  Is it > = 0?
B003     BCS   HIEND       ;  If so, continue
4C08A5   JMP   ERR3A       ;  If not, set error flag
==A4F5   HIEND
C93A     CMP   #$3A        ;  Is it > = 10?
9003     BCC   MORE1       ;  If not, continue
4C08A5   JMP   ERR3A       ;  If so, set error flag
==A4FC   MORE1
C8       INY
C003     CPY   #$03        ;  Third character checked yet?
D0EA     BNE   NEXTFR      ;  If not, continue checking
ADB11E   LDA   $1EB1       ;  Check first character for 1
C932     CMP   #$32
9011     BCC   FTCHSR      ;  If first digit = 1, continue
==A508   ERR3A
A920     LDA   #$20        ;  Clear underscore
8DC71E   STA   $1EC7
8DC81E   STA   $1EC8
```

```
8DC91E  STA   $1EC9
20D125  JSR   ERROR       ;  Display error message
4CA2A4  JMP   FETCH4      ;  Get new Feed rate input
==A519  FTCHSR            ;  Get stretch-ratio input
20EA25  JSR   CLRLNE
==A51C  FTCHA
A22D    LDX   #$2D        ;  Display underscore
8ED71E  STX   $1ED7
A000    LDY   #$00
==A523  LOOPSR
98      TYA
48      PHA
==A525  WAITSR
20E4FF  JSR   GETIN       ;  Get a character from keyboard que
C900    CMP   #$00
F0F9    BEQ   WAITSR
8D0A10  STA   TEMP
68      PLA
A8      TAY
AD0A10  LDA   TEMP
C90D    CMP   #$0D        ;  It is CR?
==A536
D003    BNE   GOSR        ;  If no, get another character
4C78A5  JMP   CHKSR       ;  If yes, check for valid input
==A53B  GOSR
C003    CPY   #$03
D003    BNE   THRDCH
4C1CA5  JMP   FTCHA       ;  Start again if 4 characters and no CR
==A542  THRDCH
C002    CPY   #$02
D00C    BNE   SCNDCH
8DC41E  STA   $1EC4       ;  Display third character
A220    LDX   #$20
8EDA1E  STX   $1EDA
C8      INY
4C23A5  JMP   LOOPSR      ;  Get another character
==A552  SCNDCH
C001    CPY   #$01
D011    BNE   FRSTCH
8DC31E  STA   $1EC3       ;  Display second character
A220    LDA   #$20
8ED91E  STX   $1ED9
A22D    LDX   #$2D
8EDA1E  STX   $1EDA
==A563
C8      INY
4C23A5  JMP   LOOPSR      ;  Get another character
==A567  FRSTCH
8DC11E  STA   $1EC1       ;  Display first character
A220    LDX   #$20
8ED71E  STX   $1ED7
A22D    LDX   #$2D
```

```
8ED91E  STX   $1ED9
C8      INY
4C23A5  JMP   LOOPSR      ;  Get another character
==A578  CHKSR             ;  Check for valid stretchratio input
ADC11E  LDA   $1EC1       ;  Check if first digit between 1 and 4
C931    CMP   #$31
B003    BCS   HISR
4DA4A5  JMP   ERR3B       ;  Display error if < 1
==A582  HISR
C934    CMP   #$34
9003    BCC   TENS
4CA4A5  JMP   ERR3B       ;  Display error if > 4
==A589  TENS              ;  Test if digits 2 and 3 between 0 and 9
A000    LDY   #$00
==A58B  NEXTSR
B9C31E  LDA   $1EC3,Y
C930    CMP   #$30
B003    BCS   HI
4CA4A5  JMP   ERR3B       ;  Display error if < 0
==A595  HI
C93A    CMP   #$3A
9003    BCC   HUNDS
4CA4A5  JMP   ERR3B       ;  Display error if > 9
==A59C  HUNDS
C8      INY
C002    CPY   #$02
F014    BEQ   TEMP1
4C8BA5  JMP   NEXTSR
==A5A4  ERR3B             ;  Clear underscores and display
A920    LDA   #$20        ;  error message if error occurred
8DD71E  STA   $1ED7
8DD91E  STA   $1ED9
8DDA1E  STA   $1EDA
20D125  JSR   ERROR
4C1CA5  JMP   FTCHA       ;  Get new stretch-ratio input if error
==A5B5  TEMP1
20EA25  JSR   CLRLNE      ;  Clear error message line
==A5B5  TEMP1A            ;  Get Temp. 1 setpoint from keyboard
A000    LDY   #$00
A92D    LDA   #$2D        ;  Display underscore
8D1F1F  STA   $1F1F
==A5BF  TMPCHR
98      TYA
48      PHA
==A5C1  WAITMP
20E4FF  JSR   GETIN       ;  Get a character from keyboard que
C900    CMP   #$00
F0F9    BEQ   WAITMP
8D0A10  STA   TEMP
68      PLA
A8      TAY
AD0A10  LDA   TEMP
```

```
C90D    CMP   #$0D        ;  Is it CR?
==A5D2
D003    BNE   GOTEMP      ;  If no, get another character
4CFFA5  JMP   CHKTMP      ;  If yes, check for valid input
==A5D7  GOTEMP
C003    CPY   #$03
D003    BNE   OTHERS      ;  Start again if 4 characters and no CR
4CB8A5  JMP   TEMP1A
==A5DE  OTHERS
C002    CPY   #$02        ;  Branch to display first and second
                             characters
D00C    BNE   SHWTMP
8D0B1F  STA   $1F0B       ;  Display third character
A220    LDX   #$20
8E211F  STX   #1F21
C8      INY
4CBFA5  JMP   TMPCHR      ;  Get fourth character
==A5EE  SHWTMP
99091F  STA   $1F09,Y     ;  Display first and second characters
A920    LDA   #$20
991F1F  STA   #1F1F,Y     ;  Clear underscore
A92D    LDA   #$2D
99201F  STA   $1F20,Y     ;  Display underscore
C8      INY
4CBFA5  JMP   TMPCHR      ;  Get next character
==A5FF  CHKTMP            ;  Check Temp.1 input for validity
A009    LDY   #$09
2081A6  JSR   TEMPS       ;  Call temperature check subroutine
C900    CMP   #$00
F011    BEQ   TEMP2       ;  If Temp.1 valid, continue with Temp.2
A920    LDA   #$20        ;  If no, clear underscores
8D1F1F  STA   $1F1F
8D201F  STA   $1F20
==A610
8D211F  STA   $1F21
20D125  JSR   ERROR       ;  Display error message
4CB8A5  JMP   TEMP1A      ;  Jump back to get Temp.1
==A619  TEMP2
20EA25  JRS   CLRLNE      ;  Clear error message
==A61C  TEMP2A            ;  Get Temp2 set point from keyboard
A000    LDY   #$00
A92D    LDA   #$2D        ;  Display underscore
8D2D1F  STA   $1F2D
==A623  LOOPT2
98      TYA
48      PHA
==A625  WAITT2
20E4FF  JSR   GETIN       ;  Get a character from keyboard que
C900    CMP   #$00
F0F9    BEQ   WAITT2
8D0A10  STA   TEMP
68      PLA
```

```
A8       TAY
AD0A10   LDA   TEMP
C90D     CMP   #$0D        ;  Is it CR?
==A636
D003     BNE   GOT2        ;  If no, get another character
4C63A6   JMP   CHKT2       ;  If yes, check for valid input
==A63B   GOT2
C003     CPY   #$03
D003     BNE   MORET2      ;  Start again if 4 characters and no CR
40C1CA6  JMP   TEMP2A
==A642   MORET2
C002     CPY   #$02        ;  Branch to display first and second
                              characters
D00C     BNE   SHOWT2
8D191F   STA   $1F19       ;  Display third character
A220     LDX   #$20
8E2F1F   STX   $1F2F
C8       INY
4C23A6   JMP   LOOPT2      ;  Get fourth character
==A652   SHOWT2
99171F   STA   $1F17,Y     ;  Display first and second characters
A920     LDA   #$20
992D1F   STA   $1F2D,Y     ;  Clear underscore
A92D     LDA   #$2D
992E1F   STA   $1F2E,Y     ;  Display underscore
C8       INY
4C23A6   JMP   LOOPT2      ;  Get next character
==A663   CHKT2             ;  Check Temp2 input for validity
A017     LDY   #$17
2081A6   JRS   TEMPS       ;  Call temperature check subroutine
C900     CMP   #$00
F011     BEQ   RETSET      ;  If Temp2 valid, continue
A920     LDA   #$20        ;  If not, clear underscores
8D2D1F   STA   $1F2D
8D2E1F   STA   $1F2E
==A674
8D2F1F   STA   $1F2F
20D125   JRS   ERROR       ;  Display error message
4C1CA6   JMP   TEMP2A      ;  Jump back to get new Temp2
==A67D   RETSET
20EA25   JSR   CLRLNE      ;  Clear error message
60       RTS               ;  Return from subroutine
;TEMPS                     ;  (Temperature setpoint check)
==A681   TEMPS
B9001F   LDA   $1F00,Y     ;  Get first digit
C930     CMP   #$30        ;  Check if first digit is 0 or 1
B003     BCS   UPPER
4CDBA6   JMP   SETERR      ;  set error flag if < 0
==A68B   UPPER
C932     CMP   #$32
9003     BCC   ORIT
4CDBA6   JMP   SETERR      ;  set error flag if > 1
```

```
==A692  ORIT
C930    CMP   #$30        ;  Is first digit = 0?
F024    BEQ   ZERO        ;  If so, check other digits for 0-9
CB      INY               ;  If not, check second digit
B9001F  LDA   $1F00,Y     ;  Is second digit between 0 and 5
C930    CMP   #$30
B003    BCS   OKTMP1
4CDBA6  JMP   SETERR      ;  Set error flag if < 0
==A6A1  OKTMP1
C936    CMP   #$36
9003    BCC   OKTMP2
4CDBA6  JMP   SETERR      ;  Set error flag if > 5
==A6A8  OKTMP2
C8      INY
B9001F  LDA   $1F00,Y     ;  Is third digit between 0 and 9?
C930    CMP   #$30
B003    BCS   OKTMP3
4CDBA6  JMP   SETERR      ;  Set error flag if < 0
==A6B3  OKTMP3
C93A    CMP   #$3A
9027    BCC   RETURN
4DCBA6  JMP   SETERR      ;  Set error flag if > 9, otherwise return
==A6BA  ZERO
C8      INY
B9001F  LDA   $1F00,Y     ;  First digit = 1.  Check if second digit is
C930    CMP   #$30        ;  between 0 and 9
B003    BCS   OKTMP4
4CDBA6  JMP   SETERR      ;  Set error flag if < 0
==A605  OKTMP4
C93A    CMP   #$3A
9003    BCC   OKTMP5
4CDA6   JMP   SETERR      ;  Set error flag if > 9
==A6CC  OKTMP5
C8      INY
B9001F  LDA   #1F00,Y     ;  Check if third digit is between 0 and 9
C930    CMP   #$30
B003    BCS   OKTMP6
4CDBA6  JMP   SETERR      ;  Set error flag if < 0
==A6D7  OKTMP6
C93A    CMP   #$3A
9003    BCC   RETURN      ;  Set error flag if > 9.  Otherwise return
==A6DB  SETERR
A901    LDA   #$01        ;  Set error flag
60      RTS               ;  Return
==A6DE  RETURN            ;  Return
20EA25  JSR   CLRLNE      ;  Clear error message line
A900    LDA   #$00        ;  Clear error flag to indicate no error
60      RTS               ;  Return
;CHECK3                   ;  (Check Heater Operation Code Input)
==A6E4  CHECK3
A900    LDA   #$00
8D1D10  STA   $101D       ;  Clear CONT and error flag
```

```
8D0B10   STA   ERRFLG
208BA4   JSR   CHKSP      ;  Check if SP key pressed
C900     CMP   #$00
D014     BNE   RTN3       ;  If SP hit, return
AD0010   LDA   $1000      ;  Get first character
==A6F6
C948     CMP   #$48       ;  If character = H, check for H1, H2, HB
F00E     BEQ   HCHAR
C94F     CMP   #$4F       ;  If character = 0, check for 01, 02
F03F     BEQ   OCHAR
C943     CMP   #$43       ;  If character = C, check for CONT
F05C     BEQ   CHARC
==A702   BAD3
A901     LDA   #$01       ;  If character none of above, set error flag
8D0B10   STA   ERRFLG
==A707   RTN3
60       RTS               ;  Return
==A708   HCHAR             ;  Check for H1, H2, HB
AD0110   LDA   $1001       ;  Get second character
C931     CMP   #$31        ;  Is character > = 1?
B003     BCS   VALID1      ;  If yes, continue
4C0A7    JMP   BAD3        ;  If not, set error flag
==A712   VALID1
D00B     BNE   VALID2      ;  If character not = 1, check for 2 or B
AD0210   LDA   $1002       ;  check third character for CR
C90D     CMP   #$0D
D0E7     BNE   BAD3        ;  If not CR,set error flag
20FA26   JSR   H1          ;  Turn on heater 1.  Code = H1
60       RTS               ;  Return
==A71F   VALID2
C932     CMP   #$32        ;  Is second character = 2?
D00B     BNE   VALID3      ;  If not, check for B
AD0210   LDA   $1002       ;  Check third character for CR
C90D     CMP   #$0D
D0D8     BNE   BAD3        ;  If no CR, set error flag
200827   JSR   H2          ;  Turn on heater 2.  Code = H2
60       RTS               ;  Return
==A72E   VALID3
C942     CMP   #$42        ;  Is second character = B?
D0D0     BNE   BAD3        ;  If not, set error flag
AD0210   LDA   $1002       ;  Check third character for CR
C90D     CMP   #$0D
D0C9     BNE   BAD3        ;  If not CR, set error flag
201627   JSR   HB          ;  Turn on both heaters
60       RTS               ;  Return
==A73D   OCHAR             ;  Check for 01 or 02
AD0110   LDA   $1001
C931     CMP   #$31        ;  Is second character = 1?
D00B     BNE   VALID4      ;  If not, check for a 2.
AD0210   LDA   $1002       ;  Check third character for CR
C90D     CMP   #$0D
D0B7     BNE   BAD3        ;  If not CR, set error flag
```

```
202D27  JSR   01          ;  Turn off heater 1
==A74E
60      RTS               ;  Return
==A74F  VALID4
C932    CMP   #$32        ;  Is second character = 2?
D0AF    BNE   BAD3        ;  If not, set error flag
AD0210  LDA   $1001       ;  Check for third character for CR
C90D    CMP   #$0D
D0A8    BNE   BAD3        ;  If not CR, set error flag
203327  JSR   02          ;  Turn off heater 2
60      RTS               ;  RETURN
==A75E  CHARC             ;  Check for CONT
AD0110 LDA    $1001
C94F    CMP   #$4F        ;  Continue if second character = 0
F003    BEQ   TSTN
4C02A7  JMP   BAD3        ;  Otherwise set error flag
==A768  TSTN
AD0210  LDA   $1002
C94E    CMP   #$4E        ;  Continue if third character = N
F003    BEQ   TSTT
4C02A7  JMP   BAD3        ;  Otherwise set error flag
==A772  TSTT
AD0310  LDA   $1003
C954    CMP   #$54        ;  Continue if fourth character = T
F003    BEQ   TSTCR
4C02A7  JMP   BAD3        ;  Otherwise set error flag
==A77C  TSTCR
AD0410  LDA   $1004
C90D    CMP   #$0D        ;  Continue if fifth character = CR
F003    BEQ   FINE
4C02A7  JMP   BAD2        ;  Otherwise set error flag
==A786  FINE
205327  JSR   CONT        ;  Set cont flag
60      RTS               ;  Return
==A78A
;HEXDEC                   ;  (Convert Hex No. to Decimal No.)
==A78A  HEXDEC
A900    LDA   #$00        ;  Clear locations to contain decimal
8D3610  STA   $1036       ;     Answer
8D3510  STA   $1035
AD3410  LDA   1034        ;  Get hex number
290F    AND   #$0F        ;  Mask high nibble
F00E    BEQ   UNDER       ;  If 1's digit = 0 , branch to convert 16's
                             digit
AA      TAX               ;  Transfer nibble for counter
==A79A
18      CLC               ;  Clear carry
F8      SED               ;  Set decimal mode
==A79C  RPT1
AD3610  LDA   $1036       ;  Get low order decimal
6901    ADC   #$01        ;  Add 1
8D3610  STA   $1036       ;  Save
```

```
CA       DEX              ;  Decrement X until 1 has been
D0F5     BNE   RPT1       ;      added low order nibble times
==A7A7   UNDER
AD3410   LDA   $1034      ;  Get hex number
4A       LSR   A          ;  Shift right 4 times
4A       LSR   A
4A       LSR   A
4A       LSR   A
F015     BEQ   FINISH     ;  If 16's digit= 0, end
AA       TAX              ;  Transfer nibble for counter
18       CLC              ;  Clear carry
==A7B2   RPT2
AD3610   LDA   $1036      ;  Get low order decimal
6916     ADC   #$16       ;  Add 16
8D3610   STA   $1036      ;  Save
A900     LDA   #$00       ;  Prepare to add carry, if any
6D3510   ADC   $1035
8D3510   STA   $1035
==A7C2
CA       DEX              ;  Decrement counter
D0ED     BNE   RPT2       ;  Repeat adding until done
==A7C5   FINISH
D8       CLD              ;  Clear decimal mode
60       RTS              ;  Return
;M8BY8                    ;  (Eight-bit Multiplication)
==A7C7   M8BY8
AD2510   LDA   $1025      ;  Load multiplier
AA       TAX
A900     LDA   #$00       ;  Clear product
8D2310   STA   $1023
8D2410   STA   $1024
==A7D3   ML1
18       CLC
AD2410   LDA   $1024      ;  Load product LSB
6D2610   ADC   $1026      ;  Add multiplicand
8D2410   STA   $1024      ;  Store product LSB
AD2310   LDA   $1023      ;  Add carry to product MSB
6900     ADC   #$00
8D2310   STA   $1023      ;  Store product MSB
==A7E5
CA       DEX              ;  Decrement multiplier
D0EB     BNE ML1          ;  Branch if multiplier non zero
60       RTS              ;  Return
         *=$2000
;CHECK4                  ;  (Check for RUN, STOP,QUIT, CHANGE)
==2000   CHECK4
A900     LDA #$00         ;  Clear error flag
8D0B10   STA   ERRFLG
208BA4   JSR   CHKSP      ;  Was space bar (sp) pressed?
C900     CMP   #$00
D01B     BNE   OUT4       ;  If yes, return
AD0010   LDA   $1000      ;  If no, check first character
```

```
C952    CMP   #$52        ;  Is character = R?
==2011
F015    BEQ   CHKR        ;  If yes, check for RUN
C953    CMP   #$53        ;  Is character = S?
F03D    BEQ   CHKS        ;  If yes, check for STOP
C951    CMP   #$51        ;  Is character =Q?
F06F    BEQ   CHKQ        ;  If yes, check for QUIT
C943    CMP   #$43        ;  Is character = C?
D003    BNE   BAD4        ;  If no, set error flag
4CC020  JMP   CHKC        ;  If yes, check for CHANGE
==2022  BAD4
A901    LDA   #$01        ;  Set error flag in memory
8D0B10  STA   ERRFLG
==2027  OUT4
60      RTS               ;  Return
==2028  CHKR              ;  Check for RUN
AD0110  LDA   $1001
C955    CMP   #$55        ;  If second character = U, continue
F003    BEQ   TRYN
4C2220  JMP   BAD4        ;  Otherwise set error flag
==2032  TRYN
AD0210  LDA   $1002
C94E    CMP   #$4E        ;  If third character = N,continue
F003    BEQ   TRYCR
4C2220  JMP   BAD4        ;  Otherwise set error flag
==203C  TRYCR
AD0310  LDA   $1003
C90D    CMP   #$0D        ;  IF FOURTH CHARACTER = CR, CONTINUE
F003    BEQ   SUBRUN
4C2220  JMP   BAD4        ;  Otherwise set error flag
==2046  SUBRUN
203927  JSR   RUN         ;  Run both moters
A9C7    LDA   #RUNL       ;  Set jump address
8D1510  STA   $1015
A9A1    LDA   #RUNH
8D1610  STA   $1016
60      RTS               ;  Return
==2054  CHKS              ;  Check for STOP
AD0110  LDA   $1001
C954    CMP   #$54        ;  If second character = T, continue
F003    BEQ   OH
402220  JMP BAD4          ;  If not, set error flag
==205E  OH
AD0210  LDA   $1002
C94F    CMP   #$4F        ;  If third character = O, continue
F003    BEQ   PEE
4C2220  JMP   BAD4        ;  If not, set error flag
==2068  PEE
AD0310  LDA   $1003
C950    CMP   #$50        ;  If fourth character = P, continue
F003    BEQ   CAR
4C2220  JMP   BAD4        ;  If not, set error flag
```

```
==2072  CAR
AD0410  LDA   $1004
C90D    CMP   #$0D        ;  If fifth character = CR, continue
F003    BEQ   SUBST
4C2220  JMP   BAD4        ;  If not, set error flag
==207C  SUBST
204527  JSR   STOP        ;  Stop motors
A994    LDA   #STOPL      ;  Set jump address
8D1510  STA   $1015
A9A1    LDA   #STOPH
8D1610  STA   $1016
60      RTS               ;  Return
==208A  CHKQ              ;  Check for QUIT
AD0110  LDA   $1001
C955    CMP   #$55        ;  If second character = U, continue
F003    BEQ   EYE
4C2220  JMP   BAD4        ;  If not, set error flag
==2094  EYE
AD0210  LDA   $1002
C949    CMP   #$49        ;  If third character = I, continue
F003    BEQ   TEA
4C2220  JMP   BAD4        ;  If not, set error flag
==209E  TEA
AD0310  LDA   $1003
C954    CMP   #$54        ;  If fourth character = T, continue
F003    BEQ   CGRT
4C2220  JMP   BAD4        ;  If not, set error flag
==20A8  CGRT
AD0410  LDA   $1004
C90D    CMP   #$0D        ;  If fifth character = CR, continue
F003    BEQ   SUBQUT
402220  JMP   BAD4        ;  If not, set error flag
==20B2  SUBQUT
205C27  JSR   QUIT        ;  Terminate process
A900    LDA   #QUITL      ;  Set jump address
8D1510  STA   $1015
A9A0    LDA   #QUITH
8D1610  STA   $1016
60      RTS               ;  Return
==20C0  CHKC              ;  Check for CHANGE
AD0110  LDA   $1001
C948    CMP   #$48        ;  If second character = H, continue
F003    BEQ   AY
4C2220  JMP   BAD4        ;  If not, set error flag
==20CA  AY
AD0210  LDA   $1002
C941    CMP   #$41        ;  If third character = A, continue
F003    BEQ   EN
4C2220  JMP   BAD4        ;  If not, set error flag
==20D4  EN
AD0310  LDA   $1003
C94E    CMP   #$4E        ;  If fourth character = N, continue
```

```
 F003     BEQ    GEE
 4C2220   JMP    BAD4        ;  If not, set error flag
 ==20DE   GEE
 AD0410   LDA    $1004
 C947     CMP    #$47        ;  If fifth character = G, continue
 F003     BEQ    EE
 4C2220   JMP    BAD4        ;  If not, set error flag
 ==20E8   EE
 AD0510   LDA    $1005
 C945     CMP    #$45        ;  If sixth character = E, continue
 F003     BEQ    SUBCHG
 4C2220   JMP    BAD4        ;  If not, set error flag
 ==20F2   SUBCHG
 204527   JSR    STOP        ;  Stop motors
 A91A     LDA    #CHNGEL     ;  Set jump address
 8D1510   STA    $1015
 A9A1     LDA    #CHNGEH
 8D1610   STA    $1016
 60       RTS                ;  Return
 ;FEED
 ==2100   FEED               ;  (Convert Feed rate/Stretch-ratio to Motor
                                Set points)
 A000     LDY    #$00        ;  Get feedrate setpoint ready to
 ==2102   FRCHAR             ;  put in hex form
 B9B11E   LDA    $1EB1,Y
 990010   STA    $1000,Y
 C8       INY
 C003     CPY    #$03
 D0F5     BNE    FRCHAR
 20A822   JSR    DECHEX      ;  Put feed rate in 1 byte hex form
 8D2210   STA    $1022       ;  Prepare to calculate motor 2 set point
 ==2113                      ;  Store converted feed rate as divisor
*A911     LDA    #$11        ;  Dividend, MBS:  Controller Constant
 8D2010   STA    $1020
*A9FA     LDA    #$FA        ;  Dividend, LSB
 8D2110   STA    $1021
 AD2210   LDA    $1022       ;  Shift dividend and divisor until
 2980     AND    #$80        ;  divisor most sig. bit = 0
 F00A     BEQ    DIVFR
 ==2124
 4E2210   LSR    $1022
 4E2010   LSR    $1020
 6E2110   ROR    $1021
 18       CLC
 ==212E   DIVFR
 20F122   JSR    D16BY8      ;  Divide Controller Constant by Feedrate
 AD1410   LDA    $1014       ;  Get drive wheel radius
 8D2510   STA    $1025       ;  Save as multiplicand
 AD2110   LDA    $1021       ;  Get quotient
 8D2610   STA    $1026       ;  Save as multiplier
 20C7A7   JSR    M8BY8       ;  Multiply quotient by radius
 ==2140
```

```
AD2410  LDA   $1024      ;  Get product
A8      TAY              ;  Save product in Y
AD2010  LDA   $1020      ;  Get remainder from division
8D2610  STA   $1026
20C7A7  JSR   M8BY8      ;  Multiply remainder by radius
AD2310  LDA   $1023      ;  Prepare to divide controller constant
==2150
8D2010  STA   $1020      ;  by product
AD2410  LDA   $1042
8D2110  STA   $1021
20F122  JSR   D16BY8     ;  Divide controller constant by product
98      TYA
18      CLC
6D2110  ADC   $1021      ;  Add original product + remainder product
==2161                   ;  for more accurate set point
8D019B  STA   I03PA      ;  Send set point to motor 2
A8      TAY
A900    LDA   #$00       ;  Get stretch-ratio set point ready
8D0010  STA   $1000      ;  to put in hex form
ADC11E  LDA   $1EC1
8D0110  STA   $1001
ADC31E  LDA   $1EC3
==2173
8D0210  STA   $1002
20A822  JSR   DECHEX     ;  Put stretch-ratio in hex form
8D2510  STA   $1025      ;  Prepare to calculate motor 1 set point
8C2610  STY   $1026
20C7A7  JSR   M8BY8      ;  Multiply motor 2 set point by stretch-ratio
AD2310  LDA   $1023      ;  Product is motor 1 set point X 10
==2185
8D2010  STA   $1020      ;  Prepare to divide by 10
AD2410  LDA   $1024
8D2110  STA   $1021
A90A    LDA   #$0A
8D2210  STA   $1022
20F122  JSR   D16BY8     ;  Divide by 10 to get set point
==2196
AD2110  LDA   $1021
8D019A  STA   I02PA      ;  Send set point to motor 1
60      RTS              ;  Return
;HEAT                   ;  (Convert Temperatures to Hex Values)
==219D  HEAT
AD091F  LDA   $1F09      ;  Prepare to convert temp. 1 set point
8D0010  STA   $1000      ;  to hex
AD0A1F  LDA   $1F0A
8D0110  STA   $1001
AD0B1F  LDA   $1F0B
8D0210  STA   $1002
==21AF
20A822  JSR   DECHEX     ;  Convert temp. 1 to hex
8D1A10  STA   $101A      ;  Save
AD171F  LDA   $1F17      ;  Prepare to convert temp. 2 set point
```

```
8D0010  STA   $1000      ;  to hex
AD181F  LDA   $1F18
8D0110  STA   $1001
==21C1
AD191F  LDA   $1F19
8D0210  STA   $1002
20A822  JSR   DECHEX     ;  Convert temp. 2 to hex
8D1B10  STA   $101B      ;  Save
60      RTS              ;  Return
;CNTHTR                  ;  (Control Heaters)
==21CE  CNTHTR
A900    LDA   #$00       ;  Clear alarm flag
8D2810  STA   ALRM
AD1C10  LDA   $101C      ;  Get probe/motor switch status
4903    EOR   #$03       ;  Turn on temp. probe 1
8D1C10  STA   $101C
8D0098  STA   I01PB      ;  Send to output port
==21DE
A000    LDY   #$00       ;  set screen position for temp. 1 display
205926  JSR   TNEW       ;  Update temp. 1 from A/D
20A822  JSR   DECHEX     ;  Convert it to hex
8D0A10  STA   $100A
AD1A10  LDA   $101A
CD0A10  CMP   $100A      ;  Compare measured to desired temp.
==21EF
101C    BPL   FIND1      ;  If measured < desired, continue
18      CLC              ;  Otherwise check for over heating
690A    ADC   #$0A
CD0A10  CMP   $100A
300B    BMI   TOOHOT     ;  If measured > desired + 10, ALARM
A9FF    LDA   #$FF       ;  If measured < desired +10,
8D049A  STA   I02T1L     ;  Set control signal to smallest
8D059A  STA   I02T1L     ;  duty cycle.
==2201
4C3D22  JMP   T2         ;  Continue
==2204  TOOHOT           ;  Overheating accurred.  Set alarm
20F725  JSR   ALARM
A901    LDA   #$01       ;  Set alarm flag for calling program
8D2810  STA   ALRM
60      RTS              ;  Return-alarm occurred
==220D  FIND1            ;  Determine temp. 1 error
AD1A10  LDA   $101A      ;  Error = desired-measured
38      SEC
ED0A10  SBC   $100A
8D1810  STA   $1018      ;  Store error value
F024    BEQ   T2         ;  If same as last time, use
C90A    CMP   #$0A       ;  same control signal
1081    BPL   FULLON     ;  If error > 10 C,use 100% dirty cycle
==221D
8D0A10  STA   $100A      ;  Multiply error x 2 for table
18      CLC              ;  look up for duty cycle since
6D0A10  ADC   $100A      ;  Error < 10 C
```

```
A8       TAY
B9E346   LDA   TABLE, Y   ;  Get timer 1 LSB
8D049A   STA   IO2T1L     ;  Send to timer
C8       INY
B9E346   LDA   TABLE, Y   ;  Get timer 1 MSB
==222F
8D059A   STA   IO2T1H     ;  Send to timer
4C3D22   JMP   T2         ;  Jump to process Temp. 2
==2235   FULLON           ;  Set 100% duty cycle for heater 1
A901     LDA   #$01
8D049A   STA   IO2T1L
8D059A   STA   IO2T1H
==223D   T2
AD1C10   LDA   $101C      ;  Get probe/motor switch status
4903     EOR   #$03       ;  Turn on temp. probe 2
8D1C210  STA   $101C
8D0098   STA   IO1PB      ;  Send to output port
A00E     LDY   #$0E       ;  Set screen position for temp. 2 display
205926   JSR   TNEW       ;  Update temp. 2 from A/D
==224D
20A822   JSR   DECHEX     ;  Convert it to hex
8D0A10   STA   $100A
AD1B10   LDA   $101B
CD0A10   CMP   $100A      ;  Compare measured to desired temp.
101C     BPL   FIND2      ;  If measured < desired, continue
18       CLC              ;  Otherwise check for overheating
690A     ADC   #$0A
==225E
CD0A10   CMP   $100A
300B     BMI   TOOHI      ;  If measured > desired + 10, ALARM
A9FF     LDA   #$FF       ;  If measured < + 10
8D049B   STA   IO3T1L     ;  Set control signal to smallest
8D059B   STA   IO3T1H     ;  duty cycle
4CA722   JMP   COVER      ;  Continue
==226E   TOOHI            ;  Over heating occurred.  Set alarm
20F725   JSR   ALARM
A901     LDA   #$01       ;  Set alarm for calling program
8D2810   STA   ALRM
60       RTS              ;  Return-alarm occurred
==2277   FIND2            ;  Determine temp.2 error
AD1B10   LDA   $101B      ;  Error = desired-measured
38       SEC
ED0A10   SBC   $100A
8D1910   STA   $1019      ;  Store error value
F024     BEQ   COVER      ;  If same as last time, use
C90A     CMP   #$0A       ;  same control signal
1018     BPL   ALLOUT     ;  If error > 10 C, use 100% duty cycle
==2287
8D0A10   STA   $100A      ;  Multiply error x 2 for table
18       CLC              ;  look up of duty cycle.  Error < 10 C
6D0A10   ADC   $100A
A8       TAY
```

```
B9E346  LDA   TABLE,Y     ;  Get timer 1 LSB
8D049B  STA   IO3T1L      ;  Send to timer
C8      INY
B9E346  LDA   TABLE,Y     ;  Get timer 1 MSB
==2299
8D059B  STA   IO3T1H      ;  Send to timer
4CA722  JMP   COVER       ;  Jump to return
==229F  ALLOUT            ;  Set 100% duty cycle for heater 2
A901    LDA   #$01
8D049B  STA   IO3T1L
8D059B  STA   IO3T1H
==22A7  COVER
60      RTS               ;  Return
;DECHEX
==22A8  DECHEX            ;  (Convert Decimal No. to Hex)
A900    LDA   #$00        ;  Clear temporary storage location
8D1310  STA   #1013
D8      CLD               ;  Clear decimal mode
AD0010  LDA   $1000       ;  Get MSB of 3 digit decimal no.
C931    CMP   #$31        ;  Does hundreds digit = 1?
D005    BNE   ADD         ;  If not, convert tens and ones
A264    LDX   #$64        ;  If so, add 100 to result
8E1310  STX   HEX
==22BA  ADD
AD0210  LDA   $1002       ;  Get ones digit
290F    AND   #$OF        ;  Mask off high nibble
8D1210  STA   $1012       ;  Save
AD0110  LDA   $1001       ;  Get tens digit
290F    AND   #$OF        ;  Mask off high nibble
18      CLC               ;  Clear carry
2A      ROL   A           ;  Shift tens digit into place
2A      ROL   A
==22CA
2A      ROL   A
2A      ROL   A
0D1210  ORA   $1012       ;  "OR" tens and ones
8D1210  STA   $1012       ;  Save result
AA      TAX
290F    AND   #$OF        ;  Mask high order nibble
18      CLC
6D1310  ADC   $1013       ;  Add ones to running sum
8D1310  STA   $1013
==22DC
8A      TXA               ;  Get accumulator back
29F0    AND   #$FO        ;  Mask low order nibble
4A      LSR   A           ;  Divide by 2
8D1210  STA   $1012       ;  Save
4A      LSR   A
4A      LSR   A
18      CLC               ;  Divide by 8
6D1210  ADC   $1012       ;  Add divide by 2 + divide by 8
6D1310  ADC   $1013
```

```
==22EC
8D1310  STA   $1013       ;  Save result
60      RTS               ;  Return
==22F0
 ;D16BY8                  ;  (16 bit by 8 bit division)
==22F1  D16BY8
A208    LDX   #$08        ;  Set bit count to 8
==22F3  DL1
0E2110  ASL   $1021       ;  Shift quotient left
2E2010  ROL   $1020       ;  Shift divident left
AD2010  LDA   $1020       ;  Load dividend
38      SEC
ED2210  SBC   $1022       ;  Subtract divisor
9006    BCC   DL2
8D2010  STA   $1020       ;  Store new dividend
==2305
EE2110  INC   $1021       ;  Increment quotient
==2308  DL2
CA      DEX               ;  Decrement bit count
D0E8    BNE   DL1         ;  Branch if bit count non zero
60      RTS               ;  Return
 ;SCRN1                   ;  (Title Screen)
==230C  SCRN1
A9F2    LDA   #$F2        ;  Specify character set 2
8D0590  STA   $9005
A9BE    LDA   #$BE        ;  Set screen/border colors as
8D0F90  STA   $900F       ;  light cyan/blue
A287    LDX   #$87        ;  Blank screen with blue
8E0190  STX   $9001
A200    LDX   #$00        ;  Clear screen with spaces
==231D
A000    DLY   #$00        ;  and specify black character color
==231F  CLEAR1
A920    LDA   #$20
99001E  STA   $1E00,Y
8A      TXA
990096  STA   $9600,Y
C8      INY
C0FC    CPY   #$FC
D0F2    BNE   CLEAR1
A000    LDY   #$00
==232F  CLEAR2            ;  Finish clearing screen
A920    LDA   #$20
99FC1E  STA   $1EFC,Y
8A      TXA
99FC96  STA   $96FC,Y
C8      INY
C0FC    CPY   #$FC
D0F2    BNE   CLEAR2
A000    LDY   #$00
==233F  STAR1             ;  Put in purple border stars
A92A    LDA   #$2A        ;  Top row stars first
```

```
99841E  STA   $1E84,Y
A904    LDA   #$04
998496  STA   $9684,Y
C8      INY
C016    CPY   #$16
DOF1    BNE   STAR1
A000    LDY   #$00
==2350  STAR2             ;  Put in bottom row stars
A92A    LDA   #$2A
994A1F  STA   $1F4A,Y
A904    LDA   #$04
994A97  STA   $974A,Y
C8      INY
C016    CPY   #$16
DOF1    BNE   STAR2
A000    LDY   #$00
==2361  STAR3             ;  Put in left column stars
A92A    LDA   #$2A
999A1E  STA   $1E9A,Y
A904    LDA   #$04
999A96  STA   $969A,Y
98      TYA
18      CLC
6916    ADC   #$16
A8      TAY
COBO    CPY   #$BO
==2372
DOED    BNE   STAR3
A000    LDY   #$00
==2376  STAR4             ;  Put in right column stars
A92A    LDA   #$2A
99AF1E  STA   $1EAF,Y
A904    LDA   #$04
99AF96  STA   $96AF,Y
98      TYA
18      CLC
6916    ADC   #$16
A8      TAY
COBO    CPY   #$VO
==2387
DOED    BNE   STAR4
A900    LDA   #M1L        ;  Specify address of message 1 in memory
8562    LDA   $0062
A942    LDA   $M1H
8563    STA   $0063
A914    LDA   #$14        ;  Specify number of characters in message
8D0B10  STA   $100B
A9C7    LDA   #$C7        ;  Specify screen location of message 1
==2398
8564    STA   $0064
A91E    LDA   #$1E
8565    STA   $0065
```

```
204926  JSR   DISPLY      ;  Display message 1
A914    LDA   #M2L        ;  Specify address of message 2 in memory
8562    STA   $0062
A942    LDA   $M2H
8563    STA   $0063
==23A9
A912    LDA   #$12        ;  Specify number of characters in message
8D0B10  STA   $100B
A90A    LDA   #$0A        ;  Specify screen location of message 2
8564    STA   $0064
A91F    LDA   #$1F
8565    STA   $0065
204926  JSR   DISPLY      ;  Display message 2
==23B9
A087    LDY   #$87
==23BB  LOADX
A2F0    LDX   #$F0
==23BD  SCROLL            ;  Scroll screen up on CRT
8C0190  STY   $9001
CA      DEX
D0FA    BNE   SCROLL
88      DEY
C019    CPY   #$19
D0F3    BNE   LOADX
60      RTS               ;  Return
 ;SCRN2                   ;  (24-hour clock screen)
==23C9  SCRN2
A920    LDA   #$20        ;  Clear screen with spaces
A000    LDY   #$00
==23CD  CLRIT
99841E  STA   $1E84,Y
C8      INY
C0FF    CPY   #$FF
D0F8    BNE   CLRIT
A900    LDA   #M1L        ;  Specify address of message 1 in memory
8562    STA   $0062
A942    LDA   #M1H
8563    STA   $0063
==23DD
A914    LDA   #$14        ;  Specify number of characters in message
8D0B10  STA   $100B
A917    LDA   #$17        ;  Specify screen location of message 1
8564    STA   $0064       ;  Specify screen location of message 2
A91E    LDA   #$1E
8565    STA   $0065
204926  JSR   DISPLY      ;  Display message 1
==23ED
A914    LDA   #M2L        ;  Specify address of message 2 in memory
8562    STA   $0062
A942    LDA   #M2H
8563    STA   $0063
A912    LDA   #$12        ;  Specify number of characters in message
```

```
8D0B10   STA   $100B
A92E     LDA   #$2E        ;  Specify screen location of message
8564     STA   $0064
==23FE
A91E     LDA   #$1E
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 2
A926     LDA   #M3L        ;  Specify address of message 3 in memory
8562     STA   $0062
A942     LDA   #M3H
8563     STA   $0063
A979     LDA   #$79        ;  Specify number of characters in message
==240F

8D0B10   STA   $100B
A99F     LDA   #$9F        ;  Specify screen location of message3
8564     STA   $0064
A91E     LDA   #$1E
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 3
A92D     LDA   #$2D        ;  Print line on screen
==241F
A000     LDY   #$00
==2421   LINE1
99581E   STA   $1E58,Y
C8       INY
C016     CPY   #$16
D0F8     BNE   LINE1
60       RTS               ;  Return
 ;SCRN2A                   ;  (Drive Wheel Diameter Screen)
==242A   SCRN2A
A9A4     LDA   #M2AL       ;  Specify address of message 2A in memory
8562     STA   $0062
A946     LDA   #M2AH
8563     STA   $0063
A93F     LDA   #$3F        ;  Specify number of characters in message
8D0B10   STA   $100B
A961     LDA   #$61        ;  Specify screen location of message 2A
8564     STA   $0064
==243B
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 2A
60       RTS               ;  Return
 ;SCRN3                    ;  (Process Preparation Screen)
==2443 SCRN3
A9B5     LDA   #M5L        ;  Specify address of message 5 in memory
8562     STA   $0062
A942     LDA   #M5H
8563     STA   $0063
A913     LDA   #$13        ;  Specify number of characters in message
8D0B10   STA   $100B
```

```
A99C    LDA   #$9C       ;  Specify screen location of message 5
8564    STA   $0064
==2454
A91E    LDA   #$1E
8565    STA   $0065
204926  JSR   DISPLY     ;  Display message 5
A920    LDA   #$20       ;  Clear 1 line on screen
A000    LDY   #$00
==245F  CLRLN
99C61E  STA   $1EC6,Y
C8      INY
C016    CPY   #$16
D0F8    BNE   CLRLN
A9C8    LDA   #M6L       ;  Specify address of message 6 in memory
8562    STA   $0062
A942    LDA   #M6H
8563    STA   $0063
==246F
A940    LDA   #$40       ;  Specify number of characters in message
8D0B10  STA   $100B
A9DD    LDA   #$DD       ;  Specify screen location of message 6
8564    STA   $0064
A91E    LDA   #$1E
8565    STA   $0065
204926  JSR   DISPLY     ;  Display message 6
==247F
A908    LDA   #M7L       ;  Specify address of message 7 in memory
8562    STA   $0062
A943    LDA   #M7H
8563    STA   $0063
A984    LDA   #$84       ;  Specify number of characters in message
8D0B10  STA   $100B
A976    LDA   #$76       ;  Specify screen location of message 7
8564    STA   $0064
==2490
A91F    LDA   #$1F
8565    STA   $0065
204926  JSR   DISPLY     ;  Display message 7
A92D    LDA   #$2D       ;  Print separating line on screen
A000    LDY   #$00
==249B  LINE2
99601F  STA   $1F60,Y
C8      INY
C016    CPY   #$16
D0F8    BNE   LINE2
60      RTS              ;  Return
 ;SCRN4                  ;  (Control Parameters Screen)
==24A4  SCRN4
A98C    LDA   #M8L       ;  Specify address of message 8 in memory
8562    STA   $0062
A943    LDA   #M8H
8563    STA   $0063
```

```
A982     LDA   #$82        ;  Specify number of characters in message
8D0B10   STA   $100B
A99A     LDA   #$9A        ;  Specify screen location of message 8
8564     STA   $0064
==24B5
A91E     LDA   #$1E
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 8
A90E     LDA   #M9L        ;  Specify address of message 9 in memory
8562     STA   $0062
A944     LDA   #M9H
8563     STA   $0063
A97F     LDA   #$7F        ;  Specify number of characters in message
==24C6
8D0B10   STA   $100B
A976     LDA   #$76        ;  Specify screen location of message 9
8564     STA   $0064
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 9
A98D     LDA   #M10L       ;  Specify address of message 10 in memory
==24D6
8562     STA   $0062
A944     LDA   #M10H
8563     STA   #0063
A912     LDA   #$12        ;  Specify number of characters in message
8D0B10   STA   $100B
A970     LDA   #$70        ;  Specify screen location of message 10
8564     STA   $0064
A91E     LDA   #$1E
==24E7
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 10
60       RTS               ;  Return
 ;SCRN5                    ;  (Heater Operation Instruct Screen)
==24ED   SCRN5
A99F     LDA   #M11L       ;  Specify address of message 11 in memory
8562     STA   $0062
A944     LDA   #M11H
8563     STA   $0063
A939     LDA   #$39        ;  Specify number of characters in message
8D0B10   STA   $100B
A976     LDA   #$76        ;  Specify screen location of message 11
8564     STA   $0064
==24FE
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 11
A920     LDA   #$20        ;  Clear lower part of screen
A8       TAY
==2508   CLR3
998F1F   STA   $1F8F,Y
```

```
C8      INY
C06B    CPY   #$6B
C0F8    BNE   CLR3
60      RTS             ;  Return
 ;SCRN6                 ;  (Heater Operation Code Screen)
==2511  SCRN6
A9D8    LDA   #M12L     ;  Specify address of message 12 in memory
8562    STA   $0062
A944    LDA   #M12H
8563    STA   $0063
A984    LDA   #$84      ;  Specify number of character in message
8D0B10  STA   $100B
A976    LDA   #$76      ;  Specify screen location of message 12
8564    STA   $0064
==2522
A91F    LDA   #$1F
8565    STA   $0065
204926  JSR   DISPLY    ;  Display message 12
A928    LDA   #$28      ;  Print (des), (act) on screen
8D101F  STA   #1F10
8D261F  STA   $1F26
A904    LDA   #$04      ;  d
==2533
8D111F  STA   $1F11
A905    LDA   #$05      ;  e
8D121F  STA   $1F12
A913    LDA   #$13      ;  s
8D131F  STA   $1F13
A929    LDA   #$29      ;  ,
8D141F  STA   $1F14
==2545
8D2A1F  STA   $1F2A
A901    LDA   #$01      ;  a
8D271F  STA   $1F27
A903    LDA   #$03      ;  c
8D281F  STA   $1F28
A914    LDA   #$14      ;  t
8D291F  STA   $1F29
==2557
60      RTS             ;  Return
 ;SCRN7                 ;  (Waiting for Heaters Screen)
==2558  SCRN7
A95C    LDA   #M14L     ;  Specify address of message 14 in memory
8562    LDA   $0062
A945    LDA   #M14H
8563    STA   $0063
A958    LDA   #$58      ;  Specify number of characters in message
8D0B10  STA   $100B
A976    LDA   #$76      ;  Specify screen location of message 14
8564    STA   $0064
==2569
A91F    LDA   #$1F
```

```
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 14
60       RTS               ;  Return
 ;SCRN8                    ;  (Process Ready Screen)
==2571   SCRN8
A9B4     LDA   #M15L       ;  Specify address of message 15 in memory
8562     STA   $0062
A945     LDA   #M15H
8563     STA   $0063
A96E     LDA   #$6E        ;  Specify number of characters in message
8D0B10   STA   $100B
A976     LDA   #$76        ;  Specify screen location of message 15
8564     STA   $0064
==2582
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 15
60       RTS               ;  Return
 ;SCRN9                    ;  (Process Running Screen)
==258A   SCRN9
A922     LDA   #M16L       ;  Specify address of message 16 in memory
8562     STA   $0062
A946     LDA   #M16H
8563     STA   $0063
A96E     LDA   #$6E        ;  Specify number of characters in message
8D0B10   STA   $100B
A976     LDA   #$76        ;  Specify screen location of message
8564     STA   $0064
==259B
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY      ;  Display message 16
A928     LDA   #$28        ;  Print (des),(act) on screen
8DB91E   STA   $1EB9
8DCF1E   STA   $1ECF
A904     LDA   #$04        ;  d
==25AC
8DBA1E   STA   $1EBA
A905     LDA   #$05        ;  e
8DBB1E   STA   #1EBB
A913     LDA   #$13        ;  s
8DBC1E   STA   #1EBC
A929     LDA   #$29        ;  )
==25BE
8DD31E   STA   $1ED3
A901     LDA   #$01        ;  a
8DD01E   STA   $1ED0
A903     LDA   #$03        ;  c
8DD11E   STA   $1ED1
A914     LDA   #$14        ;  t
8DD21E   STA   $1ED2
==25D0
```

```
60       RTS              ;  Return
;ERROR                    ;  (Error Message Display)
==25D1   ERROR
A99F     LDA   #M4L       ;  Specify address of message 4 in memory
8562     STA   $0062
A942     LDA   #M4H
8563     STA   $0063
A916     LDA   #$16       ;  Specify number of characters in message
8D0B10   STA   $100B
A934     LDA   #$34       ;  Specify screen location of message 4
8564     STA   $0064
==25E2
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY     ;  Display error message
60       RTS              ;  Return
;CLRLNE                   ;  (Clear Error Message Line)
==25EA   CLRLNE
A920     LDA   #$20       ;  Print 1 line of spaces
A000     LDY   #$00
==25EE BLAND
99341F   STA   $1F34,Y
C8       INY
C016     CPY   #$16
D0F8     BNE   BLAND
60       RTS              ;  Return
;ALARM                    ;  (Overheat Alarm)
==25F7   ALARM
20AC26   JSR   SPBAR      ;  Turn off motors and heaters
A990     LDA   #M17L      ;  Specify address of message 17 in memory
8562     STA   $0062
A946     LDA   #M17H
8563     STA   $0063
A914     LDA   #$14       ;  Specify number of characters in message
8D0B10   STA   $100B
==2607
A935     LDA   #$35       ;  Specify screen location of message
8564     STA   $0064
A91F     LDA   #$1F
8565     STA   $0065
204926   JSR   DISPLY     ;  Display message 17
A92A     LDA   #$2A       ;  Make screen red
8D0F90   STA   $900F
==2617
A90F     LDA   #$0F       ;  Set highest volume in VIC20 speakers
8D0E90   STA   $900E
A900     LDA   #$00       ;  Set note-speaker 1
8D0A90   STA   #900A
A9FB     LDA   #$FB       ;  Set note-speaker 2
8D0B90   STA   $900B
A9D7     LDA   #$D7       ;  Set note-speaker 3
==2628
```

```
8D0C90  STA   #900C
==262B  SOUND
20E4FF  JSR   GETIN      ;  Keep sounding alarm until key is pressed
C900    CMP   #$00
F0F9    BEQ   SOUND
A9BE    LDA   #$BE       ;  Restore screen/border color
8D0F90  STA   $900F
A900    LDA   #$00       ;  Turn off speakers
8D0A90  STA   $900A
==263C
8D0B90  STA   $900B
8D0C90  STA   $900C
8D0E90  STA   $900E
20EA25  JSR   CLRLNE     ;  Clear alarm message
60      RTS              ;  Return
;DISPLY                  ;  (Display a Message on Screen)
==2649  DISPLY
48      PHA
A000    LDY   #$00       ;  Clear character counter
==264C  MOREC
B162    LDA ($62),Y      ;  Get a character
9164    STA ($64),Y      ;  Display it
C8      INY
CC0B10  CPY   $100B      ;  Finished with message?
D0F6    BNE   MOREC      ;  If not, display more
68      PLA
60      RTS              ;  If so, Return
==2658
;TNEW                    ;  (Update Temps. on Screen)
==2659  TNEW
A980    LDA   #$80       ;  Initialize test byte
8D1710  STA   DIGSEL
AD0098  LDA   I01PB      ;  Clear EOC flag from CBI
==2661  NEOC
AD0D98  LDA   I01FR      ;  Check for EOC
2910    AND   #$10
F0F9    BEQ   NEOC       ;  Loop until EOC flag set
==2668  DS1
AD0198  LDA   I01PA      ;  Load Digit Select
2C1710  BIT   DIGSEL     ;  Compare with test byte
F0F8    BEQ   DS1        ;  Loop until first digit
2908    AND   #$08       ;  Isolate first digit
4908    EOR   #$08
4A      LSR   A          ;  Get digit in place
4A      LSR   A
4A      LSR   A
0930    ORA   #$30       ;  Make it ASCII
==2679
8D0010  STA   $1000      ;  Save in memory
991F1F  STA   $1F1F,Y    ;  Display first digit on screen
C8      INY
4E1710  LSR DIGSEL       ;  Shift test byte
```

```
==2683  DS2
AD0198  LDA   I01PA      ;  Load digit select byte
2C1710  BIT   DIGSEL     ;  Compare with test byte
F0F8    BEQ   DS2        ;  Loop untip second digit
290F    AND   #$0F       ;  Remove digit select bits
0930    ORA   #$30       ;  Make digit ASCII
8D0110  STA   #1001      ;  Save in memory
991F1F  STA   $1F1F,Y    ;  Display second digit on screen
==2695
C8      INY
4E1710  LSR   DIGSEL     ;  Shift test byte
==2699  DS3
AD0198  LDA   I01PA      ;  Load digit select byte
2C1710  BIT   DIGSEL     ;  Compare with test byte
F0F8    BEQ   DS3        ;  Loop until third digit
290F    AND   #$0F       ;  Remove digit select bits
0930    ORA   #$30       ;  Make it ASCII
8D0210  STA   $1002      ;  Display third digit on screen
991F1F  STA   $1F1F,Y
==26AB
60      RTS              ;  Return
;SPBAR                   ;  (Emergency Stop Motors/Heaters)
==26AC  SPBAR
A90D    LDA   #$0D       ;  Turn off motors
8D0098  STA   I01PB
8D1C10  STA   $101C
A920    LDA   #$20       ;  Turn off heaters
8D0B9A  STA   I02ACR
8D0B9B  STA   I03ACR
==26BC
60      RTS              ;  Return
;R1                      ;  (Run motor 1)
==26BD  R1
AD1C10  LDA   $101C      ;  Turn on motor 1.  Leave other switches
290B    AND   #$0B       ;  alone
8D0098  STA   I01PB
8D1C10  STA   $101C
60      RTS              ;  Return
;R2                      ; (Run motor 2)
==26C9  R2
AD1C10  LDA   $101C      ;  Turn on motor 2.  Leave other switches
2907    AND   #$07       ;    alone.
8D0098  STA   I01PB
8D1C10  STA   $101C
60      RTS              ;  Return
;S1                      ;  (Stop motor 1)
==26D5  S1
AD1C10  LDA   $101C      ;  Turn off motor 1.  Leave other switches
290B    AND   #$0B       ;    alone
0904    ORA   #$04
8D0098  STA   I01PB
8D1C10  STA   $101C
```

```
60       RTS              ;  Return
;S2                       ;  (Stop motor 2)
==26E3   S2
AD1C10   LDA   $101C      ;  Turn off motor 2.  Leave other switches
2907     AND   #$07       ;    alone
0908     ORA   #$08
8D0098   STA   I01PB
8D1C10   STA   $101C
60       RTS              ;  Return
 ;RB                      ;  (Run Both Motors)
==26F1   RB
A901     LDA   #$01       ;  Turn on both motors
8D0098   STA   I01PB
8D1C10   STA   $101C
60       RTS              ;  Return
 ;H1                      ;  (Turn on Heater 1)
==26FA   H1
A9E0     LDA   #$E0       ;  Enable Timer 1
8D0B9A   STA   I02ACR
A901     LDA   #$01       ;  Select 100% duty cycle
8D049A   STA   I02T1L
8D059A   STA   I02T1H
60       RTS              ;  Return
 ;H1                      ;  (Turn on Heater 2)
==2708   H2
A9E0     LDA   #$R0       ;  Enable Timer 1
8D0B9B   STA   I03ACR
A901     LDA   #$01       ;  Seclect 100% duty cycle
8D049B   STA   I03T1L
8D059B   STA   I03T1H
60       RTS              ;  Return
 ;HB                      ;  (Turn on Both Heaters)
==2716   HB
A9E0     LDA   #$E0       ;  Enable Timer 1 in both VIAs
8D0B9A   STA   I02ACR
8D0B9B   STA   I03ACR
A901     LDA   #$01       ;  Select 100% duty cycle
8D049A   STA   I02T1L
8D059A   STA   I02T1H
==2726
8D049B   STA   I03T1L
8D059B   STA   I03T1H
60       RTS              ;  Return
 ;01                      ;  (Turn off Heater 1)
==272D   01
A920     LDA   #$20       ;  Disable Timer 1
8D0B9A   STA   I02ACR
60       RTS              ;  Return
 ;02                      ;  (Turn off Heater 2)
==2733   02
A920     LDA   #$20       ;  (Disable Timer 1)
8D0B9B   STA   I03ACR
```

```
60       RTS              ;  Return
 ;RUN                     ;  (Run Process Motors)
==2739   RUN
AD1C10   LDA   $101C
2903     AND   #$03       ;  Turn on both motors
8D0098   STA   I01PB
8D1C10   STA   $101C
60       RTS              ;  Return
 ;STOP                    ;  (Stop both motors)
==2745   STOP
AD1C10   LDA   $101C
2903     AND   #$03       ;  Turn off both motors
090C     ORA   #$0C
8D0098   STA   I01PB
8C1C10   STA   $101C
60       RTS              ;  Return
 ;CONT                    ;  (CONTINUE process)
==2753   CONT
204527   JSR   STOP       ;  Turn off both motors
A901     LDA   #$01       ;  Set cont flag
8D1D10   STA   $101D
60       RTS              ;  Return
 ;QUIT                    ;  (Terminate Process)
==275C   QUIT
A90C     LDA   #$0C
8D0098   STA   I01PB      ;  Turn off both motors
A920     LDA   #$20       ;  Disable Timer 1 on both VIAs
8D0B9A   STA   I02ACR
8D0B9B   STA   I03ACR
A987     LDA   #$87       ;  Blank Screen with blue
8D0190   STA   $9001
==276E   OVER
4C6E17   JMP   OVER       ;  Enter endless loop
 ;COLOR                   ;  (Designate Screen Character Colors)
==2771   COLOR
A906     LDA   #$06
A000     LDY   #$00
==2775   BLUE1            ;  Make first 4 lines blue characters
990096   STA   $9600,Y
C8       INY
C058     CPY   #$58
D0F8     BNE   BLUE1
A900     LDA   #$00
A8       TAY
==2780   BLACK1           ;  Make fifth line black characters
995896   STA   $9658,Y
C8       INY
C016     CPY   #$16
D0F8     BNE   BLACK1
A906     LDA   #$06
A000     LDY   #$00
==278C   BLUE2            ;  Make line 6 blue characters
```

```
996E96  STA   $966E,Y
C8      INY
C016    CPY   #$16
D0F8    BNE   BLUE2
A900    LDA   #$00
A000    LDY   #$00
==2798  BLACK2          ;  Make lines 7-14 black characters
998496  STA   $9684,Y
C8      INY
C0B0    CPY   #$B0
D0F8    BNE   BLACK2
A902    LDA   #$02
A000    LDY   #$00
==27A4  RED1            ;  Make line 15 red characters
993497  STA   $9734,Y
C8      INY
C016    CPY   #$16
D0F8    BNE   RED1
A906    LDA   #$06
A000    LDY   #$00
==27B0  BLUE3           ;  Make line 16 blue characters
994A97  STA   $974A,Y
C8      INY
C016    CPY   #$16
D0F8    BNE   BLUE3
A900    LDA   #$00
A000    LDY   #$00
==27BC  BLACK3          ;  Make lines 17-23 black characters
996097  STA   $9760,Y
C8      INY
C09A    CPY   #$9A
D0F8    BNE   BLACK3
60      RTS             ;  Return
        *=$4000
==4000
 ;CLOCK                 ;  (Clock and FR/SR Update Routine)
==4000  CLOCK
A900    LDA   #$00
AA      TAX
CD0D10  CMP   FLAG      ;  Is clock update flag enabled?
D003    BNE   CHECKC    ;  If yes, process clock update routine
4CBFEA  JMP   $EABF     ;  If not, continue with VIC 20 interrupt
==400B  CHECKC
A5A2    LDA   $A2
CD0C10  CMP   LAST      ;  Has VIC 20 jiffy counter changed?
D003    BNE   COUNT     ;  If so, update clock
4CBFEA  JMP   $EABF     ;  If not, continue with VIC 20 interrupt
==4015  COUNT
8D0C10  STA   LAST      ;  Save VIC 20 jiffy counter
EE1110  INC   CNTR2     ;  Check if 5 sec. has elapsed
AD1110  LDA   CNTR2     ;  since last clock update
C996    CMP   #$96
```

```
F003    BEQ   INCR1       ;  If not, continue with VIC 20 interrupt
4CBFEA  JMP   $EABF
==4025  INCR1
8E1110  STX   CNTR2       ;  Clear 5 sec. counter, LSB
EE1010  INC   CNTR1
AD1010  LDA   CNTR1
C902    CMP   #$02
F003    BEQ   UPDATE
4CBFEA  JMP   $EABF
==4035  UPDATE
8E1010  STX   CNTR1       ;  Clear 5 sec counter, MSB
18      CLC
A901    LDA   #$01
A8      TAY
A905    LDA   #$05        ;  Get SEC2 clock increment
6D591F  ADC   SEC2
8D591F  STA   SEC2        ;  Update SEC2 on screen
A93A    LDA   #$3A
==4046
CD591F  CMP   SEC2        ;  If 10 sec. elasped, increment SEC2
D06F    BNE   JUMPT       ;  If not, jump out of clock update
A930    LDA   #$30
AA      TAX
8D591F  STA   SEC2        ;  Clear SEC2 digit
98      TYA
18      CLC
6D581F  ADC   SEC1        ;  Increment SEC1 digit
==4056
8D581F  STA   SEC1        ;  Display SEC1 on screen
A936    LDA   #$36
CD581F  CMP   SEC1        ;  Check if SEC1 = 6
D05A    BNE   JUMP1       ;  If not, jump out of clock update
8E581F  STX   SEC1        ;  If so, clear SEC1
98      TYA
18      CLC
6D561F  ADC   MIN2        ;  and increment MIN1 digit
==4068
8D561F  STA   MIN2        ;  Display MIN2 on screen
A93A    LDA   #$3A
CD561F  CMP   MIN2        ;  Check if MIN1 = 10
D048    BNE   JMPT        ;  If not, jump out of clock update
8E561F  STX   MIN2        ;  If so, clear MIN2
98      TYA
18      CLC
6D551F  ADC   MIN1        ;  and increment MIN1 digit
==407A
8D551F  STA   MIN1        ;  Display MIN1 on screen
A936    LDA   #$36
CD551F  CMP   MIN1        ;  Check if MIN1 = 6
D036    BNE   JUMPT       ;  If not, jump out of clock update
8E551F  STX   MIN1        ;  If so, clear MIN1
98      TYA
```

```
 18       CLC
 6D531F   ADC   HR2          ;  and increment HR2 digit
 ==408C
 8D531F   STA   HR2          ;  Display HR2 on screen
 A932     LDA   #$32
 CD521F   CMP   HR1          ;  Check if HR1 = 2
 F015     BEQ   TWO          ;  If so, branch to check HR2
 A93A     LDA   #$3A         ;  If not, check if HR2 = 10
 CD531F   CMP   HR2
 D01D     BNE   JUMPT        ;  If HR2 < 10, jump out of clock update
 ==409D
 8E531F   STX   HR2          ;  If HR2 = 10, clear HR2
 98       TYA
 18       CLC
 6D521F   ADC   HR1          ;  and increment HR1
 8D521F   STA   HR1          ;  Display HR1 on screen
 4CBFEA   JMP   $EABF        ;  Continue with VIC2 ϕ interrupt
 ==40AB   TWO                ;  HR1 = 2.  Check HR2
 A935     LDA   #$35
 CD531F   CMP   HR2          ;  Is HR2 > 4?
 D008     BNE   JUMPT        ;  If not, jump out of clock update
 8E521F   STX   HR1          ;  If so, clear HR1
 A931     LDA   #$31         ;  Make HR1 = 0
 8D531F   STA   HR2          ;  Make HR2 = 1
 ==40BA   JUMPT              ;  Feed rate/stretch-ratio (FR/SR) update
 A900     LDA   #$00
 CD2710   CMP   $1027        ;  Is FR/SR update flag enabled?
 D003     BNE   UPFS         ;  If so, continue with update
 4CBFEA   JMP   $EABF        ;  If not, continue with VIC 20 interrupt
 ==40C4   UPFS
 AD099B   LDA   $9B09        ;  Get MSB of motor 2 counter
 8D3210   STA   $1032
 AD089B   LDA   $9B08        ;  Get LSB of motor 2 counter
 8D3310   STA   $1033
 38       SEC                ;  Clear borrow
 A9FF     LDA   #$FF
 8D089B   STA   $9B08        ;  Re load VIA Timer 2 counter
 ==40D6
 8D099B   STA   $9B09
 ED3210   SBC   $1032        ;  Get actual motor 2 feedback
 8D3210   STA   $1032        ;  count and save as dividend
 8D2010   STA   $1020
 A9FF     LDA   #$FF
 ED3310   SBC   $1033
 ==40E7
 8D3310   STA   $1033
 9D2110   STA   $1021
*A966     LDA   #$66        ;  Load divisor constant
 8D2210   STA   $1022
 20F122   JSR   D16BY8      ;  Divide count by divisor constant
 AD2110   LDA   $1021       ;  Prepare to multiply result
 ==40F8
```

```
8D2510   STA   $1025      ;  by drive radius to get FR
AD1410   LDA   $1014
8D2610   STA   $1026
20C7A7   JSR   M8BY8      ;  Multiply quotient by drive radius
AD2410   LDA   $1024
8D3410   STA   $1034      ;  Prepare to convert FR to decimal
==410A
208AA7   JSR   HEXDEC     ;  Convert FR to decimal
D8       CLD
AD3510   LDA   $1035      ;  Get hundreds digit of FR
0930     ORA   #$30       ;  Make it ASCII
8DC71E   STA   $1EC7      ;  and display it
AD3610   LDA   $1036      ;  Get tens digit of FR
29F0     AND   #$F0       ;  mask low nibble
==411B
4A       LSR   A          ;  Shift tens digit into place
4A       LSR   A
4A       LSR   A
4A       LSR   A
0930     ORA   #$30       ;  Make it ASCII
8DC81E   STA   $1EC8      ;  and display it
AD3610   LDA   $1036      ;  Get ones digit
290F     AND   #$0F       ;  Mask high nibble
0930     ORA   #$30       ;  Make it ASCII
==412B
8DC91E   STA   $1EC9      ;  and display it
AD099A   LDA   $9A09      ;  Get MSB of Motor 1 counter
8D3010   STA   $1030
AD089A   LDA   $9A08      ;  Get LSB of Motor 1 counter
8D3110   STA   $1031
38       SEC              ;  Clear borrow
==413B
A9FF     LDA   #$FF
8D089A   STA   $9A08      ;  Reload VIA Timer 2 counter
8D099A   STA   $9A09
ED3010   SBC   $1030      ;  Get actual Motor 1 feedback
8D3010   STA   $1030      '  count and save as dividend
A9FF     LDA   #$FF
==414B
ED3110   SBC   $1031
8D3110   STA   $1031
18       CLC
==4152   NULL
A900     LDA   #$00
CD3010   CMP   $1030      ;  Check if MSB of count = 0
F010     BEQ   DIVID      ;  If so, branch
4E3010   LSR   $1030      ;  If not, shift Motor 1 and
6E3110   ROR   $1031      ;  Motor 2 count right until
4E3210   LSR   $1032      ;  Motor 1 MSB = 0
==4162
6E3310   ROR   $1033
18       CLC
```

```
405241  JMP   NULL
==4169  DIVID             ;  If most sig. bit of motor 1 LSB
18      CLC               ;  count = 1, Shift Motor 1 and
AD3110  LDA   $1031       ;  Motor 2 count right
2980    AND   #$80
F00D    BEQ   DIV10       ;  If most sig. bit = 0, continue
4E3010  LSR   $1030
6E3110  ROR   $1031
4E3210  LSR   $1032
==417A
6E3310  ROR   $1033
18      CLC
==417E  DIV10             ;  Prepare to form Motor2/Motor1
AD3310  LDA   $1033       ;  speed ratio to get SR
8D2110  STA   $1021
AD3210  LDA   $1032
8D2010  STA   $1020
A000    LDY   #$00
==418C  SHIFT1            ;  Multiply Motor 2 count by 10
0E2110  ASL   $1021       ;  to get significance to tenths
2E2010  ROL   $1020       ;  digit in answer
C8      INY
C003    CPY   #$03        ;  To multiply by 10, add Motor2 x 2
D0F5    BNE   SHIFT1      ;  Plus Motor2 x 8
18      CLC
0E3310  ASL   $1033
2E3210  ROL   $1032
==419E
18      CLC
AD3310  LDA   $1032       ;  Put Motor 2 count as dividend
6D2110  ADC   $1020
8D2110  STA   $1020
AD3210  LDA   $1033
6D2010  ADC   $1021
==41AE
8D2010  STA   $1021
AD3110  LDA   $1031       ;  Put Motor 1 count as divisor
8D2210  STA   $1022
20F122  JSR   D16BY8      ;  Form (SR x 10)
AD2110  LDA   $1021
8D3410  STA   $1034
==41C0
208AA7  JSR   HEXDEC      ;  Convert (SR x 10) to decimal
D8      CLD
AD3610  LDA   $1036       ;  Get tens digit and mask
29F0    AND   #$F0        ;  off low order nibble
4A      LSR   A           ;  Shift tens digit into place
4A      LSR   A
4A      LSR   A
4A      LSR   A
0930    ORA   #$30        ;  Makt it ASCII and display
8DD71E  STA   $1ED7       ;  it on screen in 1's digit
```

```
==41D2
AD3610   LDA    $1036       ;  Get ones digit and mask off
290F     AND    #$0F        ;  high nibble
0930     ORA    #$30        ;  Make it ASCII and display
8DD91E   STA    $1ED9       ;  it on screen in tenth digit
A930     LDA    #$30
8DDA1E   STA    $1EDA       ;  Display 0 in hundredths digit
A92E     LDA    #$2E
==41E3
8DD81E   STA    $1ED8       ;  Display color
4CBFEA   JMP    $EABF       ;  Process VIC 20 interrupts
==41E9
         *=$4200
==4200   M1                  ;  Message 1
4D49     DBYTE $4D49,
$4352
4352
4F50     DBYTE $4F50,
$4F52,$4F55,$5320,$4
D45,$4D42,$5241,$4E4
5
4F52
4F55
5320
4D45
4D42
==4210
5241
4E45
==4214   M2                  ;  Message 2
5052     DBYTE $5052,
$4F43
4F43
4553     DBYTE $4553,
$5320,$434F,$4E54,$5
24F,$4C4C,$4552
5320
434F
4E54
524F
4C4C
==4224
4552
==4226   M3                  ;  Message 3
3234     DBYTE $3224,
$2D48
2D48
5220     DBYTE $5220,
$434C,$4F43,$4B20,$2
020,$2020,$202
0,$2020
434C
```

```
4F43
4B20
2020
2020
==4236
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
2020
2020
2020
2020
==4246
2020
2020
2020
2020
450E    DBYTE $450E,
$1405,$1220,$5409,$0
D05,$3A28,$1818,$3A1
8,$183A
1405
1220
5409
==4256
0D05
3A28
1818
3A18
183A
1818    DBYTE $1818,
$2920,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
2920
2020
==4266
2020
2020
2020
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$2020,$5
012,$0513,$1320,$305
2,$4554
2020
```

```
==4276
2020
2020
5012
0513
1320
3C52
4554
5552    DBYTE $5552,
$4E3E,$2001,$0614,$0512,$2020,$2005,$0E1
4,$0512
==4286
4E3E
2001
0614
0512
2020
2005
0E14
0512
==4296
090E    DBYTE $090E,
$0720,$0401,$1401,
0720
0401
1401
2E      BYTE  $2E
==429F  M4              ;  Message 4
2020    Dbyte $2020,
$2020
2020
494E    DBYTE $494E,
$5641,$4C49,$4420,$4
54E,$5452,$592E,$202
0,$2020
5641
4C49
4420
454E
5452
==42AF
592E
2020
2020
==42B5  M5              ;  Message 5
5012    DBYTE $5012,
$0F03
0F03
0513    DBYTE $0513,
$1320,$5012,$0510,$0
112,$0114,$090F
1320
```

```
5012
0510
0112
0114
==42C5
090F
0E      BYTE  $0E
==42C8  M6                ;  Message 6
450E    DYBTE $450E,
$1405
1405
1220    DBYTE $1220,
$010E,$200F,$1005,$1
201,$1409,$0F0E,$202
0,$2003
010E
200F
1005
1201
1409
==42D8
0F0E
2020
2003
0F04    DBYTE $0F04,
$0520,$0612,$0F0D,$2
002,$050C,$0F17,$2C2
0,$2020
0520
0612
0F0D
2002
==42E8
050C
0F17
2C20
2020
2020    DBYTE $2020,
$2006,$0F0C,$0C0F,$1
705,$0420,$0219,$203
C,$5245
2006
0F0C
0C0F
==42F8
1705
0420
0219
203C
5245
5455    DBYTE $5455,
$524E,$3E2E
```

```
524E
3E2E
==4308  M7                  ;  Message 7
5231    DBYTE $5231,
$2F53
2F53
313D    DBYTE $313D,
$5215,$0E2F,$5314,$0
F10,$204D,$0F14,$0F1
2,$2031
5215
0E2F
5314
0F10
204D
==4318
0F14
0F12
2031
5232    DBYTE $5232,
$2F53,$323D,$5215,$0
E2F,$5314,$0F10,$204
D,$0F14
2F53
323D
5215
0E2F
==4328
5314
0F10
204D
0F14
0F12    DBYTE $0F12,
$2032,$5242,$3D52,$1
50E,$2002,$0F14,$082
0,$0D0F
2032
5242
3D52
==4338
150E
2002
0F14
0820
0D0F
140F    DBYTE $140F,
$1213,$2020,$2020,$4
34F,$4E54,$3D43,$0F0
E,$1409
1213
2020
==4348
```

```
2020
434F
4E54
3D43
0F0E
1409
0E15    DBYTE $0E15,
$0520,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
0520
==4358
2020
2020
2020
2020
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$2020,$2
020,$2020,$2020,$202
0,$3C53
==4368
2020
2020
2020
2020
2020
2020
2020
3C53
==4378
1001    DBYTE $1001
$0305,$2042,$0112,$3
E3D,$450D,$0703,$192
0,$5354
0305
2042
0112
3E3D
450D
0703
1920
==4388
5354
4F50    DBYTE $4F50,
==438C  M8              ;  Message 8
4645    DBYTE $4645,
$4544
4544
5241    DBYTE $5241,
```

```
$5445,$2020,$5354,$5
245,$5443,$4852,$415
4,$494F
5445
2020
5354
5245
5443
==439C
4852
4154
494F
2018    DBYTE $2018,
$1818,$090E,$2F0D,$0
E20,$2020,$2020,$203
1,$3A18
1818
090E
2F0D
0E20
==43AC
2020
2020
2031
3A18
2E18    DBYTE $2E18,
$1820,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
1820
2020
2020
==43BC
2020
2020
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
2020
2020
==43CC
2020
2020
2020
2020
2020
2020
2020    DBYTE $2020,
```

```
$2020,$2020,$2020,$2
020,$2020,$5445,$4D5
0,$2031
2020
==43DC
2020
2020
2020
2020
5445
4D50
2031
2020    DBYTE $2020,
$2020,$2020,$2020,$5
445,$4D50,$2032,$202
0,$2018
==43EC
2020
2020
2020
5445
4D50
2032
2020
2018
==43FC
1818    DBYTE $1818,
$2043,$2020,$2020,$2
020,$2020,$2018,$181
8,$2043
2043
2020
2020
2020
2020
2020
1818
==440C
2043
==440E  M9                ;  Message 9
450E    DBYTE $450E,
$1405
1405
1220    DBYTE $1220,
$0405,$1309,$1205,$0
420,$1305,$142D,$202
0,$2020
0405
1309
1205
0420
1305
```

```
==441E
142D
2020
2020
2010    DBYTE $2010,
$0F09,$0E14,$2014,$0
805,$0E20,$203C,$524
5,$5455
0F09
0E14
2014
0805
==442E
0E20
203C
5245
5455
524E    DBYTE $524E,
$3E2E,$4645,$4544,$5
241,$5445,$2830,$303
0,$2D31
3E2E
4645
4544
==443E
5241
5445
2830
3030
2D31
3939    DBYTE $3939,
$2909,$0E2F,$0D0E,$5
354,$5245,$5443,$485
2,$4154
2909
0E2F
==444E
0D0E
5354
5245
5443
4852
4154
494F    DBYTE $494F,
$2031,$2E30,$302D,$3
34E,$3030,$5445,$4D5
0,$3120
2031
==445E
2E30
302D
332E
```

```
3030
5445
4D50
3120
2830    DBYTE $2830,
$3030,$2D31,$3539,$2
920,$4320,$2020,$202
0,$5445
==446E
3030
2D31
3539
2920
4320
2020
2020
5445
==447E
4D50    DBYTE $4D50,
$3220,$2830,$3030,$2
D31,$3539,$2920
3220
2830
3030
2D31
3539
2920
43      BYTE  $43
==448D  M10             ;  Message 10
430F    DBYTE $430F,
$0E14
0E14
120F    DBYTE $120F,
$0C20,$5001,$1201,$0
D05,$1405,$1213
0C20
5001
1201
0D05
1405
==449D
1213
==449F  M11             ;  Message 11
450E    DBYTE $450E,
$1405
1405
1220    DBYTE $1220,
$0110,$1012,$0F10,$1
209,$0114,$0520,$202
0,$2020
0110
0F10
```

```
1209
0114
==44AF
0520
2020
2020
0805    DBYTE $0805,
$0114,$0512,$200F,$1
005,$1201,$1409,$0F0
E,$2003
0014
0512
200F
1005
==44BF
1201
1409
0F0E
2003
0F04    DBYTE $0F04,
$052C,$1408,$050E,$2
03C,$5245,$5455,$524
E
052C
1408
050E
==44CF
203C
5245
5455
524E
3E      BYTE  $3E
==44D8  M12               ;  Message 12
4831    DBYTE $4831,
$2F4F
2F4F
313D    DBYTE $313D,
$4805,$0114,$0512,$3
120,$4F4E,$2F4F,$464
6,$2020
4805
0114
0512
3120
4F4E
==44E8
2F4F
4646
2020
4832    DBYTE $4832,
$2F4F,$323D,$4805,$0
114,$0512,$3220,$4F4
```

```
E,$2F4F
2F4F
323D
4805
0114
==44F8
0512
3220
4F4E
2F4F
4646    DBYTE $4646,
$2020,$4842,$3D42,$0
F14,$0820,$0805,$011
4,$0512
2020
4842
3D42
==4508
0F14
0820
0805
0114
0512
1320    DBYTE $1320,
$4F4E,$2020,$2020,$4
34F,$4E54,$3D43,$0F0
E,$1409
4F4E
2020
==4518
2020
434F
4E54
3D43
0F0E
1409
0E15    DBYTE $0E15,
$0520,$2020,$2020,$2
020,$2020,$2020,$202
0,$2020
0520
==4528
2020
2020
2020
2020
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$2020,$2
020,$2020
```

```
==4538
2020
2020
2020
2020
2020
2020    DBYTE $2020,
$2020,$3C53,$1001,$0
305,$2002,$0112,$3E3
D,$450D
2020
3C53
==4548
1001
0305
2002
0112
3E3D
450D
0703    DBYTE $0703,
$1920,$5354,$4F50
1920
==4558
5354
4F50
==455C  M14                ;  Message14
2057    DBYTE #2057,
$4149
4149
5449    DBYTE $5449,
$4E47,$2046,$4F52,$2
048,$4541,$5445,$525
3,$2020
4E47
2046
4F52
2048
4541
==456C
5445
5253
2020
2020    DBTYE $2020,
$2020,$2020,$2020,$2
020,$2020,$2020,$202
0
2020
2020
2020
2020
==457C
2020
```

```
2020
2020
2020    DBYTE $2020,
$2020,2020,$4348,$4
14E,$4745,$3D43,$080
1,$0E07
2020
2020
4348
414E
==458C
4745
3D43
0801
0E07
0520    DBYTE $0520,
$5001,$1201,$0D14,$1
213,$5155,$4954,$3D5
4,$0512
5001
1201
0D14
==459C
12113
5155
4954
3D54
0512
0D09    DBYTE $0D09,
$0E01,$1405,$2050,$1
20F,$0305,$1313
0E01
1405
==45AC
2050
120F
0305
1313
==45B4  M15             ;  Message 15
2050
$524F
524F
4345    DBYTE $4345,
$5353,$2052,$4541,$4
459,$2054,$4F20,$525
5,$4E20
5353
2052
4541
4459
2054
==45C4
```

```
4F20
5255
4E20
2020    DBYTE $2020,
$2020,$2020,$2020,$2
020,$2020,$2020,$202
02020
2020
2020
2020
==45D4
2020
2020
2020
2020    DBYTE $2020,
$2020,$2020,$5255,$4
E3D,$5215,$0E20,$101
2,$0F03
2020
2020
5255
4E3D
==45E4
5215
0E20
1012
0F03
0513    DBYTE $0513,
$1320,$0D0F,$140F,$1
213,$5155,$4954,$3D5
4,$0512
1320
0D0F
140F
==45F4
1213
5155
4954
3D54
0512
0D09    DBYTE $0D09,
$0E01,$1405,$2010,$1
20F,$0305,$1313,$434
3,$414E
0E01
1405
==4604
2010
120F
0305
1313
4348
```

```
414E
4745    DBYTE $4745,
$3D43,$0801,$0E07,$0
520,$1001,$1201,$0D1
4,$1213
3D43
==4614
0801
0E07
0520
1001
1201
0D14
1213
==4622  M16             ;  Message 16
2020    DBYTE $2020,
$2050
2050
524F    DBYTE $524F,
$4345,$5353,$2052,$5
54E,$4E49,$4E47,$303
0,$202
4345
5353
2052
554E
4E49
==4632
4E47
2020
2020
4348    DBYTE $4348,
$414E,$4745,$3D43,$0
801,$0E07,$0520,$100
1,$1201
414E
4745
3D43
0801
==4642
0E07
0520
1001
1201
0D14    DBYTE $0D14,
$1213,$5354,$4F50,$3
D53,$140F,$1020,$4D0
F,$140F
1213
5354
4F50
==4652
```

```
 3D53
 140F
 1020
 4D0F
 140F
 1213    DBYTE $1213,
 $2020,$2020,$2020,$5
 155,$4954,$3D54,$051
 2,$0D09
 2020
 2020
 ==4662
 2020
 5155
 4954
 3D54
 0512
 0D09
 0E01    DBYTE $0E01,
 $1405,$2010,$120F,$0
 305,$1313,$2020,$202
 0,$2020
 1405
 ==4672
 2010
 120F
 0305
 1313
 2020
 2020
 2020
 2020    DBYTE $2020,
 $2020$2020,$2020,$2
 020,$2020,$2020,$202
 0
 ==4682
 2020
 2020
 2020
 2020
 2020
 2020
 2020
 ==4690  M17             ;  Message 17
 4F56    DBYTE $4F56,
 $4552
 4552
 4845    DBYTE $4845,
 $4154,$494E,$4720,$4
 F43,$4355,$5252,$454
 4
 4154
```

```
494E
4720
4F43
4355
==46A0
5252
4544
==46A4  M2A                 ;  Message 2A
450E    DBYTE $450E,
$1405
1405
1220    DBYTE $1220,
$0409,$010D,$0514,$0
512,$200F,$0620,$202
0,$2020
0409
010D
0514
0512
200F
==46B4
0620
2020
2020
200F    DBYTE $200F,
$1514,$1015,$1420,$0
412,$0916,$0520,$170
8,$0505
1514
1015
1420
0412
==46C4
0916
0520
1708
0505
0C      BYTE  $0C
2020    DBYTE $2C20,
$2020,$030C,$0F13,$0
513,$1420,$090E,$030
8,$2C30
2020
030C
0F13
==46D5
0513
1420
090E
0308
2C30
322D    DBYTE $322D,
```

```
$3939
3939
  TABLE                    ;  Heater Control Look-up Table
==46E3  TABLE
FFFF    WORD  $FFFF,$
E000,$B300,$8000,$5A
00,$4A00,$4000,$3800
00E0
00B3
0080
005A
004A
0040
0038
==46F3
0032    WORD  $3200,$
0101
0101
```

APPENDIX B

Video Screen Displays

This appendix illustrates the video screen displays used to advance the operator through the processing sequence. Each new displayed message is noted to the right of the "screen".

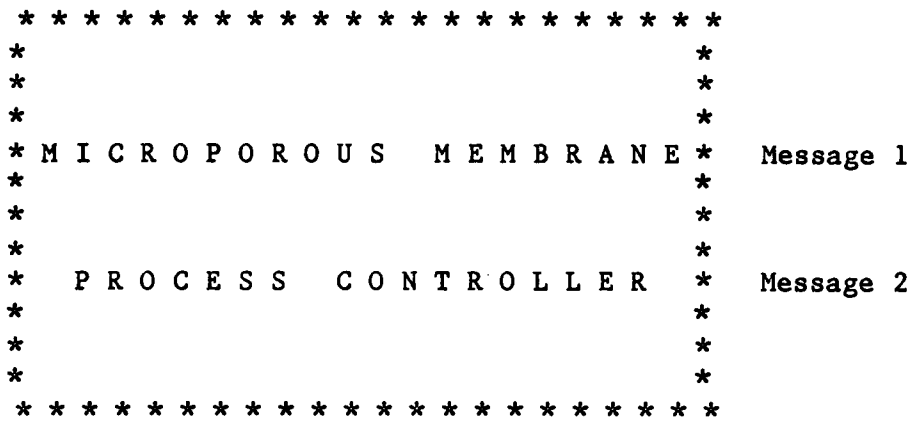


Figure B1 Monitor Display after SCRN1 Subroutine Call

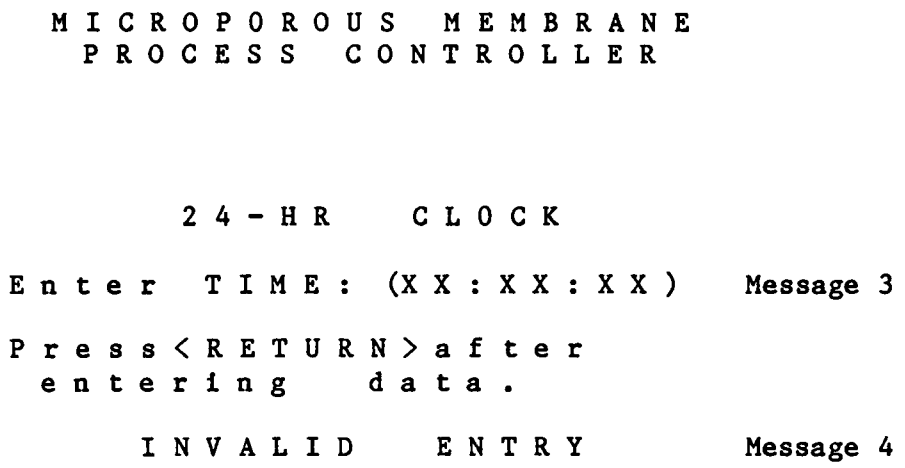


Figure B2. Monitor Display after SCRN2 Subroutine Call

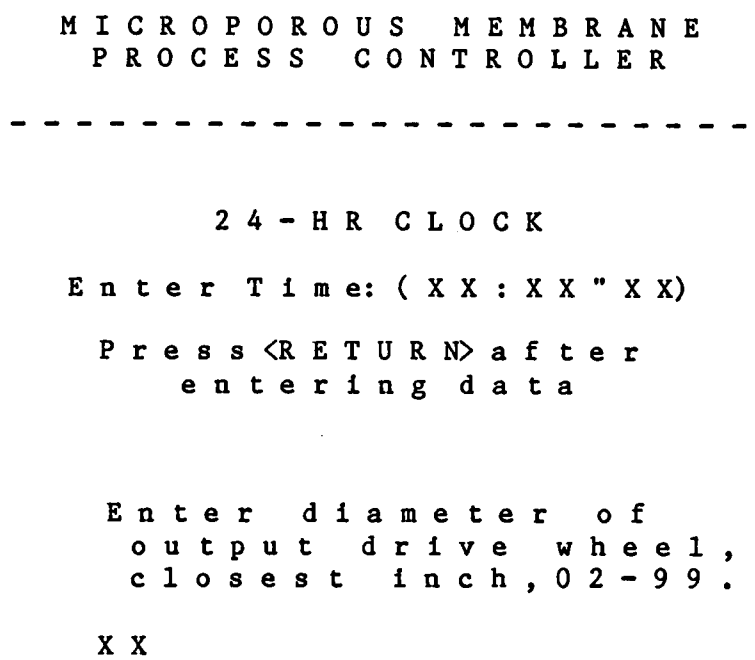


Figure B3. Monitor Display after SCRN2A Subroutine Call

```
MICROPOROUS MEMBRANE
  PROCESS CONTROLLER

- - - - - - - - - - - - - - - - - - - - - - - -

Process Preparation                    Message 5

  Enter an operation
code from below,                       Message 6
followed by <RETURN>.

      INVALID ENTRY.
         XX:XX:XX
- - - - - - - - - - - - - - - - - - - - - - - -
R1/S1=RUN/STOP Motor1
R2/S2=RUN/STOP Motor2                  Message 7
RB=RUN Both Motors
CONT=Continue

<SPACE BAR> = Emgcy STOP
```

Figure B4. Monitor Display after SCRN3 Subroutine Call

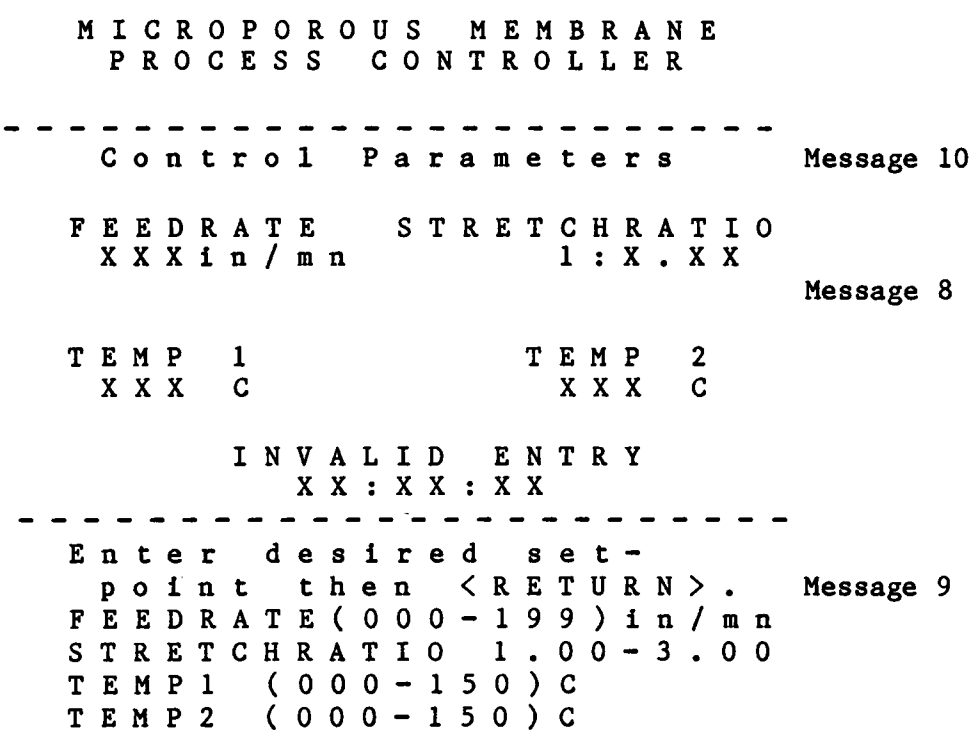


Figure B5. Monitor Display after SCRN4 Subroutine Call

```
        M I C R O P O R O U S   M E M B R A N E
          P R O C E S S   C O N T R O L L E R

- - - - - - - - - - - - - - - - - - - - - - - - -

          C o n t r o l   P a r a m e t e r s

        F E E D R A T E        S T R E T C H R A T I O
          X X X i n / m n                1 : X . X X X


        T E M P   1                    T E M P   2
          X X X   C                      X X X   C


                  X X : X X : X X
- - - - - - - - - - - - - - - - - - - - - - - - -

        E n t e r   a p p r o p r i a t e
        h e a t e r   o p e r a t i o n   c o d e ,  Message 11
        t h e n < R E T U R N > .
```

Figure B6. Monitor Display after SCRN 5 Subroutine Call

```
        MICROPOROUS MEMBRANE
          PROCESS CONTROLLER

- - - - - - - - - - - - - - - - - - - - - - - -
          Control Parameters

      FEEDRATE        STRETCHRATIO
        XXXin/mn            1:X.XX


      TEMP 1                    TEMP 2
        XXX  C          (des)     XXX  C
        XXX             (act)     XXX
              INVALID ENTRY
                 XX:XX:XX
- - - - - - - - - - - - - - - - - - - - - - - - -
      H1/01=HEATER1  ON/OFF
      H2/02=HEATER2  ON/OFF
      HB=BOTH HEATERS  ON          Message 12
      CONT=continue

      <SPACE BAR>=Emgcy.STOP
```

Figure B7. Monitor Display after SCRN6 Subroutine Call

```
        MICROPOROUS MEMBRANE
          PROCESS CONTROLLER

- - - - - - - - - - - - - - - - - - - - - - - - -

          Control Parameters

    FEEDRATE          STRETCHRATIO
      XXXin/mn              1:X.XX

    TEMP 1                    TEMP 2
      XXX  C       (des)      XXX  C
      XXX               (act)  XXX
              INVALID ENTRY
                 XX:XX:XX
- - - - - - - - - - - - - - - - - - - - - - - - -
    WAITING FOR HEATERS

CHANGE=Change Parametrs                Message 14
QUIT=Terminate Process

<SPACE BAR>=Emgcy.STOP
```

Figure B8. Monitor Display after SCRN7 Subroutine Call

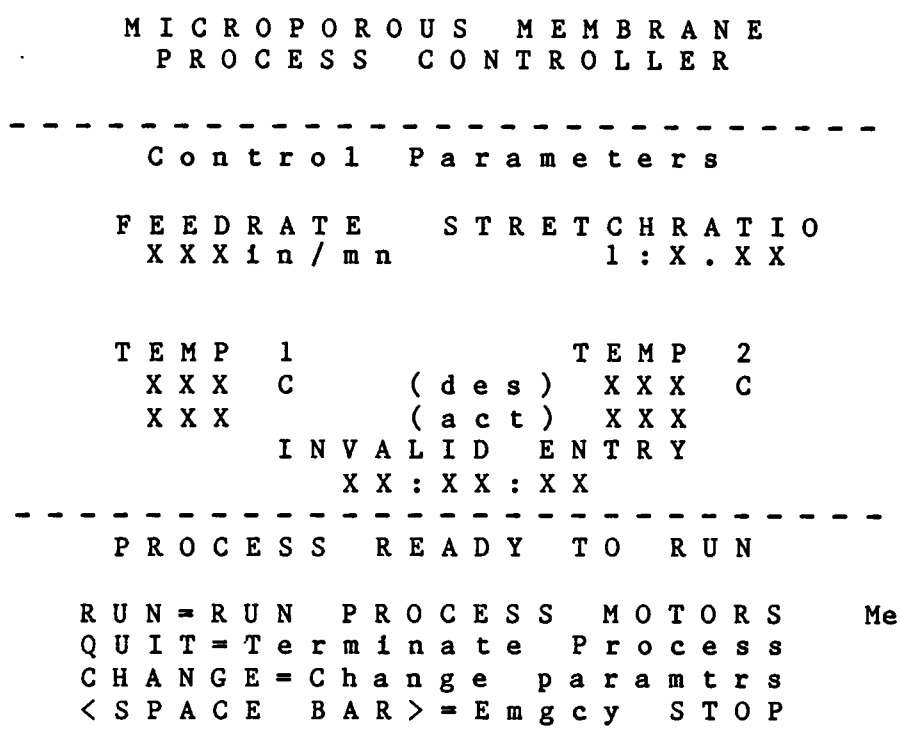


Figure B9. Monitor Display after SCRN8 Subroutine Call

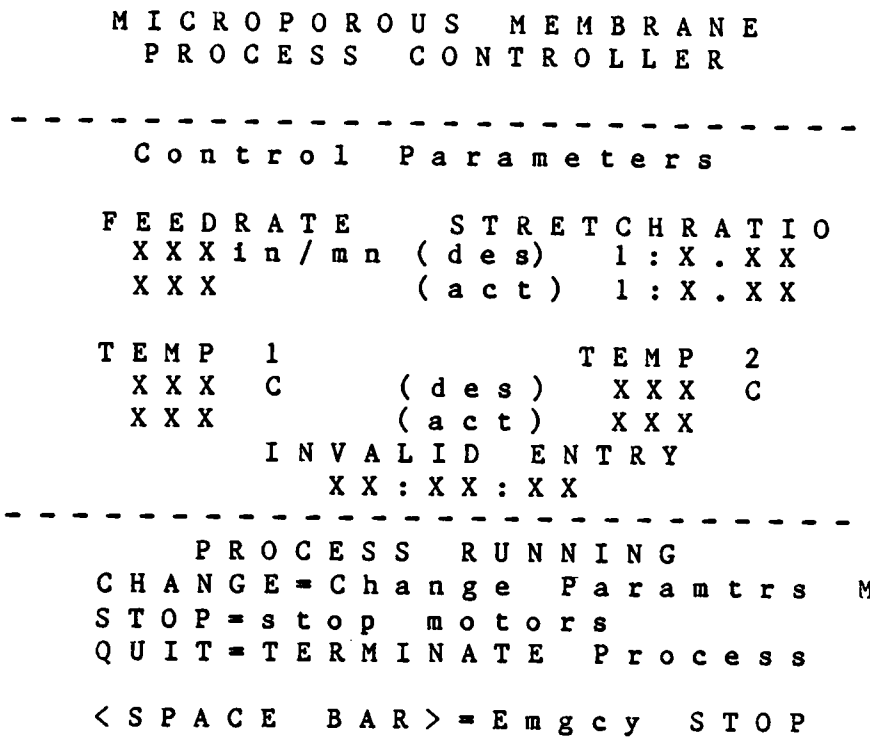


Figure B10. Monitor Display after SCRN9 Subroutine Call

APPENDIX C

VIC 20 Power-Up Initialization Squence

This appendix lists the power-up initialization sequence contained in the VIC 20 Kernal ROM.

Hex Address	Op Code		Comment
FD22	LDX #$FF	;	
	SEI	;	Disable interrupts
	TXS	;	Set stack pointer to $00FF
	CLD	;	
	JSR $FD3F	;	Test for Auto-Start sequence
	BNE $FD2F	;	If not found, continue
	JMP ($A000)	;	If found, jump to user routine
FD2F	JSR $FD8D	;	RAM test and clear
	JSR $FD52	;	Set vectors
	JSR $FDF9	;	Initialize I/O
	JSR $E518	;	Initialize screen
	CLI	;	Enable Interrupts
	JMP ($C000)	;	Basic Area

APPENDIX D

Construction Details

This appendix presents construction details of the interface and buffer circuits. The buffer circuit layout, shown in Fig. D1, is contained on a single VECTORBORD ^{R} wire-wrap perf board. Likewise, the interface circuit layout, shown in Fig. D2, is housed on a single VECTORBORD. ^{R}

All integrated circuit chips are socketed with tin or gold wire-wrap sockets. Input and output connections are made with wire-wrap terminals. Tables D1 through D4 list the components on the two boards.

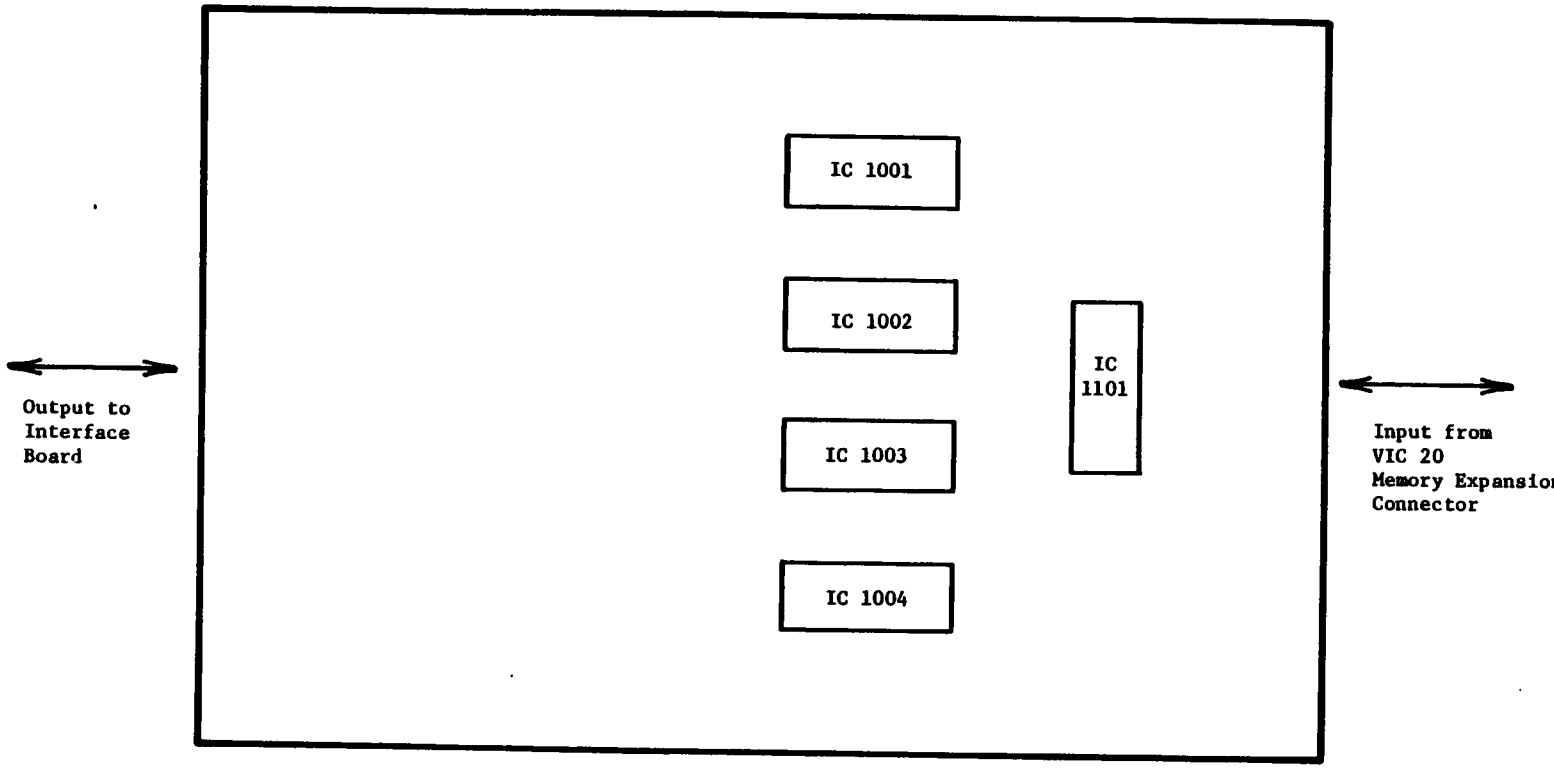


Figure D1. Buffer Circuit Board Layout

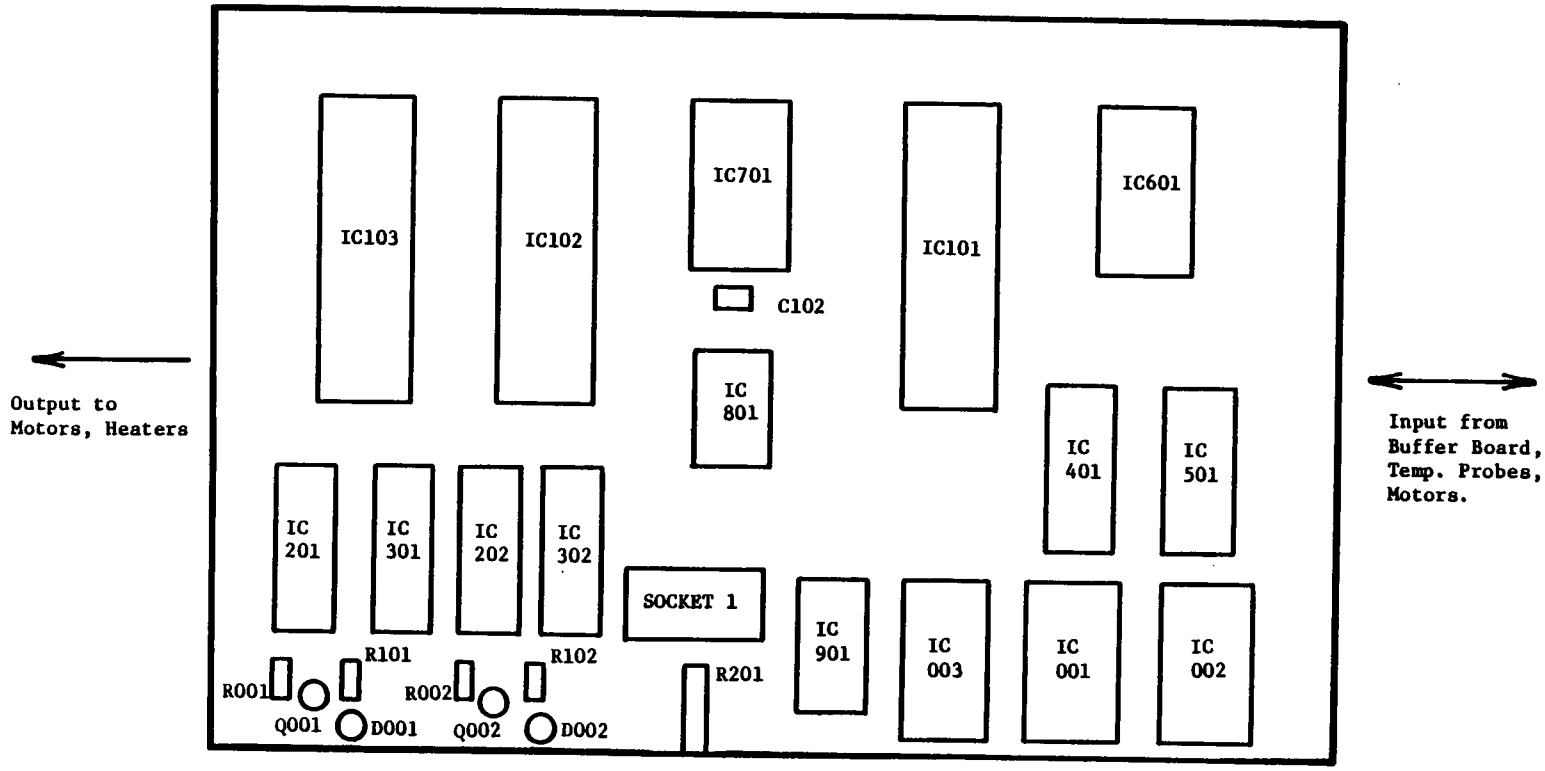


Figure D2. Interface Circuit Board Layout

Table D1. Discrete Components - Resistors

Component	Value	
R001	1K	
R002	1K	
R101	220 Ω	
R102	220 Ω	
R201	10 K Pot.	
		Socket 1 Pins
R301	470K	5 - 10
R401	150K	6 - 9
R501	33K	1 - 14
R502	33K	3 - 12

Table D2. Discrete Components - Capacitors

Component	Value	Socket 1 Pins
C001	10 MFD	2-13 (+)
C002	10 MFD	4-11 (+)
C101	0.1 MFD	7 - 8
C102	0.1 MFD	

Table D3. Discrete Components - Transistors and Diodes

Component	Value
Q001	2N2222
Q002	2N2222
D001	LED
D002	LED

Table D4. Discrete Components - Integrated Circuits

Component	Value	
IC 001	2716	EPROM
IC 002	2716	EPROM
IC 003	2716	EPROM
IC 101	6522	VIA
IC 102	6522	VIA
IC 103	6522	VIA
IC 201	74121	One-Shot
IC 202	74121	One-Shot
IC 301	74192	Counter
IC 302	74192	Counter
IC 401	7400	NAND Gate
IC 501	7404	Inverter
IC 601	74154	Decorder
IC 701	14433	A/D Converter
IC 801	1403	Voltage Reference
IC 901	4016	CMOS Switch
IC 1001	74245	Buffer
IC 1002	74245	Buffer
IC 1003	74245	Buffer
IC 1004	74245	Buffer
IC 1101	7420	NAND Gate

APPENDIX E

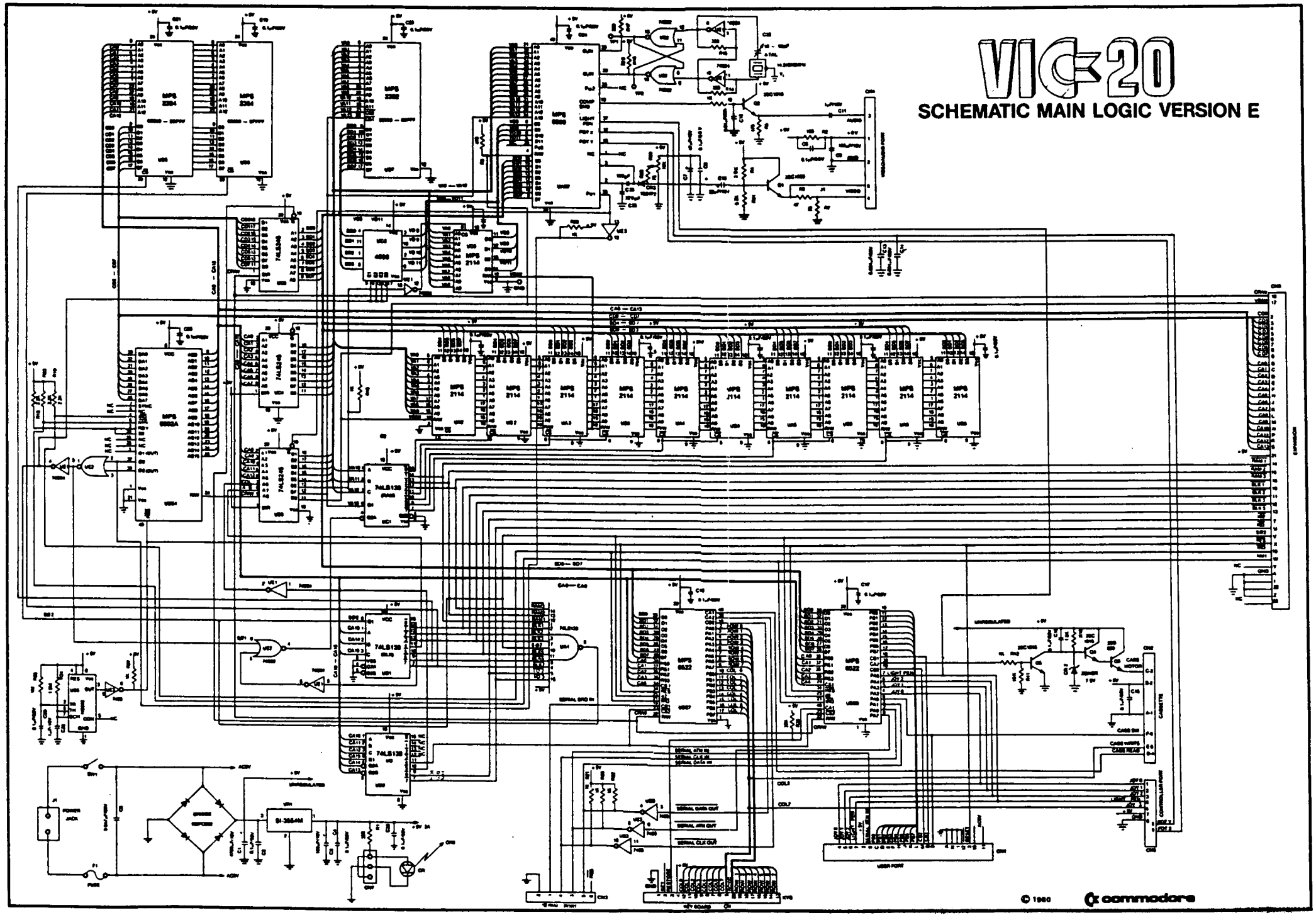


Figure E1. VIC 20 Schematic

APPENDIX F

Operator's Feed rate/Stretch-ratio Selection Procedure

This appendix presents a copy of the system operator's guide for selecting a valid feed rate and stretch-ratio combination. Only one calculation is necessary to determine if the selected combination can be achieved by the motors.

INPUT PARAMETER CHECK

FOR MICROPOROUS MEMBRANE PROCESS CONTROLLER

PROCEDURE TO CHECK VALIDITY OF FEED RATE/STRETCH-RATIO PARAMETER COMBINATION:

1. Choose a valid Feed rate (FR) from Table 1.
2. Choose a Stretch-ratio (SR) and use Eq. 1 to determine validity.

Table 1. Valid Feed Rate Range

Drive Diameter	Valid Feed Rate Range (in/min)	
	min	max
2"	20	199
4"	40	199
6"	60	199
8"	80	199

$$X = 4602 \cdot \frac{d \cdot SR}{FR} \qquad \text{(Eq. 1)}$$

A Stretch-ratio is valid if Eq. 1 gives X in the range $24 \leq X \leq 210$.

d = Drive diameter
SR = Chosen Stretch-ratio
FR = Valid Feed Rate

The vita has been removed from the scanned document

DESIGN OF A MICROCOMPUTER-BASED MICROPOROUS MEMBRANE PROCESS CONTROLLER

by

Douglas R. Browning

(ABSTRACT)

A microcomputer-based process controller has been developed to produce porous membrane material. The production process is based on stretching the material in a constant temperature solvent bath. This thesis describes the hardware and software designed to direct and monitor the process.

A VIC 20 is used as the process-controlling microcomputer. The system features two dedicated motor controllers and two channels for controlling temperature. The motor controllers determine the material feed rate and rate of stretch. The temperature controllers keep the system at a selectable constant temperature. An interactive assembly language program directs the entire process and monitors the controlled variables.

A complete description of the interface and temperature control circuitry is given. The software used to direct the process is also discussed and presented.