To the Graduate Council:

I am submitting herewith a thesis written by Siak Thong Yeo entitled "The Use of Embedded Microcontrollers in the Design of a Low-Cost Industrial Robot." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Robert W. Rochelle, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

THE USE OF EMBEDDED MICROCONTROLLERS IN THE DESIGN OF A LOW-COST INDUSTRIAL ROBOT

A Thesis

presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Siak Thong Yeo

June 1988

DEDICATION

This thesis is dedicated to my parents, sisters, and brothers. It would not have been possible for me to obtain my Bachelor of Science and Master of Science degrees from the University of Tennessee, Knoxville if not for their constant support, patience, and understanding.

ACKNOWLEDGMENTS

I would like to thank the International Business Machines Corporation (IBM) for this research assistantship. This research was being sponsored under their Technical Interchange Program. I would also like to thank the Electrical and Computer Engineering department of the University of Tennessee (Knoxville) for offering me a graduate assistantship while I was persuing my Master's degree there.

I especially like to express my deep gratitude to Dr. Robert W. Rochelle who obtained this research assistantship for me and also another research assistantship prior to this. He has been providing me with invaluable assistance, advise, and suggestions. His patience and dedication are also greatly appreciated. I also like to thank Dr. Robert Bodenheimer and Dr. Frank Speckhart for their assistance and for serving on my committee. It has been a great pleasure to work with Mark Granger who is the graduate student who worked on the mechanical side of this project. Our many discussion sessions have brought frequent "break-throughs" in getting the robot to work.

Finally, I like to thank Mr. Vic Hair, Mr. Paul Hackenworth, Mr. Dave Langer, and Mr. Gene Rohrer of IBM for their reviews and suggestions.

ABSTRACT

Embedded microcontrollers were used to control the X-Y (or horizontal) motions of a high-speed, low-cost industrial robot. The design and implementation were based on the Intel 8031 (the ROMless version of the Intel 8051) microcomputer. The X-Y or horizontal motions of the robot were accomplished by using four four-phase stepper-motors (two for each axis). Two microcontrollers, one for each axis, were used to step and control the stepper-motors. An IBM PC/AT was used as the main system controller, which transmits both data and instructions to the microcontrollers.

A twisted pair was used to establish the serial communications between the IBM PC/AT and the microcontrollers. The communication protocol that was chosen uses the built-in serial communication feature of the Intel 8031. A user-friendly BASIC program was written to allow the robot user to download data and send instructions to the microcontroller of the robot system. However, assembly language routines, to be called by the main program, were necessary in order to transmit data in the nine-bit format required by the serial communication mode of the microcontrollers.

A second BASIC program was written to convert informations, entered into the IBM PC/AT by the user as moves or tasks for the robot, into "move" data for the X and Y microcontrollers. A third BASIC program was also written to convert acceleration and deceleration profiles of the stepper-motors into "ramp-up" and "ramp-down" data for the Intel 8031 microcontrollers.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION 1
   1.1 Description of the Robot Mechanism 2
   1.2 Description of the Robot Control System 4

2. HARDWARE DESIGN OF THE ROBOT CONTROL SYSTEM . . . 7
   2.1 Features of the Intel 8031 Microcomputer 7
   2.2 Parallel Communication vs Serial Communication 8
   2.3 External Memory of the Microcontrollers 9
   2.4 Control Lines from the 8031 to the Stepper-Motor Drivers 11
   2.5 Stepper-Motor Driver Used to Test the 8031 Programs 13
   2.6 Hardware and Interface of the X and Y Microcontrollers 13

3. SOFTWARE DESIGN OF THE ROBOT CONTROL SYSTEM . . 21
   3.1 Types of Data and Instructions for the Microcontrollers 22
   3.2 Robot Control Program for the Main Controller 27
   3.3 Assembly Language Routines for Serial Communications 30
   3.4 Program for the X and Y Microcontrollers 32
   3.5 Programming of the EPROM 41
   3.6 Program to Create the "move" Data Set 42
   3.7 Program to Convert Acceleration and Deceleration Profiles to "ramp-up" and "ramp-down" Data 45

4. SUGGESTIONS FOR FUTURE IMPROVEMENTS AND ADDITIONS 47

5. RESULTS AND CONCLUSIONS 50

CHAPTER PAGE

LIST OF REFERENCES 52

APPENDICES 54

A Pin-Outs and Special Function Registers of the Intel 8031 55

B Flowcharts of the BASIC Program ROBOT3 57

C Flowcharts of the Assembly Language Program ROBOT 78

D Flowcharts of the MCS-51 Assembly Language Program ROX 83

E Flowcharts of the BASIC Program MOVE 100

F Flowchart of the BASIC Program RC 107

G Listing of BASIC Program ROBOT3.BAS 109

H Listing of the Assembly Language Program ROBOT.ASM 118

I Listing of the MCS-51 Assembly Language Program ROY.SRC. 123

J Listing of the BASIC Program MOVE.BAS 137

K Listing of the BASIC Program RC.BAS 143

VITA 145

LIST OF FIGURES

FIGURE		PAGE
1.1	Prototype of the Robot with the Conceptual Z-Arm	3
1.2	Block diagram of the Robot Control System	5
2.1	Diagram of the Modified Proprinter Board	12
2.2	Control Lines for Stepper-Motor Drivers	12
2.3	Circuit Diagram of Stepper-Motor Driver for the 1.1 Amp Motor	14
2.4	Pin-Outs of the D-Shell Connector of the Serial/Parallel Adaper	15
2.5	Serial Communication Interface of the Main Controller to a Microcontroller	17
2.6	Circuit Diagram of Hardwares and Interfaces of the Microcontroller	19
2.7	Hardware Lay-Out of the X and Y Microcontrollers	20
3.1	Rough Sketch of the Segments Involved in a Typical Stepper-Motor Move	26
3.2	Simplified flowchart of Interrupt Service Routine (ISR) for the Microcontroller	34
3.3	Simplified flowchart of Subroutine to Receive Incoming Data	35
3.4	Phase Patterns for Clockwise Step	38
3.5	Phase Patterns for Counter-Clockwise Step	38
3.6	Using an AND Gate to Synchronize Moves When More Than Two Microcontrollers are Used	40

CHAPTER 1

INTRODUCTION

The International Business Machines Corporation (IBM) is currently involved with the University of Tennessee in a Technical Interchange Program (TIP). This program enables universities to do research on IBM-sponsored projects. One such project involves the design and prototyping of a high-speed, low-cost industrial robot. This project has supported three Mechanical Engineering senior design classes. It also supported for a year one graduate student in Electrical and Computer Engineering and one graduate student in Mechanical Engineering.

Currrent industrial robots are too expensive and slow to be used for light and simple assembling tasks on the assembly line. There is a definite need for a high-speed, low-cost industrial robot to accomplish placement tasks. IBM therefore suggested to the Mechanical Engineering department of the University of Tennessee, Knoxville that a project be established to design and build a stepper-motor-driven robot with open-loop control having a simple Cartesian mechanical layout. The use of stepper-motors would provide a very inexpensive way to control the positioning of the robot. The motors are also capable of performing at high speed. The open-loop control was desirable because it not only could cut down on the cost of the electrical hardware, but it also could make the control design less complex, eliminating the need for microcontrollers with high computing power. A Cartesian mechanical layout would keep the cost of the mechanism low and would greatly simplify the kinematics and dynamics involved in the control of the robot. In simple terms, the robot has to be low-cost, high-speed, and easy to control and use.

In controlling the robot, IBM has envisioned using the IBM PC/AT as the main controller of the robot system. IBM has therefore graciously provided an IBM PC/AT along with the necessary technical literature for use in the project.

The microcomputers used for controlling the stepper-motors were the Intel 8031. The design of the Intel 8031 single-chip microcomputer is optimized for control applications. Extensive on-chip support is provided for one-bit variables as a separate data type, allowing direct bit manipulation and testing in control and logic systems that require Boolean processing [1]. It also has a larger program and data memory, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer [1]. The Intel 8031 is therefore extremely well suited to be used as the microcontroller for each degree of freedom of the robot. In addition, the Intel Corporation has been a close business partner of IBM and the use of Intel chips for an IBM-sponsored project will surely be appreciated by both corporations.

1.1 Description of the Robot Mechanism

The mechanism for the X-Y motion of the robot was designed and built by the Mechanical Engineering graduate student, Mark Granger. The maximum speed for motion along each axis is 35 inches per second. The mechanism for another linear (the vertical or Z-axis) and three rotary degrees of freedom is still in the design stage. Therefore, the controls related to those four additional degrees of freedom have not as yet been implemented.

The prototype of the industrial robot is shown in Figure 1.1. A conceptual Z-arm is included to depict what a finished robot will look like. The gripper (not shown in the Figure 1.1) will be at the base of the Z-arm. The control tube labeled **p** is driven

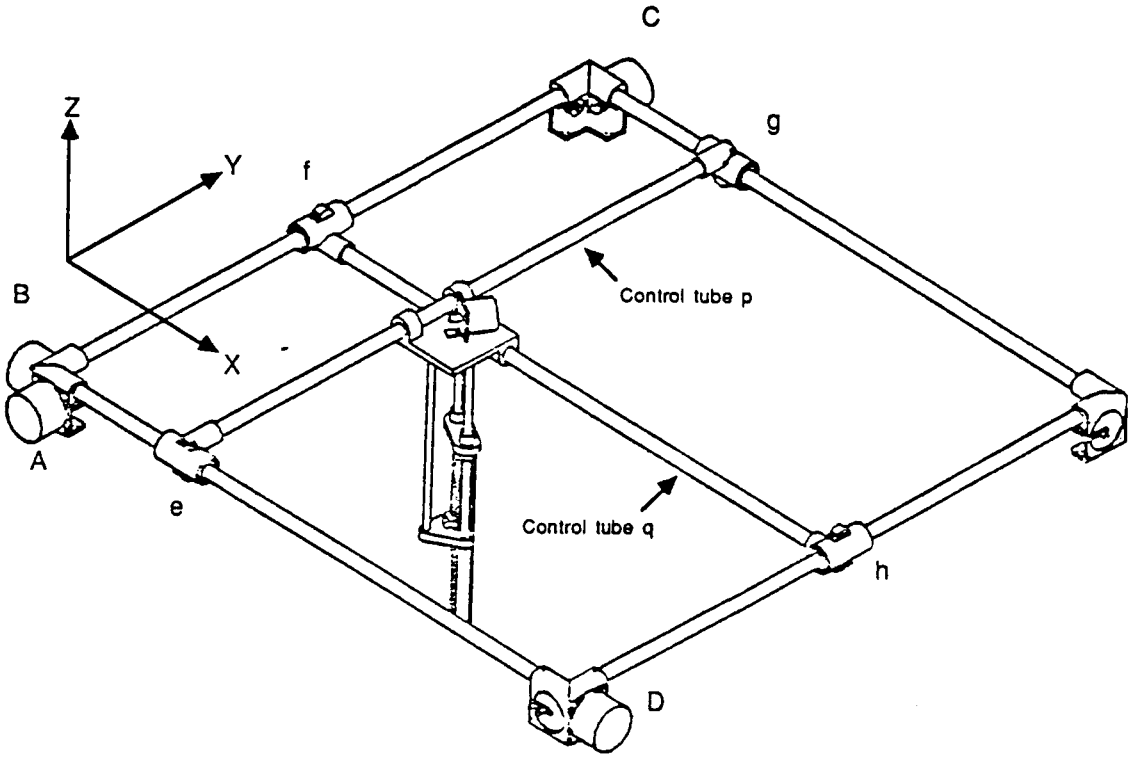


Figure 1.1 Prototype of the Robot with the Conceptual Z-Arm

by stepper-motors A and C, and the control tube labeled **q** is driven by stepper-motors B and D. Control tube **p** controls motion in the X direction and control tube **q** controls motion in the Y direction. For the control tube **p** to move, slider bracket **e** was fixed to a timing belt that is mounted on a timing gear on the stepper-motor A, and slider bracket **g** was fixed to another timing belt mounted on a timing gear on the stepper-motor C. Similarly, control tube **q** has slider brackets **f** and **h** fixed to timing belts that are mounted on timing gears on stepper-motors B and D respectively (the timing belts are not shown in Figure 1.1).

Since the robot designed is X-Y Cartesian based, it was useful to define coordinates (0,0) as being the home position. In reality, this home position , (0,0), is the position of which the slider brackets **e** and **f** can get as close to the corner bracket (which houses stepper-motors A and B) as possible without any physical contact being made between them. (Please see Figure 1.1 for clarity.)

The stepper-motors used are 2.1-amp, four-phase, and 1.8 degree per step (or 200 steps per revolution) motors. Each stepper-motor requires its own driver, and IBM has provided all of them. These stepper-motor drivers are modified from scrapped IBM Proprinter boards which come with their own power supplies.

1.2 Description of the Robot Control System

The block diagram in Figure 1.2 shows the layout of the robot control system. An IBM PC/AT is used as the system or main controller which transmits data to the X and Y microcontrollers. Data flow was designed to flow one way from the IBM PC/AT to the microcontrollers, but a seperate line to be used by the microcontrollers to interrupt the PC (when the robot loses its "stepping" and needs to be reset) is forseeable in the final design. The system controller (which is the PC) will not be tied real-time to

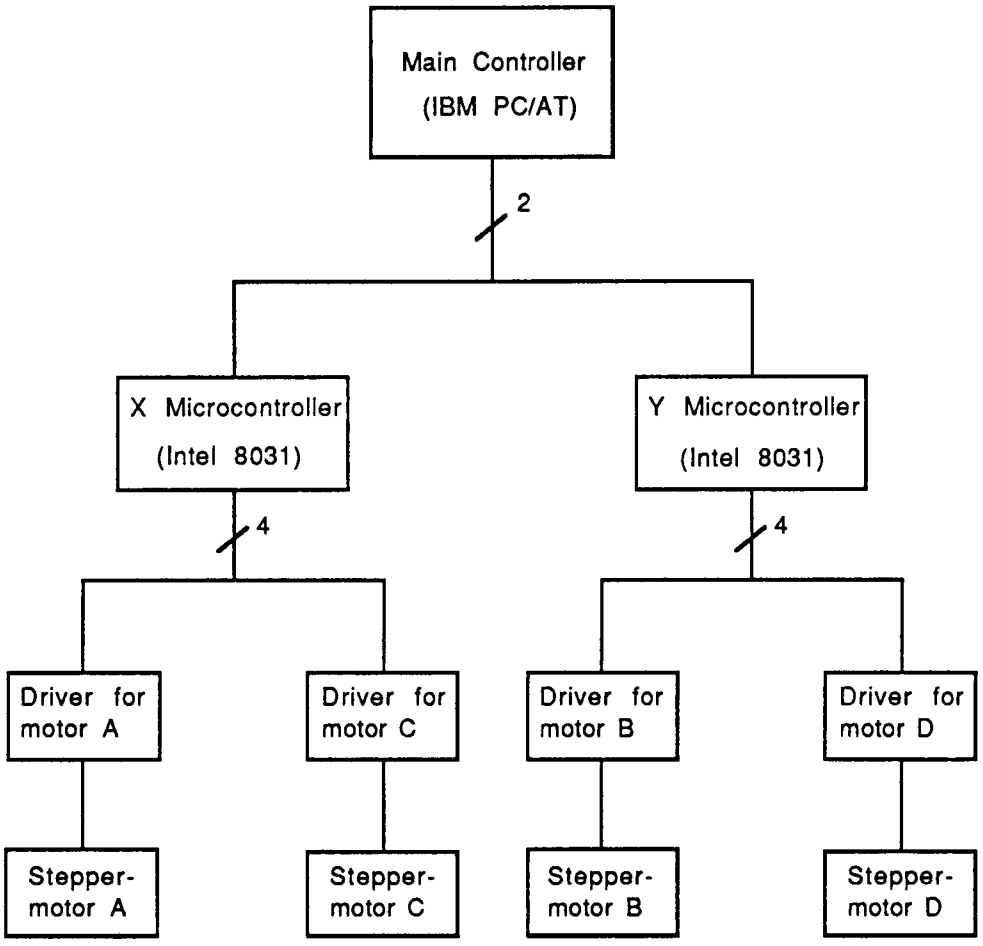


Figure 1.2 Block diagram of the Robot Control System

the microcontrollers and can be used to perform other tasks once the downloading of data and the sending of instruction(s) is completed. The physical link between the main controller and the microcontrollers was a twisted pair (a wire for the transmission line and a wire for the common ground).

The X microcontroller steps and controls both the A and C stepper-motors at the same time through their drivers. Therefore, stepper-motors A and C operate identically. The Y microcontroller controls the B and D stepper-motors the same way. A single crystal was used on the two microcontrollers to achieve synchronization.

The stepper-motor drivers used are converted from IBM Proprinter boards. They can drive stepper-motors at either full current (2.0 amp) or half current (0.8 amp). This feature made it possible to conserve energy and prevent heating while the stepper-motor is not "stepping".

CHAPTER 2

HARDWARE DESIGN OF THE ROBOT CONTROL SYSTEM

This chapter first discusses the useful features of the Intel 8031. The hardware design considerations and their corresponding implementations for the robot control system is next discussed as follows:

- Parallel communication vs serial communication
- External memory of the microcontrollers
- Control lines from the 8031 to the stepper-motor drivers
- Stepper-motor driver used to test the 8031 programs
- Hardwares and interfaces of the X and Y microcontrollers

2.1 Features of the Intel 8031 Microcomputer

The Intel 8031 is a high-performance single-chip microcomputer designed for use in sophisticated real-time applications. The 8031 integrates on a single chip an eight-bit CPU, 128 bytes of internal RAM (read/write data memory), a Boolean Processor, several Special Function registers, four 8-bit I/O ports of which two (ports P0 and P2) are used in accesses to external memory, serial I/O port for either multi-processor communications or full duplex UART, 111 assembly language instructions, and has the capability of addressing up to 64K bytes of external program memory and up to 64K bytes of external data memory. It also features two four-mode timer/counters and a five-source interrupt structure with two priority levels.

The I/O port P3 also serves the functions of various special features of the 8031

as listed below:

PORT PIN	ALTERNATE FUNCTION
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

The Intel 8031 utilizes an oscillator frequency in the range of 3 MHz to 12 MHz. Each machine cycle consists of 12 oscillator periods. The 8031 also features a reset pin RST. A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset. The reset reconfigures all the Special Function registers, but leaves the internal RAM contents unchanged. The pin-outs and Special Function registers of the 8031 can be found in Appendix A.

2.2 Parallel Communication vs Serial Communication

Communications (or data transfer) can be established either in parallel or in serial. Parallel communication allows for more data to be transmitted per second. It also requires a simpler interface and a less complex software [2]. However, parallel commuication is more susceptible to noise, especially in an assembly area where high voltage machinery operates. Parallel communication also requires more lines because it needs one line per bit (each bit-line will also need a ground coupling to help reduce the noise problem).

Serial communication has the advantage of requiring only a twisted pair. It is also less susceptible to noise, especially when the physical link uses the RS-232C format which transmit digital signals in the -12 to +12 volts range. But, serial communication tends to be slower, and the hardware and sofware more complex [2].

The IBM PC/AT came with a serial/parallel communications adapter that provides a parallel port and a serial port. The parallel port makes possible the attachment of various devices that accept eight bits of parallel data at standard TTL levels [3]. The serial portion of the adapter is fully programmable and start, stop, and parity bits can be added or removed. It also has a programmable baud-rate generator that allows operation from 50 baud to 9600 baud [3].

Since, the robot can be expected to be operated in a "noisy" environment and there will be at least six microcontrollers (one for each degree of freedom of the robot) that requires communications with the main controller, serial communication seems to be the best solution. The Intel 8031's capability for multiprocessor communication, which is a form of serial communication, also supported this choice. Although the hardware for serial communications is more complex, both the IBM PC/AT and the Intel 8031 already had them incorporated. The biggest advantage of using the multiprocessor communication feature of the the 8031 comes from the fact that only a twisted pair is needed betweeen the microcomputers.

2.3 External Memory of the Microcontrollers

The 8031 can address up to 64K bytes of external program memory. Since, the external program memory can only be read (not written), some kind of ROM has to be used. An MCS-51 Macro Assembler Development System was not available for this project. Therefore, the checking and testing of 8031 programs has to be done on the

prototype. This means that the 8031 program will have to be in the ROM first. In order to be able to change or alter the program in the ROM readily, an Erasable and Programmable ROM (an EPROM) has to be used.

The 8031 can also address up to 64K bytes of external data memory. The external data memory must provide the access of stored data (reading) and the ability to alter the stored data (writing). A RAM will have to be used for external data memory. There are two basic types of RAMs - static and dynamic. Dynamic RAMs are noted for high capacity, moderate speed and low power consumption. However, they require periodic charge refreshing to maintain the data storage [4]. This means that the circuitry to handle the dynamic RAM subsystem refresh will have to be added. Static RAMs store ones and zeros (data) using traditional flip-flop logic-gate configurations [4]. They retain their content as long as the power remains on. They are faster and require no refresh circuitry.

In determining the EPROM and RAM size to be used, there were two important considerations. The first involved the size of the final 8031 program code. The second was should the "ramp-up" and "ramp-down" data be stored in the EPROM or in the RAM. After the programs were developed for the X and Y microcontrollers, it was found that their program code were less than 1K bytes each. The total of the "ramp-up" and "ramp-down" data turned out to be about 1.4K bytes. The "move" data which will be stored in the RAM was designed to be in a "compact" form (the "move" data format is discussed in chapter 3). Therefore, both the external program memory and external data memory requirements turned out to be rather low.

A 4K bytes EPROM was consequently chosen to store the 8031 program code. To be compatible with the 8031, the generic Intel 2732 EPROM was used. The present "ramp-up" and "ramp-down" data could periodically be revised or changed due to the

changes in the mechanical design of the robot. This made it more practical to have the "ramp-up" and "ramp-down" data in the RAM. Since, there is no need to use a high capacity RAM for the data memory, static RAM was used so as to avoid having to incorporate the extra circuitry required by the high capacity dynamic RAM. The static RAM used was the generic 2016 which has 2K bytes of storage space. The static RAM therefore provided storage for the "ramp-up", "ramp-down", "move", and "robot" data.

2.4 Control Lines from the 8031 to the Stepper-Motor Drivers

The stepper-motors used in the robot have four-phases. The four phases are A, $\overline{A}$, B, and $\overline{B}$. Since two of the phases are always the complement of each other, the Proprinter driver took advantage of this by only requiring the controller to supply phases A and B. It generates phases $\overline{A}$ and $\overline{B}$ internally. The Proprinter also has the capability of driving the stepper-motor at full or half current, selected by setting a bit high or low, respectively. When the stepper-motor is "stepping" it requires the full current (2.0 amp) to generate sufficient torque. However, the stepper-motor still needs to maintain some kind of torque to hold its position (assuming that the load is light) when it is not "stepping". The half current therefore keeps the stepper-motor "locked" and at the same time cuts down on the power consumption.

Three control lines were needed by each driver. The I/O ports P1.4 and P1.5 of the 8031 were used to control phases A and B of the stepper-motor, respectively. The current was controlled by the I/O port P1.0. Figure 2.1 shows where the three control lines are to be connected on the modified Proprinter board. Since, both the X and Y microcontrollers controls two stepper-motors each, their corresponding drivers will share the same three control lines. This is shown in Figure 2.2. below.

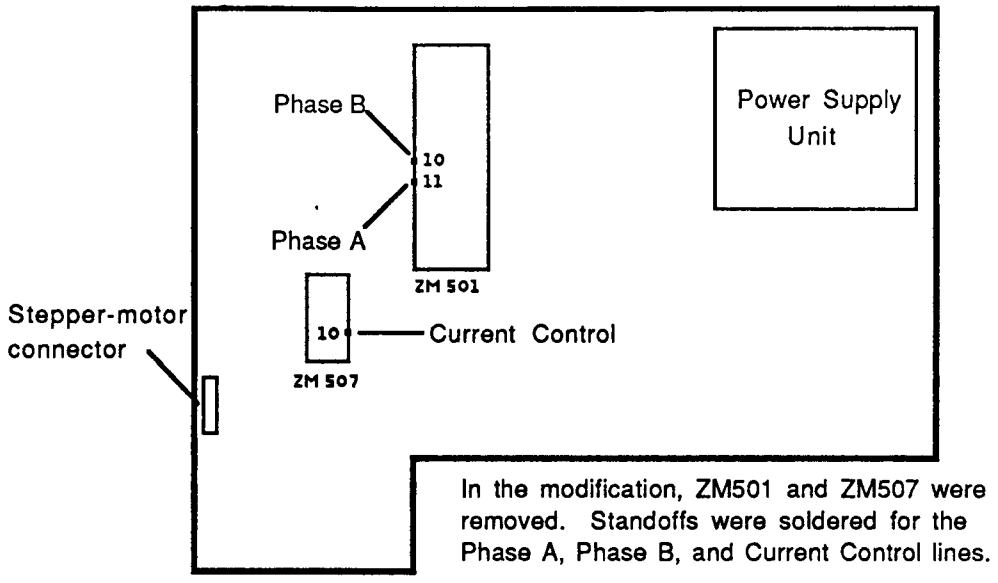


Figure 2.1 Diagram of the Modified Proprinter Board

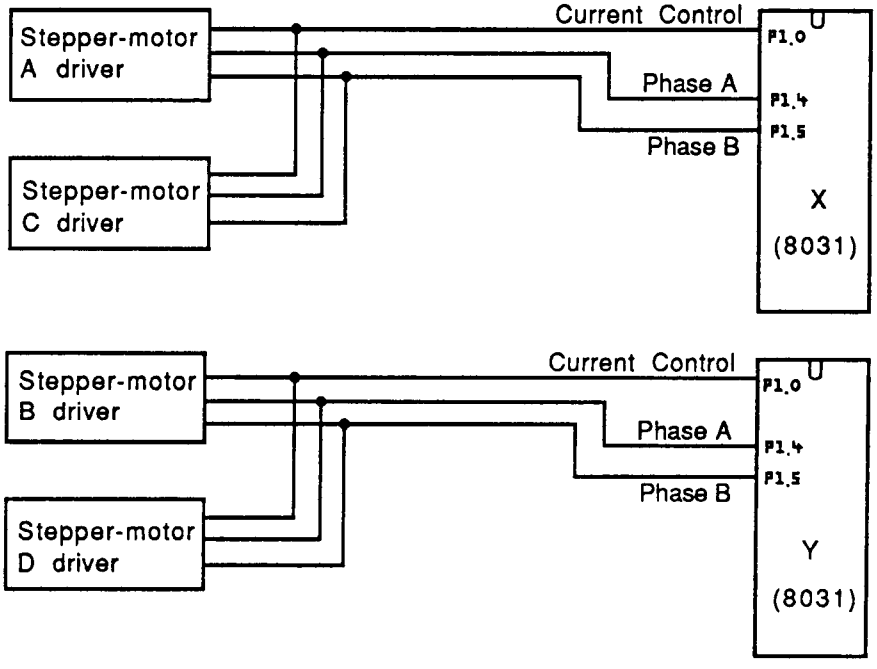


Figure 2.2 Control Lines for Stepper-Motor Drivers

2.5 Stepper-Motor Driver Used to Test the 8031 Programs

In the initial testing of the 8031 programs, neither the modified Proprinter board nor the 2.1 amp stepper-motor were used. A 1.1 amp, four-phase, 1.8 degree per step motor was used instead. The stepper-motor driver used to drive this 1.1 amp motor was built by using a stepper-motor driver chip (SanKen SI-7301) provided by IBM. The schematic of the stepper-motor driver circuit is shown in Figure 2.3. This driver requires the controller (the 8031) to provide all the four phases - A, $\overline{A}$, B, and $\overline{B}$. It also requires the controller to provide a current control signal to set the stepper-motor at either full or half current - just like the modified Proprinter board's Current Control line. The I/O ports P1.4 to P1.7 of the 8031 were used to provide the four phases A, B, $\overline{A}$, and $\overline{B}$ respectively. I/O port P1.0 provided the current control signal.

Although both the 1.1 amp stepper-motor and its driver used for testing the 8031 programs were not incorporated in the final robot design, it was extremely helpful to have them. They were used regularly to test and debug the 8031 programs.

2.6 Hardware and Interface of the X and Y Microcontrollers

This section discusses how the hardware and interfacing electronics used on the microcontrollers were organized and laid out. Since, both the X and Y microcontrollers have similar circuitry, only one of them will be discussed.

The serial adapter port of the IBM PC/AT uses a nine-pin D-shell connector that is classified as an RS-232C port. The functions of the nine pins are shown in Figure 2.4. Of the nine pins, only pin three (Transmit Data pin) and pin five (Ground) are needed since the communication is only one way. On the receiving end, the 8031's serial input port (RXD) uses the TTL format. In order to interface the RS-232C Transmit Data line to the TTL RXD line, an RS-232C to TTL converter (1489A) was

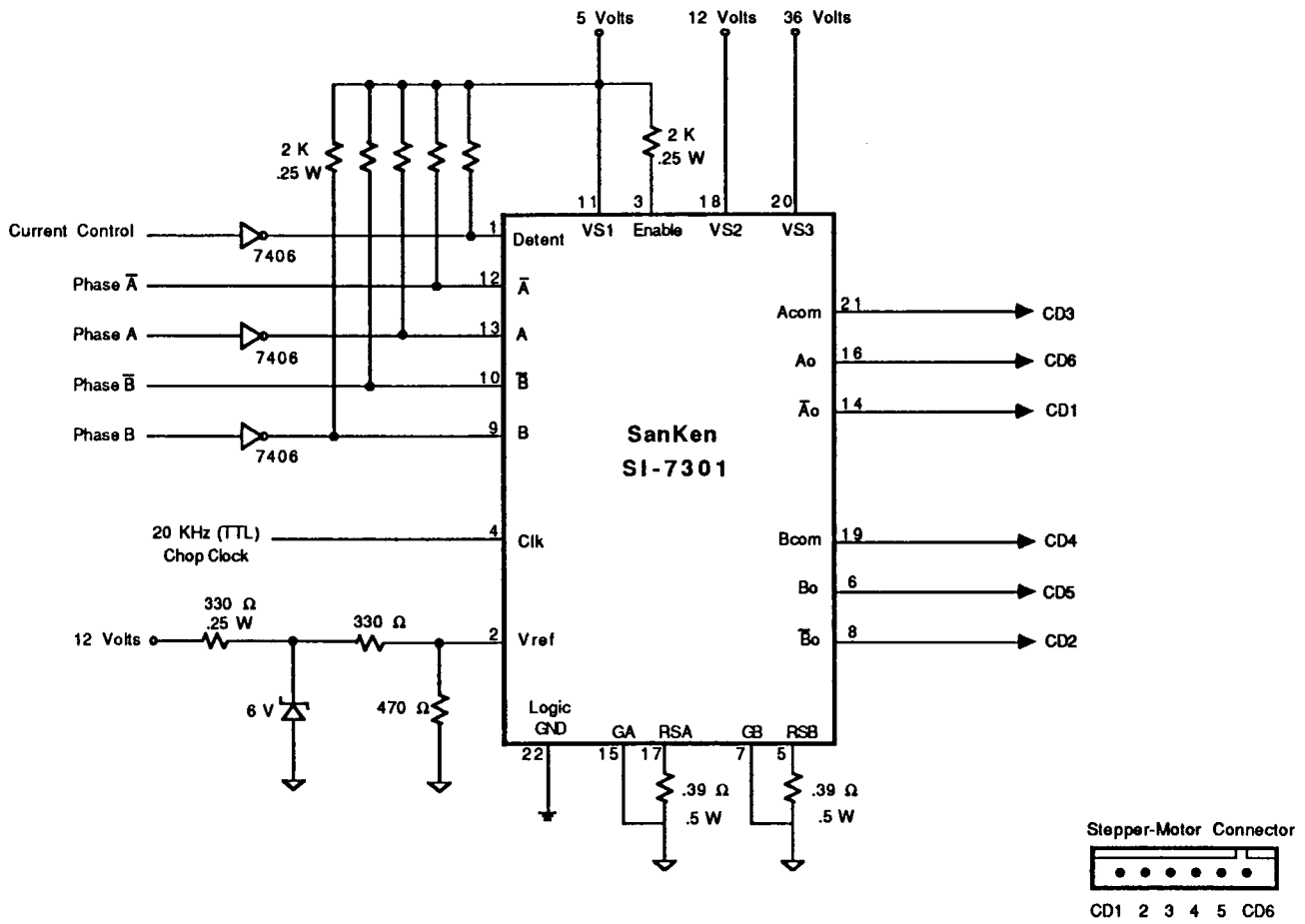


Figure 2.3 Circuit Diagram of Stepper-Motor Driver for the 1.1 Amp Motor

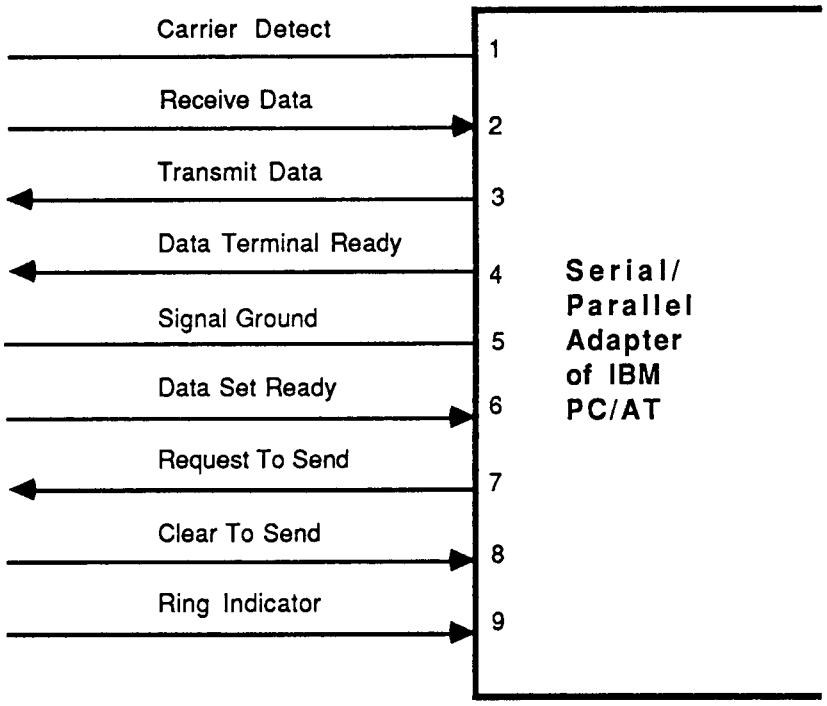


Figure 2.4 Pin-Outs of the D-Shell Connector of the Serial/Parallel Adapter

used as shown in Figure 2.5. A data transmission rate of 1200 baud was used.

The output of the 1489A will be connected to the serial input ports (RXDs) of all the microcontrollers in the robot system (multiprocessor mode). Therefore, the drive capability of the 1489A had to be determined to avoid overloading the device. The calculations are as follows:

At high state, maximum current the 1489A sources, I_{OH} = -0.5 mA

At high state, maximum input current for RXD, I_{IH} = 0.1 mA

The worst case loading at high state for 1489A is 0.5/0.1

= 5 devices

At low state, maximum current the 1489A can sink, I_{OL}= 10 mA

At low state, maximum current RXD sources, I_{IL} = -0.5 mA

The worst case loading at low state for 1489A is 10/0.5

= 20 devices

The above calculations show that each RS-232C to TTL converter (1489A) can drives at most five different 8031 serial port inputs (RXDs). Each 1489A chip comes with

The 8031 has two types of accesses to external memory - accesses to external program memory and accesses to external data memory. Accesses to external program memory (the EPROM) use signal $\overline{PSEN}$ (program store enable) as the read strobe. Accesses to external data memory (the static RAM) use $\overline{RD}$ or $\overline{WR}$ (alternate functions of P3.7 and P3.6) to strobe the memory. Fetches from external program memory always use a 16-bit address. Accesses to external data memory can use either a 16-bit address or an 8-bit address [1].

Whenever a 16-bit address is used, the high byte of the address comes out on port P2, where it is held for the duration of the read or write cycle. The low byte of the

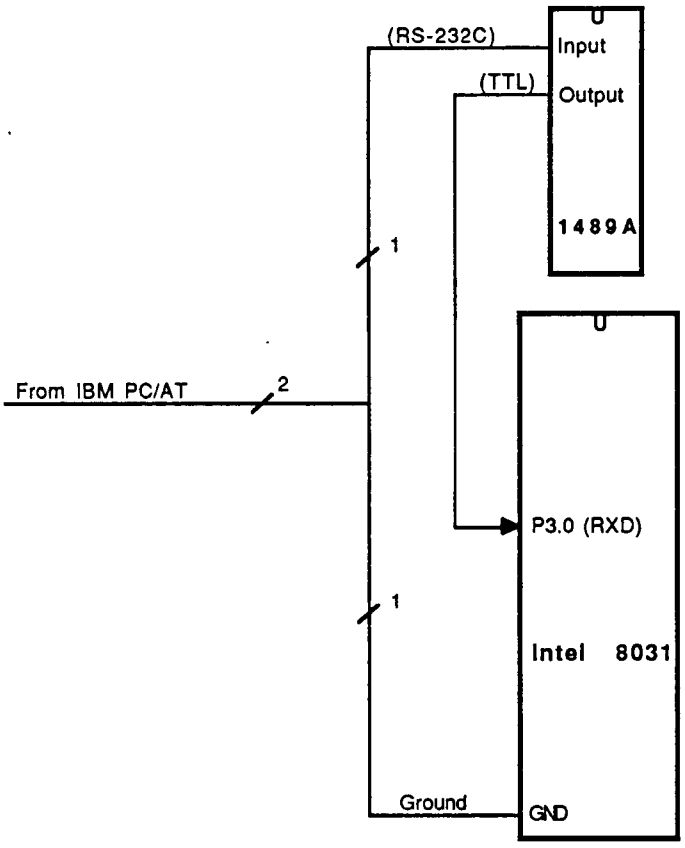


Figure 2.5 Serial Communication Interface of the Main Controller to a Microcontroller

address is time-multiplexed with the data byte on port P0. The signal ALE (Address Latch Enable) was used to capture the address byte into an external latch. The external latch used was the 74374 Octal D-type flip-flops with tri-state bus-driving outputs. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on the port P0 just before $\overline{WR}$ is activated, and remains there until after $\overline{WR}$ is deactivated. In a read cycle, the incoming byte is accepted at port P0 just before the read strobe is deactivated. The interfacing of the external program memory (EPROM) and external data memory (static RAM) to the 8031 is shown in Figure 2.6. The 8031 also requires that the signal $\overline{EA}$ be activated (tied low) to enable the program bytes to be fetched from the EPROM.

Only the lower four of the eight high-address bits of port P1 are needed for addressing the external program memory because the EPROM is only 4K bytes (see Figure 2.6). As for the addressing of external data memory in the static RAM, only the lower three of the eight high-address bits of port P1 are needed as the RAM size is only 2K bytes.

Figure 2.6 also shows the set-up for a power-up reset and a manual reset using the RST pin of the 8031. The manual reset is initiated by pushing a push-button. A 10-MHz crystal and two 30-pf capacitors are used for the oscillator circuitry.

In Figure 2.7 the complete lay-out of the hardwares and interfaces of the X and Y microcontrollers is found. This lay-out is useful for locating the actual locations of the different components on the breadboard.

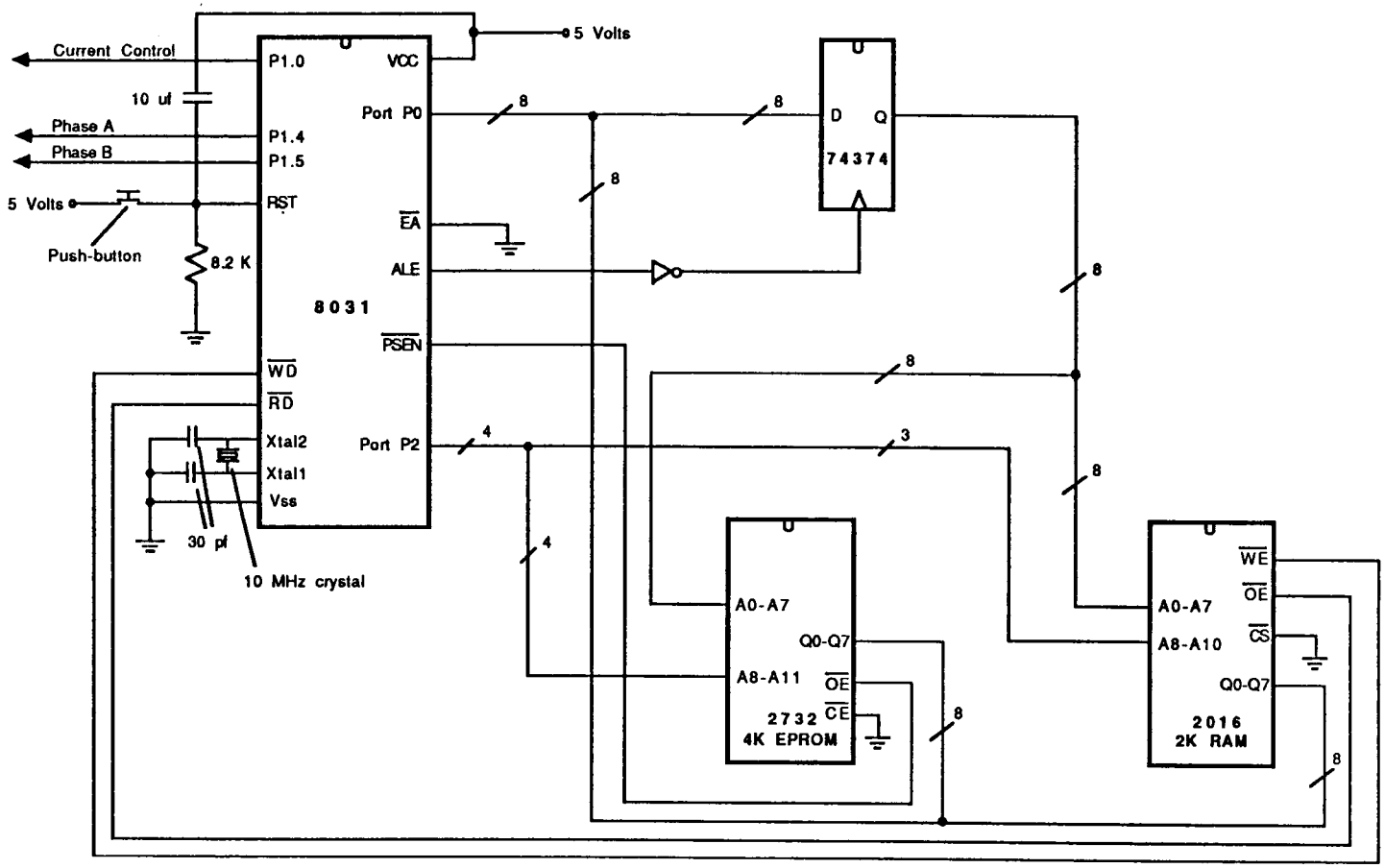


Figure 2.6 Circuit Diagram of Hardwares and Interfaces of the Microcontroller

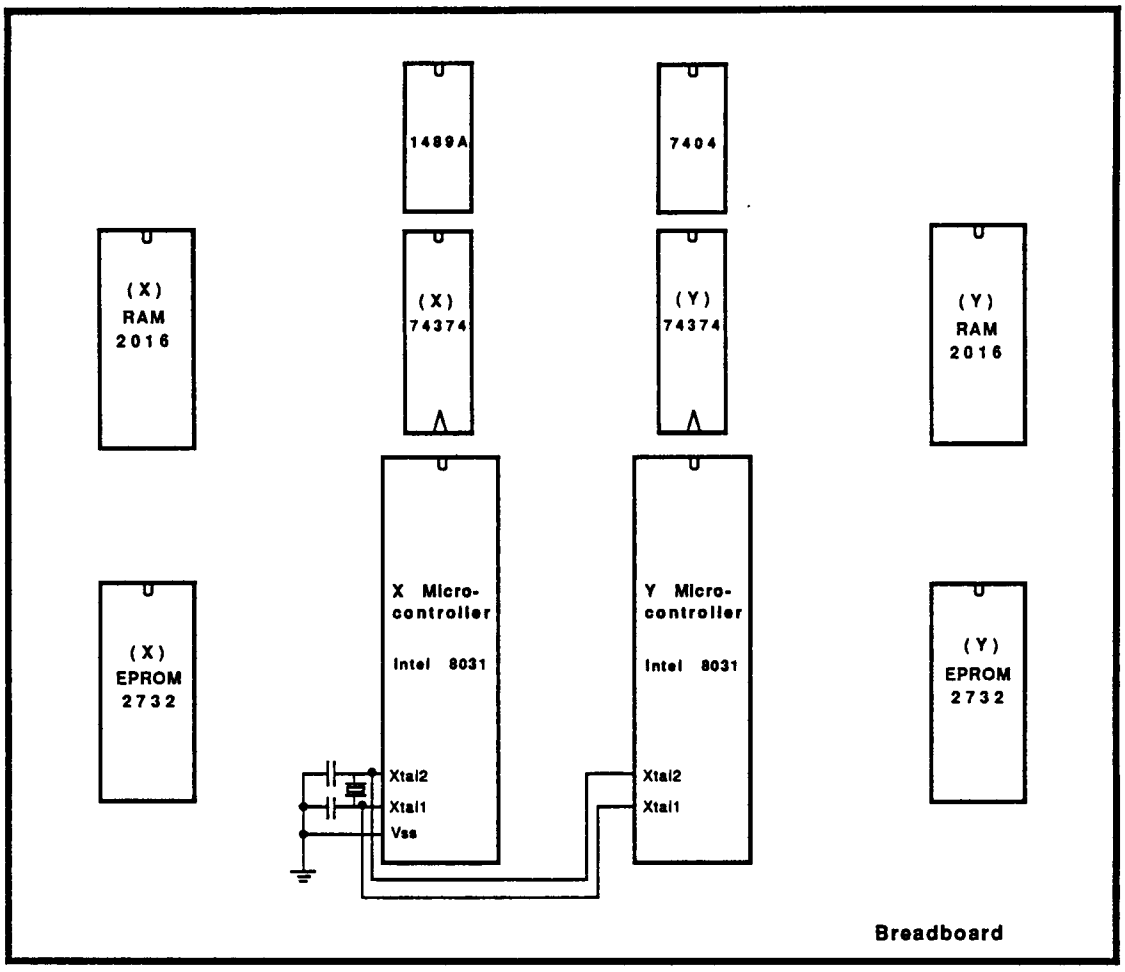


Figure 2.7 Hardware Lay-Out of the X and Y Microcontrollers

CHAPTER 3

SOFTWARE DESIGN OF THE ROBOT CONTROL SYSTEM

The software design is the core of this thesis. Programs had to be developed for both the main controller and the microcontrollers. For the main controller there is a wide choice of language to use as the IBM PC/AT is a very popular machine of which most if not all programming languages are supported. Since the function of the main controller involves communications, a language that allows that feature to be easily programmed would be the best choice. The programming on the main controller was also expected to involve assembly language, but as much of a higher level language as possible was used.

Three higher-level languages were considered and used at one time or another. They were FIFTH (a version of FORTH), BASIC, and C. FIFTH, which is a more user-friendly version of FORTH, was the first language to be used. However, many problems were encountered with FIFTH as it was a recently created language without a proper manual. The language still had bugs in it, and the so-call manual was not entirely accurate. There were also very few books on FORTH (which is the only way to learn FIFTH), and not too many people were familiar with this language. The version of FIFTH purchased is not the "professional" version and did not have some of the capabilities to access the peripherals and hardwares of the PC which FORTH has. (FORTH was tried, but it was decided that the costs of the software would be too high.) BASIC, which was free and was much preferred by IBM, was then used.

BASIC was extremely easy to use and the number of books written on it makes it an unbeatable choice. The BASIC interpreter also made programming and debugging less time consuming. The Turbo C version of the ever popular C language was also acquired. It was intended that both C and BASIC be used to do the same job, and from their performance, program size, and program flexibility (like making additions and changes to program), determine one as the more suitable language to be used for writing the main controller program. Several simple program modules in C were written during the learning process. Unfortunately, time did not permit the C program to be developed. Therefore, BASIC was the final and only high level language used.

Programming also involves the microcontrollers. The MCS-51 assembly language, which is the machine language of the 8031, was used. Unfortunately, an MCS-51 development system was not available. All the 8031 programs were wirtten and assembled on the IBM PC/AT. An MCS-51 cross-assembler was used on the PC to assemble the 8031 programs. To test an 8031 program, the assembled codes had to be programmed into an EPROM and tested on the prototype (usually the 1.1 amp stepper-motor and its single-chip motor driver were used). Due to the lack of the MCS-51 development system, the EPROM had to be erased and reprogrammed constantly.

3.1 Types of Data and Instructions for the Microcontrollers

Before getting into the programming, it is necessary to explain what the different data and instruction types are and also how the different data types are organized. There are three types of data that have to be downloaded into the microcontrollers. They are "ramp" data (which consists of two kinds - "ramp-up" and "ramp-down"), "move" data, and "robot" data (the number of repetitions).

The "ramp-up" data is actually a table of converted stepper-motor delay times. A

stepper-motor delay time is the time in-between the motor step advancements. The "ramp-up" data therefore contains delay times that decreases in value, causing the angular velocity of the motor to increase as it steps. Conversely, the "ramp-down" data contains delay times that increases in value, causing the angular velocity of the motor to decrease as it steps. There is only one set of "ramp-up" data and one set of "ramp-down" data for both the X and Y microcontrollers This is because the mechanisms used for both the X and Y directions were identical (see Figure 1.1). The "ramp-up" and "ramp-down" data were converted from the acceleration and deceleration profiles of the 2.1 amp stepper-motor using the program named RC (the program is discussed later in this chapter). The acceleration and deceleration profiles used were obtained by the mechanical engineering graduate student.

The 8031 is an eight-bit machine. Stepper-delay times are typically in the range of ten milliseconds to 400 microseconds. Due to the constrains of the microcontrollers, it was determinded that the resolution of the stepper-motor delay time can best be about ten microseconds. Therefore, the "ramp" data also has that ten microseconds resolution. The number of possible "ramp" data is therefore computed as follows:

$$\text{Number of possible "ramp" data} = (10\ \text{msec} - 400\ \text{usec}) / 10\ \text{usec}$$
$$= 960$$

Because a byte can only handle 256 different delay times, two bytes (a high byte and a low byte) had to be used to represent all the possible "ramp" data.

The "move" data are data used by the microcontrollers to move the control tubes **p** and **q** (see Figure 1.1) from one point to another, and in the process carries the Z-arm from one point to another. Since the control tubes for the X and Y directions move independently of each other, their "move" data are therefore different. However, their

"move" data format is the same. Each "move" data contains the following informations:

a. Direction-of-move/no-move/end-of-move (1 byte)
b. Total number of ramp-up steps (2 bytes)
c. Total number of constant-speed steps (2 bytes)
d. Total numb- - of ramp-down steps (1 byte)
e. Idle time (1 byte)

The direction-of-move/no-move/end-of-move data is a byte of data that the X and Y microcontrollers use to find out whether they need to step their stepper-motors or not, and if they do the direction to step the stepper-motors (clockwise or counter-clockwise). If this byte of the X microcontroller contains a **one**, the X microcontroller will know that the stepper-motors A and C are to be stepped in the clockwise direction causing the control tube **p** to move in the positive direction (away from the home position) of the X-axis. A **two** in this byte does the reverse. The stepper-motors A and C will be stepped in the counter-clockwise direction, causing the control tube **p** to move in the negative direction (towards the home position) of the X-axis. A **three** in this byte of the X microcontroller implies that a move along the X-axis is not required. All the X microcontroller has to do, if this byte is a **three**, is to wait for the Y microcontroller to get done with its move. This no-move feature allows the Z-arm to move along the other axis (which is the Y-axis) to get to a certain position, while this microcontroller (the X microcontroller) waits for it to get there, so that the next task or move can be accomplished in proper synchronisation. There can be one or more "move" data in a "move" data set. The last data of a "move" data set is always a **zero** and it will appear in this byte (i.e. in the direction-of-move/no-move/end-of-move byte). A **zero** in this byte means that all the required moves have been executed.

The ramp-up steps data consist of two bytes. The microcontroller interprets the high byte as multiple of a 100 steps and the low byte as unit steps. Therefore, if the high byte contains three and the low byte contains 75, the total number of ramp-up steps is 375. The ramp-up steps corresponds with the "ramp-up" data set. If the ramp-up steps is 375, than the stepper-motor will be stepped until it reaches the 375th "ramp-up" data. The number of ramp-up steps should therefore never be greater than the number of "ramp-up" data. The ramp-up steps will also bring the stepper-motor to aconstant speed.

The constant-speed steps data also consist of two bytes and is interpreted the same way ramp-up steps data are. Therefore, the maximum number of constant speed steps is 25,600 (255 x 100 + 100). The time delay used for constant-speed stepping is always that of the last ramp-up step.

Owing to friction, the ramp-down process requires less steps. For both the X and Y directions, it took less than 255 steps to ramp-down to a halt from the maximum speed. Therefore, the ramp-down steps data can fit into a byte. The microcontrollers need this ramp-down steps data to compute which "ramp-down" data to use to start the ramp-down, so that the ramp-down process runs smoothly and always end at the last "ramp-down" data. Figure 3.1 shows a rough sketch of the ramp-up, constant-speed, and ramp-down segments of a typical stepper-motor move.

The last "move" data is the idle time requested by the user. This is incorporated so that the robot can be synchronised with the outside world, like waiting for the parts to be assembled to come down the assembly line, etc.

Each "move" data therefore consist of seven bytes. In order for the robot to perform a complete task, like go to point A to pick something to be placed at point B and so on, several "move" data are grouped together to form a single "move" data set.

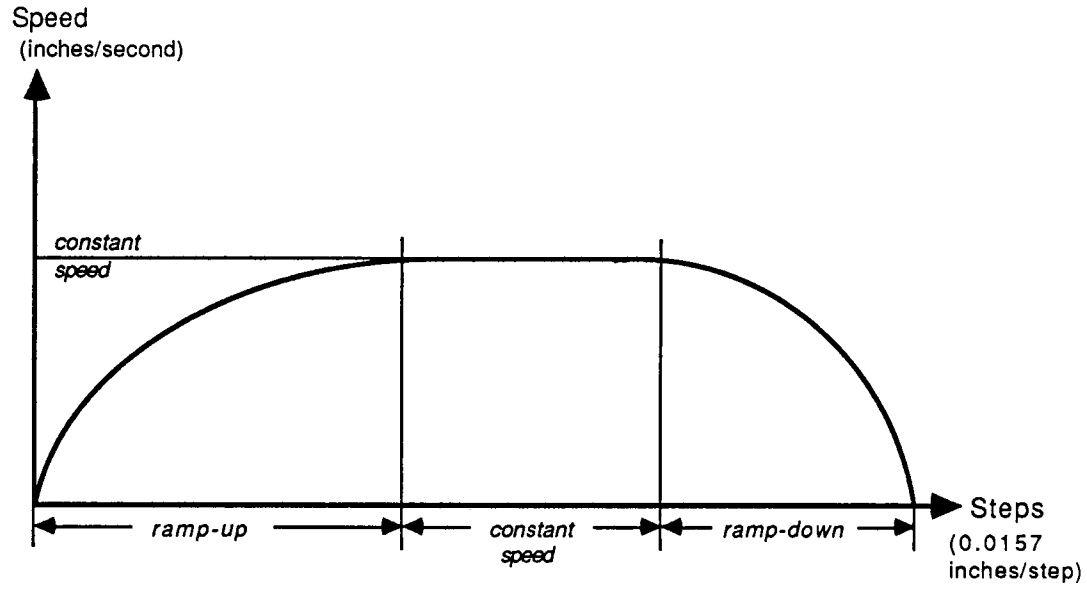


Figure 3.1 Rough Sketch of the Segments Involved in a Typical Stepper-Motor Move

The "robot" data is actually the number of repetitions. The microcontrollers use these data to determine how many times to repeat a task or, in other words, how many times to repeat the moves in a "move" data set. Since, the "robot" data is only a byte in size, there can only be a maximum of 255 repetitions.

All these data ("ramp-up", "ramp-down", "move", and "robot") are stored in the static RAM once they are received from the main controller. In fact, the "ramp-up" and "ramp-down" data could have been in the EPROM (as discussed in chapter 2), because they remains the same. The "move" and "robot" data will have to be reloaded if a different task is required or if the number of repetitions to perform a task is to be changed.

Instructions are also downloaded from the main controller to the two microcontrollers. They differ from data in that they are not stored in the static RAM once they are received. Only two types of instructions were used - execution and reset. The execution instruction is used to instruct the microcontrollers to start executing their moves. Therefore, it is imperative that all the data types are already downloaded and stored in the static RAM of the microcontrollers before the execution instruction is sent.

Initially the reset instruction was used to initialize the robot to a known position called the home position without the use of sensors. The home position, based on the X-Y coordinates, is at point (0,0). This automatic reset was achieved by letting the X and Y microcontrollers step the stepper-motor slowly at half current in the counter-clockwise direction until it physically encounters an obstacle - the corner bracket at (0,0). In order for the robot to be initialized from any unknown position, the total number of steps used was based on the assumption that the Z-arm is at the furthest possible position from the home position. This presents a problem when the Z-arm is not at the furthest position. This is because the stepper-motors will continue to step at half current until all the steps are done, even the though Z-arm is already at the home position. There were no bushings used to reduce the impact between the corner and the slider brackets. Therefore, the corner bracket at location (0,0) and the slider brackets e and f could easily be damaged, not to mention the stepper-motors themselves. This is definitely a poor way to re-initialize the robot without the use of sensors. Hence, the reset feature of the microcontroller programs was taken out and put on hold

3.2 Robot Control Program for the Main Controller

The robot control program on the main controller was written in BASIC (with assembly language routines) and is called ROBOT3. This program is used to transmit

data and instructions (execute and reset instructions) to the X or Y microcontrollers. Since, the robot will eventually be operating in an assembly line by a factory worker, the program was designed to be user-friendly requiring no prior knowledge of programming by the user (as long as the user understands English and knows how to "hit" the keyboard). Simple windows and menus were used to prompt the user. The user responds by typing specific keys, and if the user makes a mistake by hitting an invalid key, the speaker in the PC will "beep" to notify the user of the mistake.

In writing this program, there is a need to take the 8031 microcontrollers into consideration. In order for the 8031 to use its multiprocessor communications feature (mode 3), data has to be in the 9-bit format. The lower eight bits consist of the actual byte of data while the ninth bit is a sort of addressing bit. A one in the ninth bit causes a serial communications interrupt to occur in all the 8031s, while a 0 in the ninth bit does not. In general, the lower eight bits of data will be an address code if the ninth bit is a one. The details of how the 8031s handle the incoming data, discerning address byte from data byte, etc. is discussed in section 3.4.

In transmitting data to the microcontrollers, the main controller must include information on the address of the 8031 to which the data is to be downloaded. This address is in the high nibble of the lower eight bits of the very first nine-bit data to be sent. A nibble is sufficient for the address code because the program was written to have the capability to address three 8031s (the X, Y, and Z microcontrollers). Since there are three different data types (as discussed in section 3.1), the low nibble of this byte was used to inform the microcontrollers what data types are being transmitted. The term Address Code will be used to describe the information this first byte of data gives. The Address Code used for the X, Y, and Z microcontrollers are as follows:

DATA TYPE	ADDRESS CODES IN HEX		
	X	Y	Z
"ramp"	15	25	45
"move"	16	26	46
"robot"	13	23	43

In general, the second byte transmitted is the data itself. However, there are two kind of "ramp" data - "ramp-up" and "ramp-down" - and the microcontrollers need to know which one it is receiving. Therefore, the second byte is used to contain this information. It is convenient to call this second byte the Data Code byte. The Data Code byte is also used when transmitting "move" data as there is a future possibility that several "move" data sets (different robot tasks) might be kept in the static RAM of the microcontroller. As for the "robot" data transmission, the second byte itself is the data (i.e. the number of repetitions).

Instructions are also transmitted to the microcontrollers. Instructions always affect all the microcontrollers. Although two instructions were used at the initial stage of this project, only the execute instruction remained active. The reset instruction, although still available, does not require the microcontrollers to take any action. The execute instruction, which is merely the first btye of data itself (with a one in the ninth bit), is used to interrupt all the microcontrollers to let them know that they are to start executing their moves.

The main controller program, Robot3, always expects data to be in a disk file except for the "robot" data which is to be entered by the user when asked. The user has to give the name of the disk file that contains the data that is to be transmitted. Robot3 then opens the file and places the data in an array. Data is then transmitted one byte at a time from that array, right after the Address Code byte (first byte) and Data Code byte

(second byte). At the end of the transmission the screen of the main controller (the PC) will inform the user of the Address Code and Data Code used. This permit the user to verify which of the microcontrollers is getting the data transmitted.

The only way data can be in the nine-bit format is to include the parity-bit. Although BASIC supports serial communications, it does not give the programmer the leisure of specifying the parity-bit if the data itself is more than seven bits (ASCII codes use seven bits). This does not mean that BASIC does not use the parity-bit during the actual transmission of eight-bit data. It just means that BASIC will not support parity check as a way of detecting error in data transmission/reception if the data has eight bits. To get around this problem a more flexible language had to be used and none is more flexible than the machine language itself. Machine or assembly language routines are therefore used to do the actual transmission of the nine-bit data. The BASIC program Robot3 was first compiled and then linked with the assembly language routines to produce an executable file, Robot3.exe.

The way the data transmission from the main controller to the X and Y microcontrollers is set up now, the user has to separately download the data even if the data is the same for both. For example, the "ramp-up" data, although similar for both the X and Y microcontrollers, can only be received by the microcontroller that matches the Address Code in the first byte.

3.3 Assembly Language Routines for Serial Communications

The assembly language routines were written to control the serial port (COM1) of the serial/parallel communication adapter. The function 0 Hex of the BIOS (Basic I/O System) interrupt function 14 Hex was used to initialize the COM1 port to 1200 baud, *appropriate* parity, one stop bit, and eight data bits.

As discussed earlier, the parity bit (of which the 8031s see as the ninth-bit) is extremely important. The parity bit should only be a one when it is going to interrupt all the 8031s. Hence, only the first byte which contains the Address Code (see section 3.2) and instructions (execution and reset) can have a one in the parity bit. This is done by deliberately making all Address Codes and instructions odd-parity, and initializing the COM1 port to transmit data using even-parity, resulting in a parity bit (or ninth-bit) of one.

In order to avoid interrupting the unaddressed microcontrollers, the ninth-bit associated with the Data Codes and data must be a zero. Since, it was highly impratical to deliberately make all Data Codes and data (especially) even-parity and initializing COM1 port to transmit them using even-parity so as to get a zero in the ninth-bit, a more suitable solution was used. In order to set the COM1 port to always transmit a parity bit of zero, the parity of the data (and Data Codes) had to be determined first. The microprocessor of the IBM PC/AT ,which is the Intel 80286, features a parity flag, PF, in its flag register that is affected whenever an arithmetical or logical operation is being performed. This flag is set if the low-order byte contains an even number of ones resulting from the operation. Therefore, each data (or Data Code) is first logically ORed with a Hexadecimal value FF to to get the parity flag set accordingly. If the parity flag is set, the COM1 port will be initialize with even-parity. COM1 port will be initialize to transmit data with odd-parity if PF is not set. In this way, data will always be transmitted with a parity bit of zero.

The IBM PC/AT is a very fast machine, and not all the hardware peripherals it uses can handle its speed. The COM1 port which uses the National Semiconductor 8250 UART is one example and this was found out the "hard way". The COM1 port had to be reinitialized everytime data was to be transmitted so that the parity bit can be

set correctly. As discussed earlier, only instructions or the Address Code byte should interrupt all the microcontrollers. Unfortunately, the transmission of data was also interrupting all the microcontrollers putting the multiprocessor serial communication protocol in disarray. The BASIC program, assembly language routines, and the 8031 programs were thoroughly checked for "bugs" and none were fc_nd. Finally, the data coming off the transmission line was observed using a logic analyzer. It was found from the logic analyzer display that the eight bits of data themselves were being transmitted corrrectly. But, the parity bit (or ninth-bit) was not always zero when data (or Data Codes) were transmitted. This indicated that the COM1 port must not have been re-initialized properly, putting parity bits of one at certain times when they should have always been zero when data (or Data Codes) are being transmitted. A delay was then inserted in-between data transmissions to see if it made any difference. With a "long enough" delay inserted, the parity bits transmitted were oberved to be correct. With the help of the logic analyzer, it was determined that the delay had to be at least 9 milliseconds. It was therefore concluded that the COM1 needed about 9 milliseconds for each initialization to be completed - which is very slow as far as the PC is concern.

3.4 Program for the X and Y Microcontrollers

The machine language of the 8031 is known as MCS-51 Assembly Language. The programs for X and Y microcontrollers were developed and assembled on the IBM PC/AT. Their codes are then transfered into two separate EPROMs to be used as the external program memory for the X and Y microcontrollers. The programs for the X microcontroller, called ROX, and the Y microcontroller, called ROY, are the same except for their Address Codes. The program ROY can be found in appendix I. The discussion in this section is therefore applicable to either microcontroller.

The main routine of the 8031 program is used to set the stage for the reception of data and instructions using multiprocessor serial communications mode 3, to set the timer0 to auto-reload with a count-down time of 0.1 millisecond to be used in the timing delays for the stepper-motors, to set the timer1 for 1200 baud, to enable the serial port interrupt, and also to initialize the stepper-motors with a known phase pattern at half current. When a serial port interrupt is generated by the first byte of a data stream or by an instruction, it is serviced by the interrupt service routine. The first thing done by the interrupt service routine is to fetch the eight bits of data (which is either an Address Code or an instruction code), and check to see if it is a reset instruction code, an execute instruction code or an Address Code. The simplified flowchart in Figure 3.2 shows the "main routine" of the interrupt service routine (ISR). A reset requires no action (the initial implementation was discarded). An execute instruction causes the microcontroller to start stepping the stepper-motors it controls.

If it is an Address Code, the microcontroller will have to use it to check and make sure that it is being addressed. If it is not, it simply exits the interrupt service routine and waits for another serial port interrupt to occur. If the microcontroller is being addressed, it goes into a subroutine (the simplified flowchart is shown in Figure 3.3 below) to commence receiving data. First, the subroutine needs to clear bit SM2 of the SCON (Serial Port Control) register to allow for the RI (Receive Interrupt) flag to be set when the received ninth bit is zero. For the microcontrollers that are not addressed (i.e. their addresses do not match the Address Code), their SM2 bits remained set. Therefore, the microcontrollers that are not addressed will be unable to set their RI flags when the ninth bit of the data they received is a zero. In this way, the microcontrollers can tell, by checking the RI flag, that a data byte has arrived at the serial port. This is how the multiprocessor serial communications of the 8031 works.

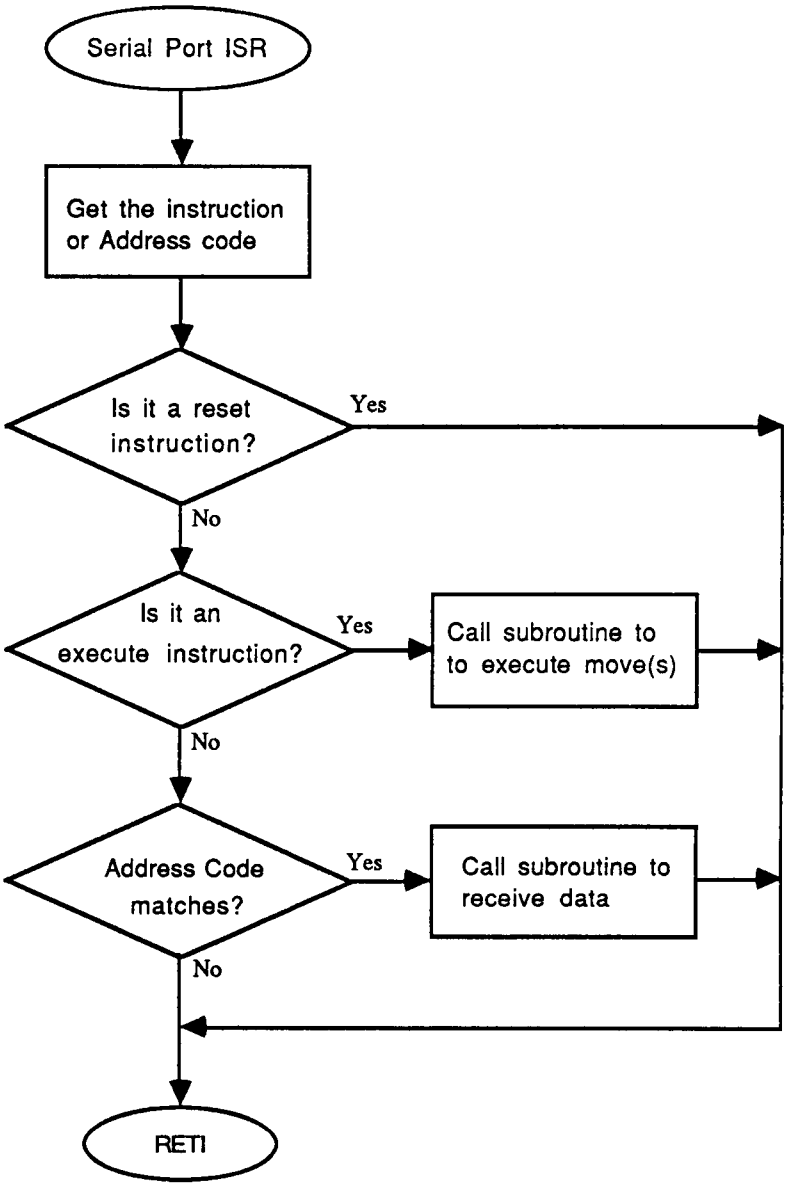


Figure 3.2. Simplified flowchart of Interrupt Service Routine (ISR) for the Microcontroller

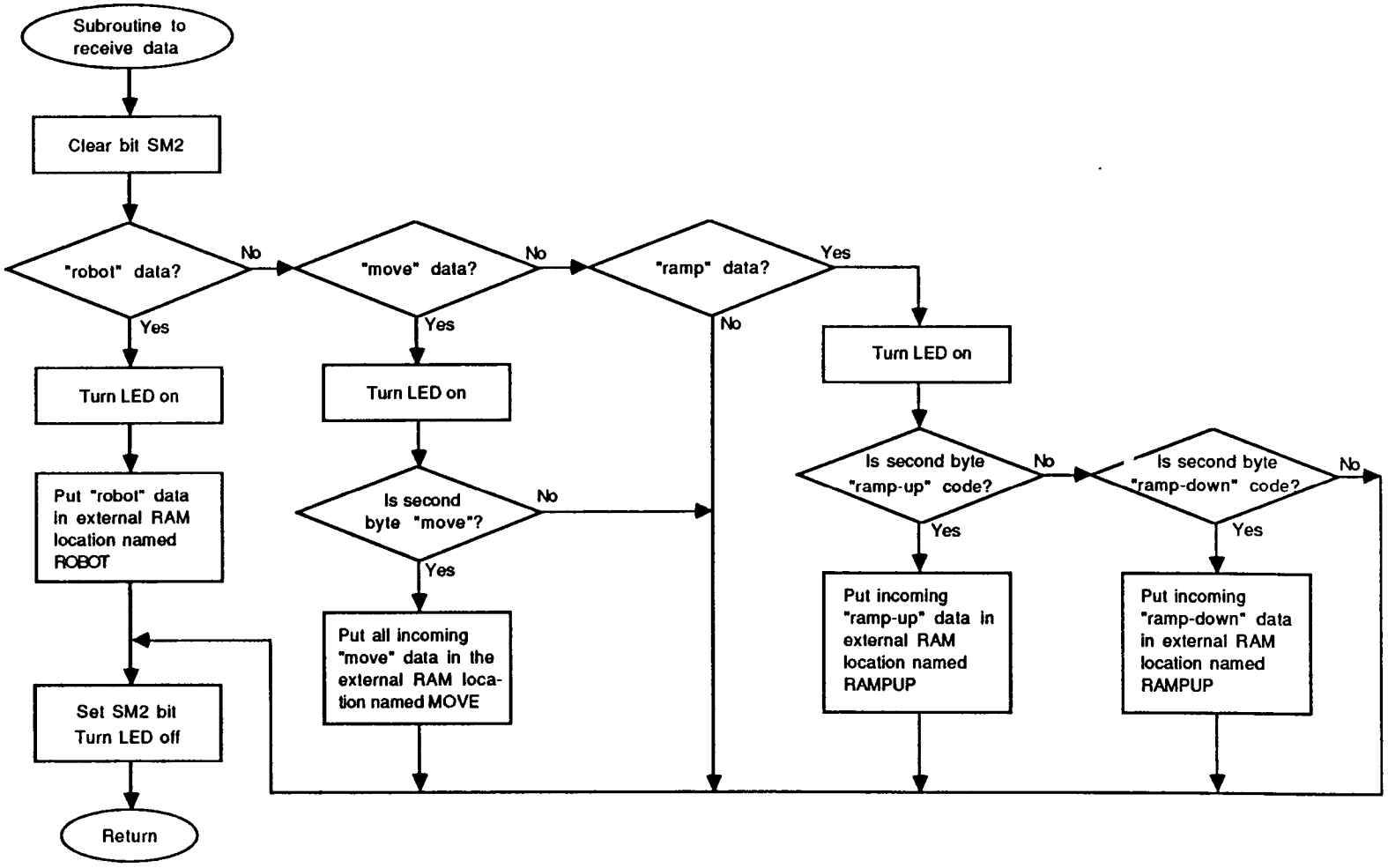


Figure 3.3. Simplified flowchart of Subroutine to Receive Incoming Data

The Address Code also contains information on the type of data the microcontroller is going to receive so that the data can be stored in the correct RAM location or address after they arrived. Since there are two kinds of "ramp" data, "ramp-up" data and "ramp-down" data, the microcontroller needs more information to tell which it is receiving. Hence, the second byte associated with the "ramp" data stream is used to contain that information. For the "move" data type, this method is also used. This is actually not necessary because there is at present only one kind of "move" data set. However, it is incorporated so that a different "move" data set can be used if needed or desired at a later time. For "robot" data, the second byte is not a Data Code but rather the number of repetitions itself.

When data is being received by the addressed microcontroller, the RI flag is automatically set by hardware. The subroutine then clears the RI flag and goes on to load the received data into its approriate static RAM location. No error detection or any type of checksum is performed on the data received. Also the microcontroller does not know before-hand how many data bytes it is going to receive. In order for the microcontroller to know that all the data have arrived, a subroutine is used to monitor the RI flag every 0.4 milliseconds for a time period of about 50 msec (note that it takes only about 10 milliseconds in-between data transmissions). If that time runs out and the RI flag is still not set, it quits reception of data by setting bit SM2 of the SCON register and jumps out of the interrupt service routine.

The microcontroller is also programmed to turn on an LED via I/O port P3.5 when it is receiving data so that the user can see that the data transmitted from the main controller is being received by the correct microcontroller(s). This LED is turned off at the end of the data reception subroutine.

As it is shown in Figure 3.2, the serial-port interrupt service routine has to call a

subroutine that executes or moves the robot if the first byte that it receives from the main controller is an execute instruction. For the robot (or the Z-arm) to move, the stepper-motors would have to be stepped. The stepper-motors that are used have four phases or windings each. To make a stepper-motor step, these four phases have to be energized (1) and de-energized (0) properly. Figure 3.4 shows how these four phases must be energized or de-energized to make a stepper-motor turn in the clockwise direction. Figure 3.5 shows the sequence for energizing (1) and de-energizing (0) the four phases to make the stepper-motor step in the counter-clockwise direction.

When the modified Proprinter is used to drive the stepper-motor, only phases A and B need to be directly controlled by the microcontroller. The phases for A and B, which are complements of A and B, are generated by the Proprinter board. Two phase-pattern tables, which are derived from Figure 3.4 (for the clockwise direction) and Figure 3.5 (for the counter-clockwise direction), are set up in the internal RAM of the microcontroller. A pointer is used to keep track of the phase pattern with which the stepper-motor is being energized. Phase A comes from I/O ports P1.4 and phase B from the I/O port P1.5. The current-control bit, which allows the stepper-motor's windings to be energized at either full current (2.0 amp) or half current (0.8 amp), is also incroporated in these two phase-pattern tables. The current-control bit is always a one in the phase-pattern tables so that the phases to be energized will be energized at full current. The I/O port P1.0 is used to send the current-control signal to the stepper-motor driver.

When executing a move, the first thing the microcontroller does is to fetch the seven bytes of "move" data. From the first byte, one of the following pieces of information is obtained - the direction of the move, no move, or the end of move. The next two bytes contain the total number of ramp-up steps needed in order to ramp-up to

$\overline{B}$	$\overline{A}$	B	A
1	1	0	0
0	1	1	0
0	0	1	1
1	0	0	1

Figure 3.4 Phase Patterns for Clockwise Step

$\overline{B}$	$\overline{A}$	B	A
0	0	1	1
0	1	1	0
1	1	0	0
1	0	0	1

Figure 3.5 Phase Patterns for Counter-Clockwise Step

the speed desired (by the user). This is done by using the number of ramp-up steps to know how far down the "ramp-up" data table (which is in stored in the static RAM) it needs to go. On reaching the last ramp-up step, the "ramp-up" data to be used will translate into the speed desired by the user. This same "ramp-up" data is to be used by the next two bytes of "move" data, which contains the number of constant speed steps required. Only this "ramp-up" data will be used by all the required constant speed steps, causing the stepper-motor to step at a constant speed. The sixth byte contains the number of ramp-down steps. In order to know which "ramp-down" data to start with, the microcontroller must first compute the address of the first "ramp-down" data to use. For example, if the user had chosen the maximum speed (35 inches per second), then the first ramp-down step will get its "ramp-down" data from the very beginning of the "ramp-down" data set location in the RAM. However, if the user used a speed that is lower than the maximum, the microcontroller needs to compute the start address, A_s, of the "ramp-down" data it needs to use by subtracting the number of ramp-down steps, **R**, from the location or address of the last "ramp-down" data, A_L. Therefore, the start address is:

$$A_s = R - A_L$$

By starting at the "ramp-down" address A_s, the last ramp-down step will end up using the last "ramp-down" data of the "ramp-down" data set. The seventh and last byte of information of a "move" data contains the idle time. The is the time for which the robot remains idle after reaching its destination.

At the end of the idle time the microcontroller pulls its I/O port P3.4 high and monitors its I/O port P3.3 waiting for it to go high. The I/O port P3.4 of the X microcontroller is linked to the I/O port P3.3 of the Y microcontroller, and the I/O port

P3.4 of the Y microcontroller is linked to the I/O port P3.3 of the X microcontroller. This is done so that the X and Y microcontrollers can synchronize themselves for the next move. Figure 3.6 shows how this is accomplished by the use of an AND gate when there are more than two microcontrollers in the robot system.

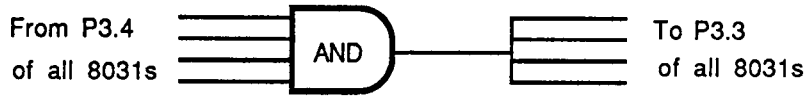


Figure 3.6 Using an AND Gate to Synchronize Moves When More Than Two Microcontrollers are Used

The last data of a "move" data set is always a zero. This last "move" data, a zero, is used to let the microcontroller know that all the moves coded in the "move" data set has been executed. The program than checks the "robot" data to see if there is any repetition of the moves required. If so, it decrements the "robot" data (or number of repetitions) by one and goes on to execute the moves again. This continues until the "robot" data is decremented to zero.

The X and Y microcontrollers used a 10-MHz crystal as the oscillator. Each machine cycle, which consists of 12 oscillator periods, is therefore 1.2 microseconds. Since, the "ramp" data (time delay between steps) is extremely critical, the time used up

by all the machine language instructions that will affect the "ramp" data has to be computed and taken into consideration. The NOP (no operation) instruction was also frequently used to make the timing work out correct. This is necessary because the control of the timing delay and phase pattern of the stepper-motor steps involves several stages. Each stage was broken down into a different routine. To get the timing right some of these routines had NOPs added to them. The timing overhead (extra time needed) of the routines involved in the control of the timing was found to be about 90 microseconds. Therefore, the "ramp" data being used are actually 90 microseconds less than the actual step delay times. Section 3.7 explains how this 90 microcseconds overhead is being taken into account when the "ramp-up" and "ramp-down" data set were being created by the BASIC program RC.

3.5 Programming of the EPROM

After the microcontroller program was assembled on the IBM PC/AT by the MCS-51 cross-assembler, the object codes had to be programmed into an EPROM to be used as the external program of the microcontroller. An EPROM programmer was used (on the IBM PC/AT) to program the 2732 EPROM. However, there were some initial problems involved with the programming of the EPROM. The cross-assembler was adding some extra codes (possibly program name codes and directive codes) into different sections of the program's object codes. Therefore, in order to program the EPROM correctly, the so-called object codes of the microcontroller program had to be loaded into the memory buffer of the EPROM programmer and compared with the listing of the assembled program. The EPROM would then be programmed in stages, making sure that the extra codes were not programmed into the EPROM. It has not yet been determined why the extra codes were added by the cross-assembler. However, it

is suspected that they could be there so that an Intel MCS-51 development system could interface properly with the IBM PC/AT. An MCS-51 development system was not available for this project.

3.6 Program to Create the "move" Data Set

This program is written so that a user who plans to create moves or "tasks" for the robot can do so with ease (i.e. not having to know the technical details of the robot). This user-friendly program was written in BASIC and is named MOVE.

MOVE is set up based on the X-Y Cartesian coordinates. The coordinates are measured in inches. The home position (for the Z-arm) is at (0,0). The furthest point the Z-arm can go from the home position is (41,41). With the pulley used on the stepper-motor shaft, each step of the stepper-motor translates into about 0.0157 inches. At the start of the program, the disk files containing the "ramp-up" data and "ramp-down" data of the stepper-motor are opened and their data loaded into two separate arrays. This is needed in order to compute the ramp-up steps, constant speed steps, and ramp-down steps based on the speed (for the robot's move) given by the user. The user will first be prompted to enter separately the initial X and Y coordinates (or the Z-arm coordinates) in inches. Then, the user will be required to enter the final X and Y coordinates. Next, the idle time (the time in which the robot just sits idle after reaching its destination) for X and Y will have to be entered. Finally, the speeds for the X and Y directions are specified by the user. These speeds are actuallly the constant speeds to be reached by the stepper-motors at the end of their ramping-up. Based on all this information the program will have to come out with the "move" data that must contain the following - direction-of-move/no-move/end-of-move code, the number of ramp-up steps, the number of constant speed steps, the number of ramp-down steps, and the

idle time.

The direction-of-move/no-move/end-of-move code is computed from the initial and final coordinates. The initial coordinate (X_i for the X axis) is subtracted from the final coodinate (X_f), i.e. $X_f - X_i$, and if the result is positive, it means that the stepper-motors should step in the clockwise direction to move the control tube away from the home position. The code used in this byte for the clockwise direction is **one**. A negative result means the stepping should be in the counterclockwise direction pulling the control tube towards the home position. **Two** is the code used for the counter-clockwise direction. The absolute value of the result is then divided by 0.0157 to convert the distance in inches into the number of required stepper-motor steps, **T**. If the result of $X_f - X_i$ is zero, it means that no move is required. **Three** is the code used for representing no-move. The end-of-move, which has a code of **zero**, is associated with the very last data of a "move" data set. So essentially all that needs to be done is to attach an extra zero at the very end of a "move" data set for this code.

The speed given by the user is used along with the "ramp-up" and "ramp-down" data in the two arrays to compute the number of ramp-up steps, number of constant speed steps, and the number of ramp-down steps. To compute the number of ramp-up steps for a particular move, the program converts the given speed into a step delay time and matches it with the step delay time of the "ramp-up" data array one at a time. When a match is found, that position of the array is the number of ramp-up steps required. If the number of ramp-up steps is **R**, then **R** is converted into two bytes of data of which the high byte is a mutiple of 100 steps and the low byte the unit steps. If the speed given by the user is found to be too high, meaning that the delay time converted from the given speed is smaller than the delay time of the last "ramp-up" data, the program simply quits after informing the user that the maximum speed allowed is 35 inches per

second. The number of ramp-down steps is to be computed next. The delay time converted from the given speed is matched with those delay times in the "ramp-down" data array. When a match occurs, the array position of the "ramp-down" data that matches the converted delay time is subtracted from the array position of the last "ramp-down" data. This gives the number of required ramp-down steps, **D**. To get the total number of constant speed steps, **C**, all it needs to do is to subtract the number of ramp-up steps, **R**, and the number of ramp down steps, **D**, from the total number of steps, i.e. **T - (R + D)**. If **C** is negative, then it knows that the speed chosen by the user for that move is too high, because it simply takes too many steps to ramp-up to that speed. The program therefore informs the user that he has selected an invalid speed, and all the user has to do is to enter a lower speed. This will go on until **C** works out to be a positve value or a zero. (A zero in **C** simply means that there is no constant-speed step.) Once a positive value of **C** is obtained, it is converted into a high byte and a low byte just like the way the ramp-up steps is represented. The last byte of the "move" data is the idle time which requires no computation as it comes straight from the user. These seven data bytes constitute one "move" data. More "move" data can be created the same way and the program automatically appends them to the first to form a single "move" data set.

After the user has programmed all the moves needed (to perform a certain task), the "SPACE" key can be used to save the "move" data on two separate disk files (one for the X and one for the Y microcontrollers). The "move" data created can also be viewed on the screen by typing the "ESC" key. This feature was included mainly as a debugging tool. The "move" data that is created will be "lost" if they are not saved in the disk files. That is why the program is written such that the user cannot quit the program without having to save it first (unless the user specified a speed which is

greater that 35 inches per second which automatically caused the program to stop and display message). While saving the "move" data in the disk files, the program always sticks an extra zero (which is transparent to the user) to be the last data of a "move" data set. This last data (a zero), when used by the microcontroller, will appear as the direction-c^ move/no-move/end-of-move data to let the microcontroller know that all the required moves have been performed. The size of the "move" data is restricted by the size of the external static RAM of the microcontroller. At the present, there can only be a maximum of 631 bytes in a "move" data set, giving a maximum of 90 moves (each "move" data consists of seven bytes, and each "move" data set has a byte added to it as the last data).

3.7 Program to Convert Acceleration and Deceleration Profiles to "ramp-up" and "ramp-down" Data

The acceleration profile is a table of delay times that will bring a stationary control tube (**p** or **q**) to its maximum possible (and practical) speed. A deceleration profile is a table of delay times that will bring a control tube moving at its maximum speed to a halt. These profiles are therefore required by the microcontrollers so that the stepper-motors they control can be properly brought to a maximum speed from a halt, or be brought to a halt from a maximum speed. In order for the microcontroller to use these delay times they must first be converted into "ramp" data.

The BASIC program called RC was written to do the conversion. The converted data are known as "ramp-up" data (for acceleration) and "ramp-down" data (for deceleration). As discussed and shown in section 3.1, these "ramp" data require two bytes each. As far as the microcontroller is concerned, the high byte of a "ramp" data represents a multiple of 0.1 millisecond, and the low byte represent a multiple of 0.01

millisecond (the resolution of a "ramp" data is 10 microseconds). Due to the timing overhead of the microcontroller program, the "ramp" data had to be adjusted to make the delay times work out right. The program RC therefore subtracts 90 microseconds (which is the overhead time of the microcontroller program) from the delay time entered by the user. The result is then rounded off to the nearest 10 microseconds. Next, by applying simple arithmetical operations, the result is converted into the two bytes of "ramp" data. Both the "ramp-up" and "ramp-down" data used were created by this program. Before a user quits the program, the "ramp-up" or "ramp-down" data must be saved in disk files.

All the BASIC programs written were compiled and linked to produce executable files. In this way, the programs execute much faster than when they are being executed in an interpreter environment.

CHAPTER 4

SUGGESTIONS FOR FUTURE IMPROVEMENTS AND ADDITIONS

This high-speed, low-cost industrial robot project does not end with this thesis. This project is still being actively pursued by IBM and the University of Tennessee, Knoxville. Therefore, this section could come in useful to future designer(s) of the control for the Z-arm (including the gripper, etc) of this robot.

One disadvantage of using the serial port interrupt service routine to send the execute instruction to start executing the robot's move(s) is that another serial port interrupt cannot occur until the routine being serviced is completed. This means that a manual reset (by pushing the reset push-button) has to be used to stop a malfunctioning robot. A better solution would be to use another interrupt of a higher priority (like INT0) to interrupt the robot and make it do a reset in the interrupt service routine of the interrupt (INT0). This interrupt (INT0) can be initiated from the IBM PC. However, the disdvantage is that it will require the presence of the user to detect the malfunction and to send the interrupt signal to the microcontrollers. This therefore require an extra line to carry the interrupt signal from the main controller to the microcontrollers. Optical encoders could be incorporated on the motor shaft to detect the "missing" of a step, but this essentially amounts to a closed-loop rather than an open-loop system.

The reset feature that was abandoned would have worked if the corner bracket at the home position (0,0) and slider brackets **e** and **f** had some kind of cushion on it (like rubber bushings). It would have worked because the stepper-motor would be

energized at only half current (to keep the motor torque low) and the speed would be set to be sufficiently low to avoid any major impact between the corner bracket and the slider brackets. A better method would involve the use of microswitches mounted on the corner bracket (0,0) in such a way as to cause a reset; that will freeze the stepper-motors when the slider brackets **e** and **f** come in contact with them. Hence, an impact between the slider brackets and the corner bracket can be prevented. Optical detectors can also be used to serve the same purpose.

Due to the problems experienced with programming the EPROM (with the correct codes and not directive codes) and debugging the program on the prototype, it is almost imperative that an MCS-51 development system be acquired. At the present, the "ramp-up", "ramp-down", and "robot" data have to be downloaded separately into the X and Y microcontrollers even though they are identical. This can be avoided by making the Address Code for the X and Y microcontrollers the same. However, the X and Y microcontrollers must still be able to tell who the incoming "move" data is for. This can be acomplished by making use of the second data byte, the Data Code byte. As discussed in section 3.4 this feature of differentiating "move" data is already incorporated. All that is needed is to establish different Data Codes for "move" data in the X and Y microcontrollers.

An even better solution would be to program the "ramp" data into the external program memory, the EPROM. This means that no "ramp" data whatsoever has to been downloaded from the main controller to the microcontrollers. This will make the start-up process of the robot system much simpler and much faster. Without putting the "ramp" data in the EPROM, a user can be expected to send six sets of "ramp-up" data, six sets of "ramp-down" data, six sets of "move" data, and also six sets of "robot" data to a final completed robot that has six degrees of freedom.

Although, there are three BASIC programs written to make the robot accessible to the user, two of them could have been put together to form a single program. The two programs are ROBOT3 and MOVE. In this way, the user need not switch program if a different set of "move" data has to be created for a new task. The static RAM size should definitely be expanded to allow for more moves, and maybe even for different sets of moves.

CHAPTER 5

RESULTS AND CONCLUSIONS

The robot has been tested extensively with the "move" data created by the program MOVE. The main reason was to find out if the robot was able to go to the same positions if the moves were repeated several times. The results indicated that the MOVE program used for creating the "move" data for the robot works. The robot was able to return to the same positions of the moves at every repetition.

The transition of the ramping-up, constant speed, and ramping-down stages in a move was found to be very smooth. Therefore, the program RC was to be able to convert the acceleration and deceleration profiles of the stepper-motor into "ramp" data correctly.

The data transmission from the main controller to the microcontrollers has not presented any more problem after the short delay was inserted between the initialization of the COM1 port of the PC (as discussed in section 3.3). No bad or erroneous data (due to noise problem) was found to be transmitted either.

This high-speed, low-cost industrial robot when completed, could be a great contribution to the assembly-line type industry. It can be easily dismantled (due to its simple design), transported to another location (due to its relatively light weight), and set-up in a short period of time. However, for such a robot to be cheap and simple the design for the electrical controls must also be simple. That is why an open-loop type control was used.

The use of the Intel 8031 microcomputers as microcontrollers not only helped keep the cost of the electrical hardware low, but also make the communications between

the main controller and the microcontrollers very simple. The RS-232C is a very standard port and is relatively noise-immune as its transmission line operates in the +12 to -12 voltage range. Although, a PC is required as the main controller, the PC is free to perform other tasks most of the time. This is because once the robot is started-up (i.e. "ramp" data, "move" data, and "robot" are already received and placed in the external RAM), all that is required is for the user to send the execute instruction (by hitting the "ENTER" key at the prompt of the main controller program), and the robot starts working on its own. Then, the main program can be exited so that the PC can be used for doing something else.

This project has been a great learning experience requiring work in robotics, electrical-to-mechanical interface, computer-to-computer interface and communications, hardware, and software. It also required an indepth study of the architectures of two entirely different microcomputer systems - the IBM PC/AT and the Intel 8031.

LIST OF REFERENCES

LIST OF REFERENCES

1. Microcontroller Handbook, Intel Corporation, 1986.

2. J. C. Cluley, Computer Interfacing and On-Line Operation, Crane, Russak and Company, Inc., 1975.

3. IBM Peripherals Technical Reference

4. Memory Components Handbook, Intel Corporation, 1986.

5. Microsystem Components Handbook, Microprocessors Volume I, Intel Corporation, 1986.

6. Murray Sargent III and Richard L. Shoemaker, The IBM PC from the Inside Out, Addison-Wesley, 1986.

7. MCS-51, Macro Assembler User's Guide, Intel Corporation.

8. Macro Assembler Version 2.00 (Reference), IBM Corporation, 1986

9. Steven Armburst and Ted Forgeson, Programmer's Reference Manual for IBM Personal Computers.

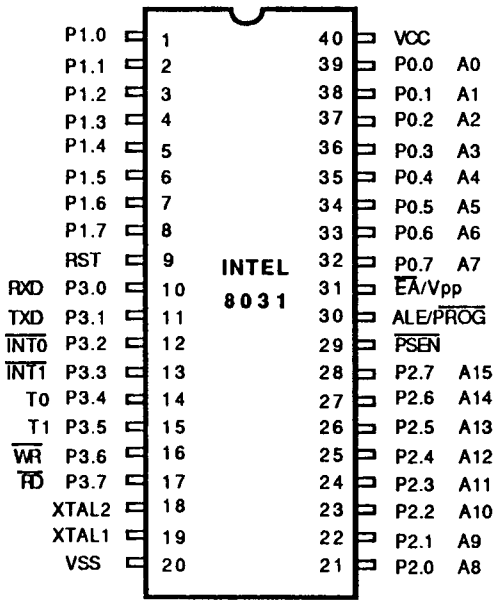
10. Takashi Kenjo, Stepping Motor and their Microprocessor Controls.

11. IBM BASIC reference, 1984.

APPENDICES

APPENDIX A

Pin-Outs and Special Function Registers of the Intel 8031



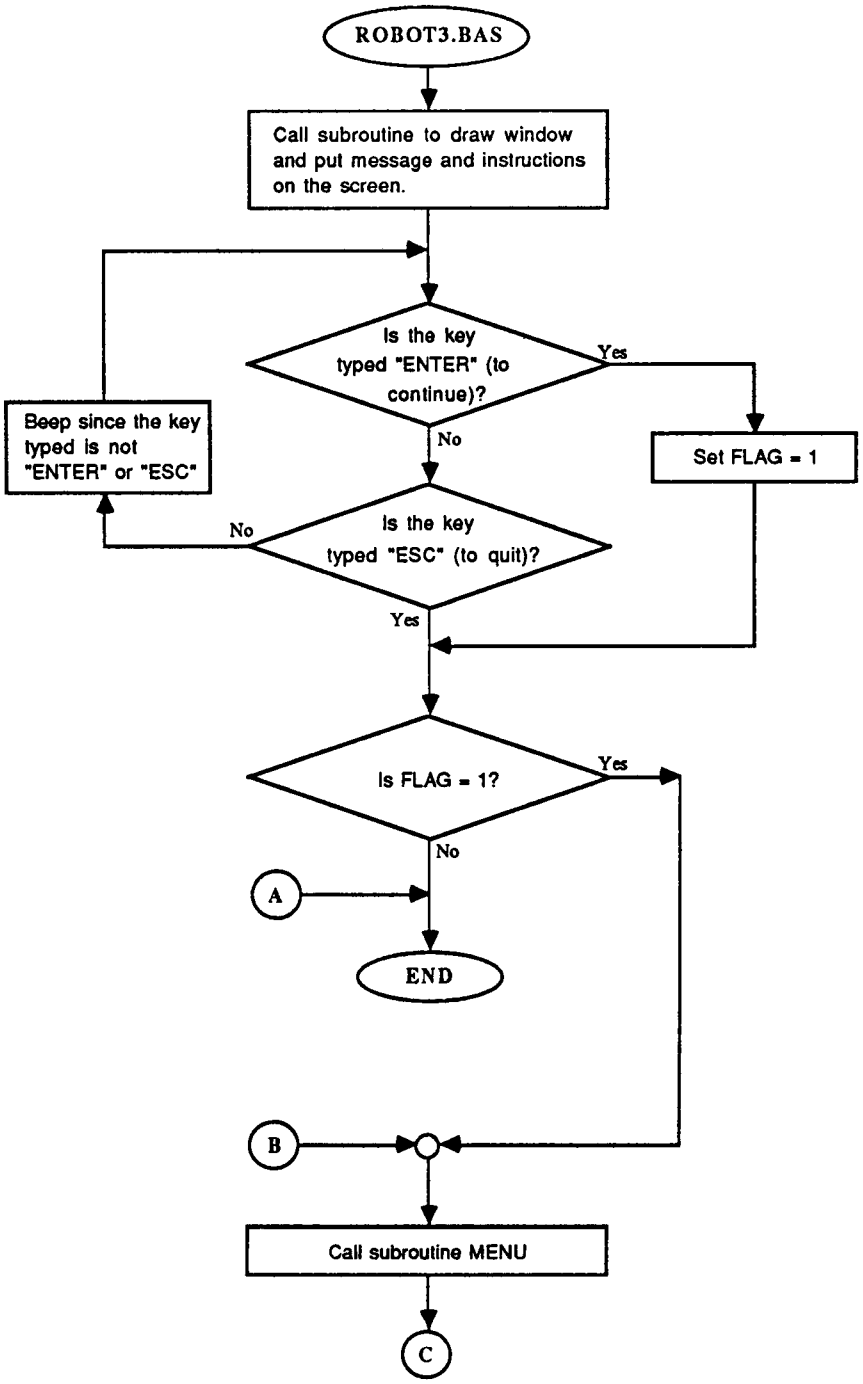
Pin-Outs for the Intel 8031

SYMBOL	NAME	ADDRESS
ACC	Accumulator	0E0H
B	B Register	0F0H
PSW	Program Status Word	0D0H
SP	Stack Pointer	081H
DPH	Data Pointer (High Byte)	083H
DPL	Data Pointer (Low Byte)	082H
P0	I/O Port 0	080H
P1	I/O Port 1	090H
P2	I/O Port 2	0A0H
P3	I/O Port 3	0B0H
IP	Interrupt Priority Control	0B8H
IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	089H
TCON	Timer/Counter Control	088H
TH0	Timer/Counter 0 (High Byte)	08CH
TL0	Timer/Counter 0 (Low Byte)	08AH
TH1	Timer/Counter 1 (High Byte)	08DH
TL1	Timer/Counter 1 (Low Byte)	08BH
SCON	Serial Control	098H
SBUF	Serial Data Buffer	099H
PCON	Power Control	087H

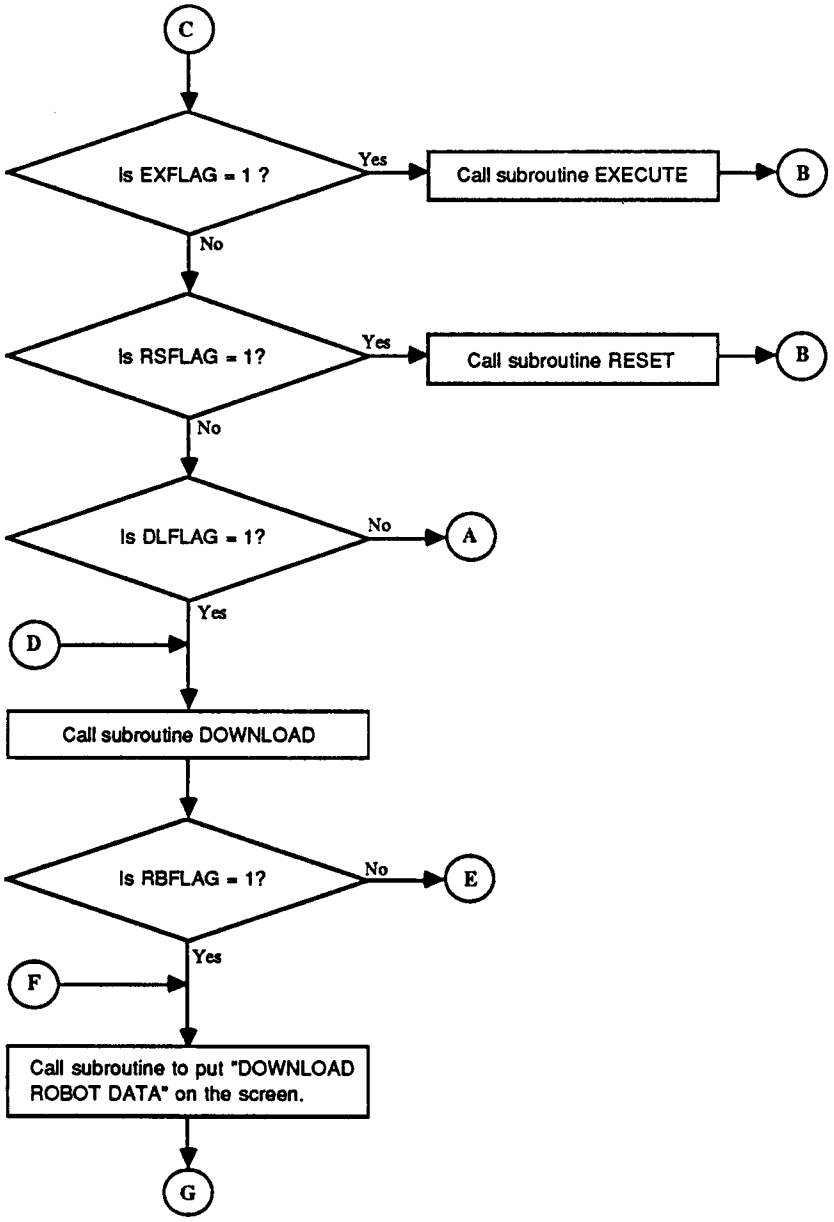
Special Function Registers of the Intel 8031

APPENDIX B

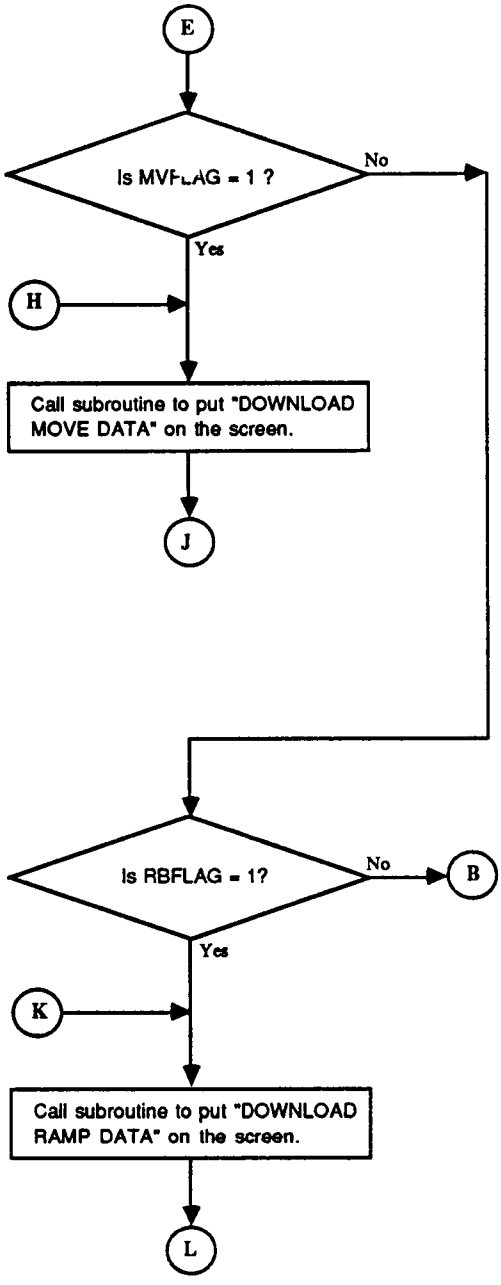
Flowcharts of the BASIC Program ROBOT3



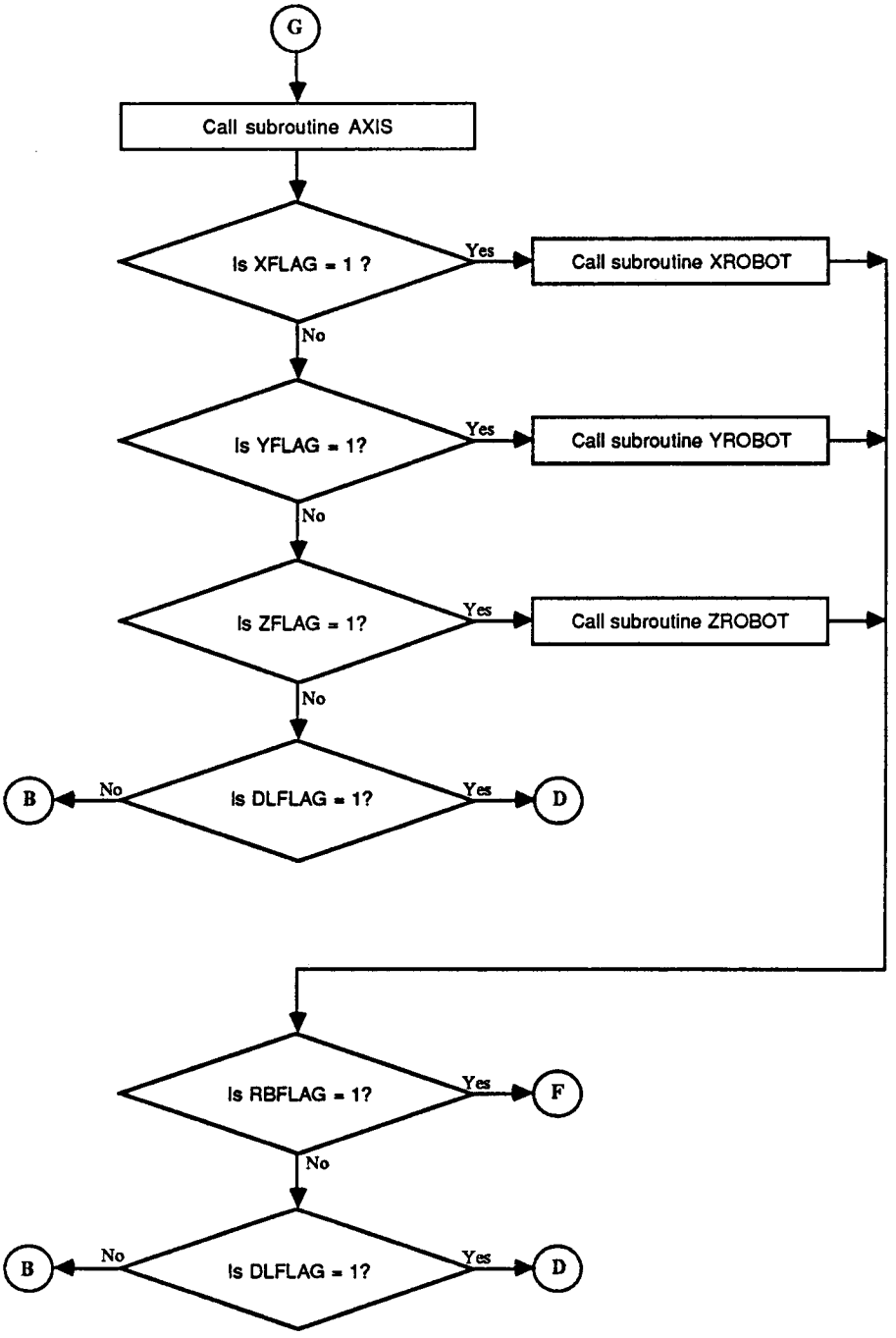
Flowchart of ROBOT3.BAS



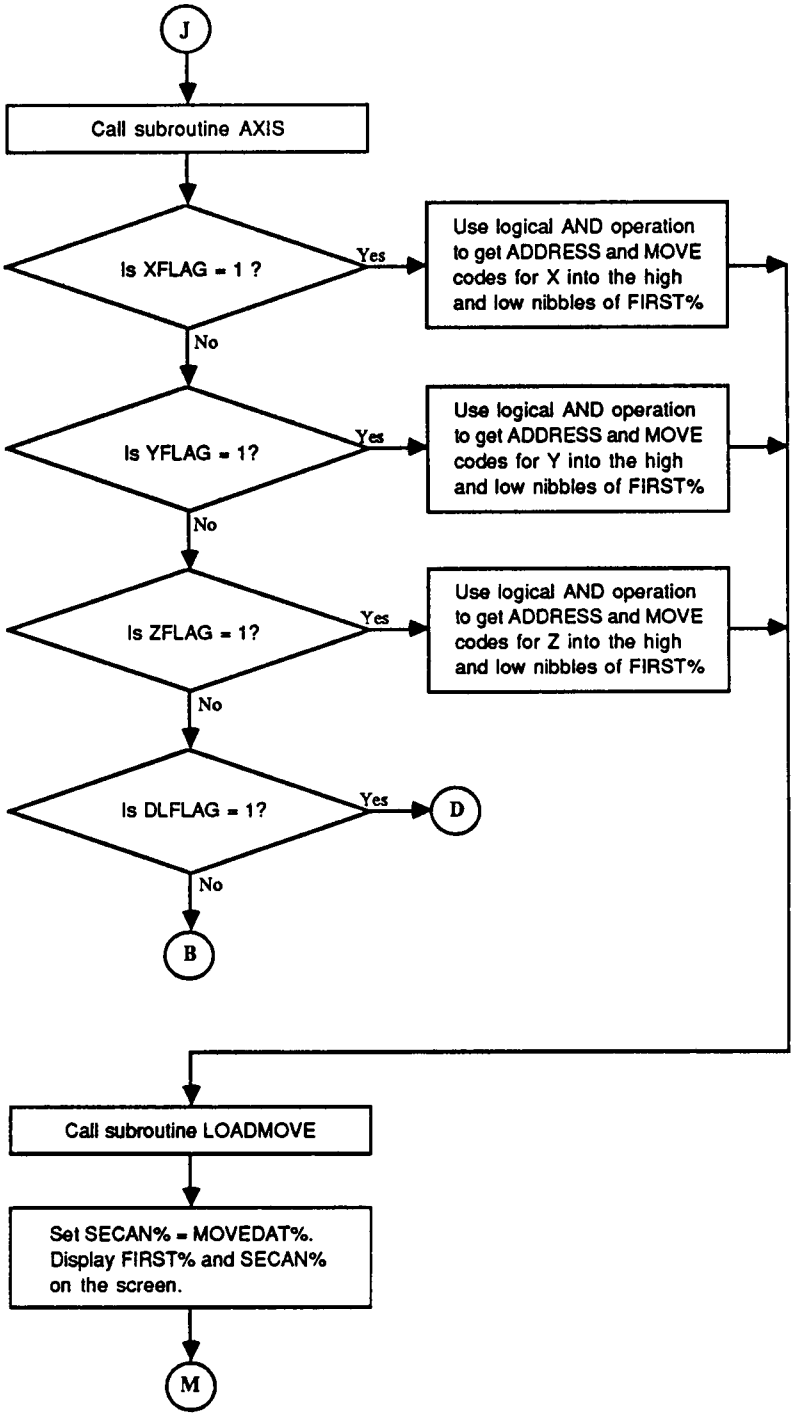
Flowchart of ROBOT3.BAS (continued)



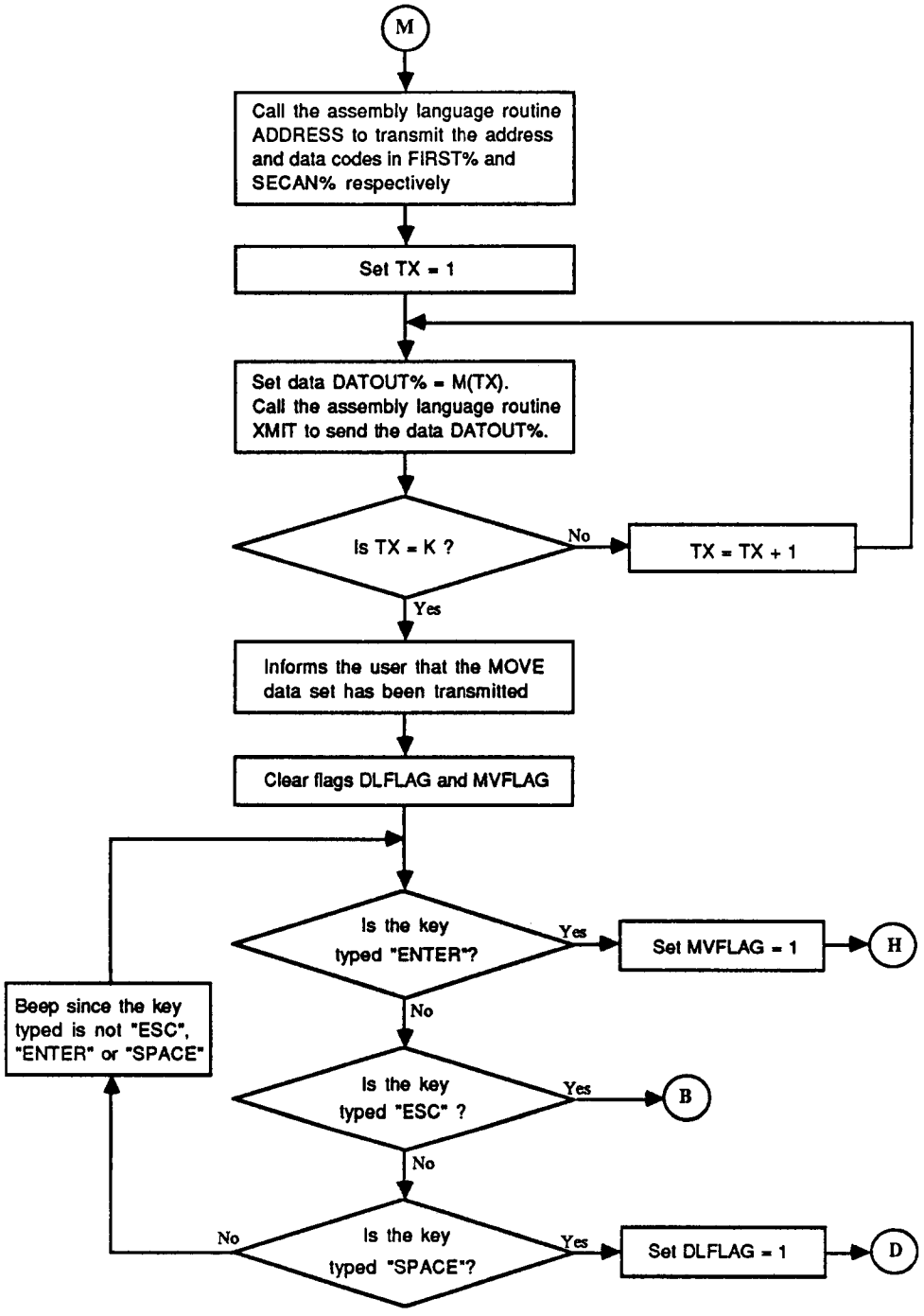
Flowchart of ROBOT3.BAS (continued)



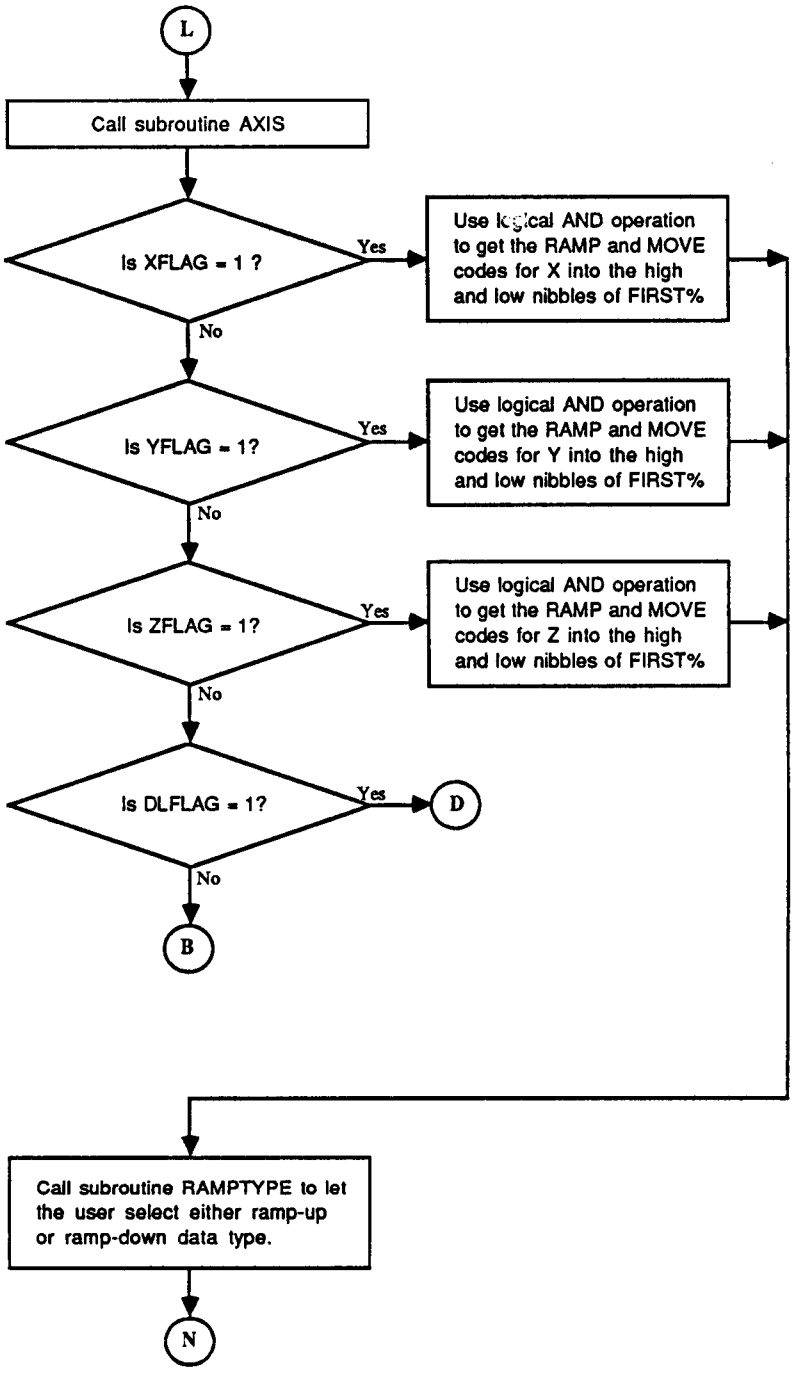
Flowchart of ROBOT3.BAS (continued)



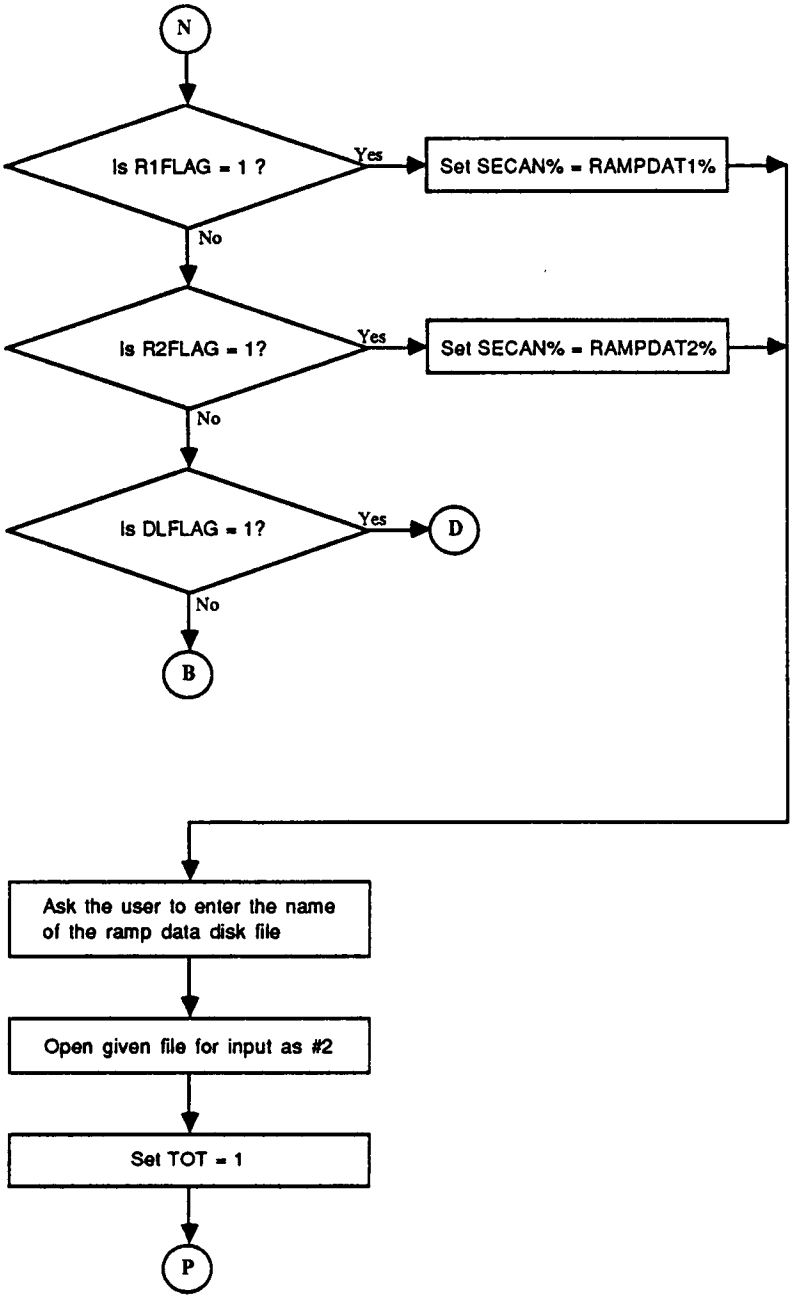
Flowchart of ROBOT3.BAS (continued)



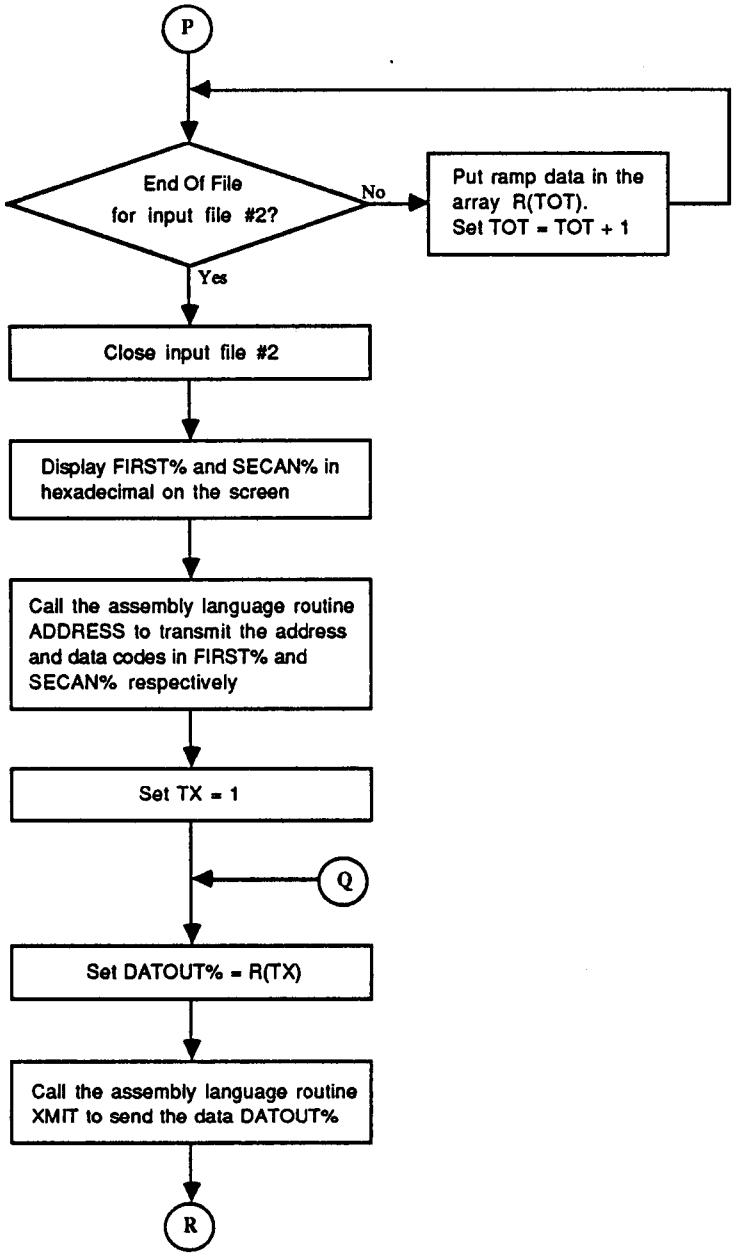
Flowchart of ROBOT3.BAS (Continued)



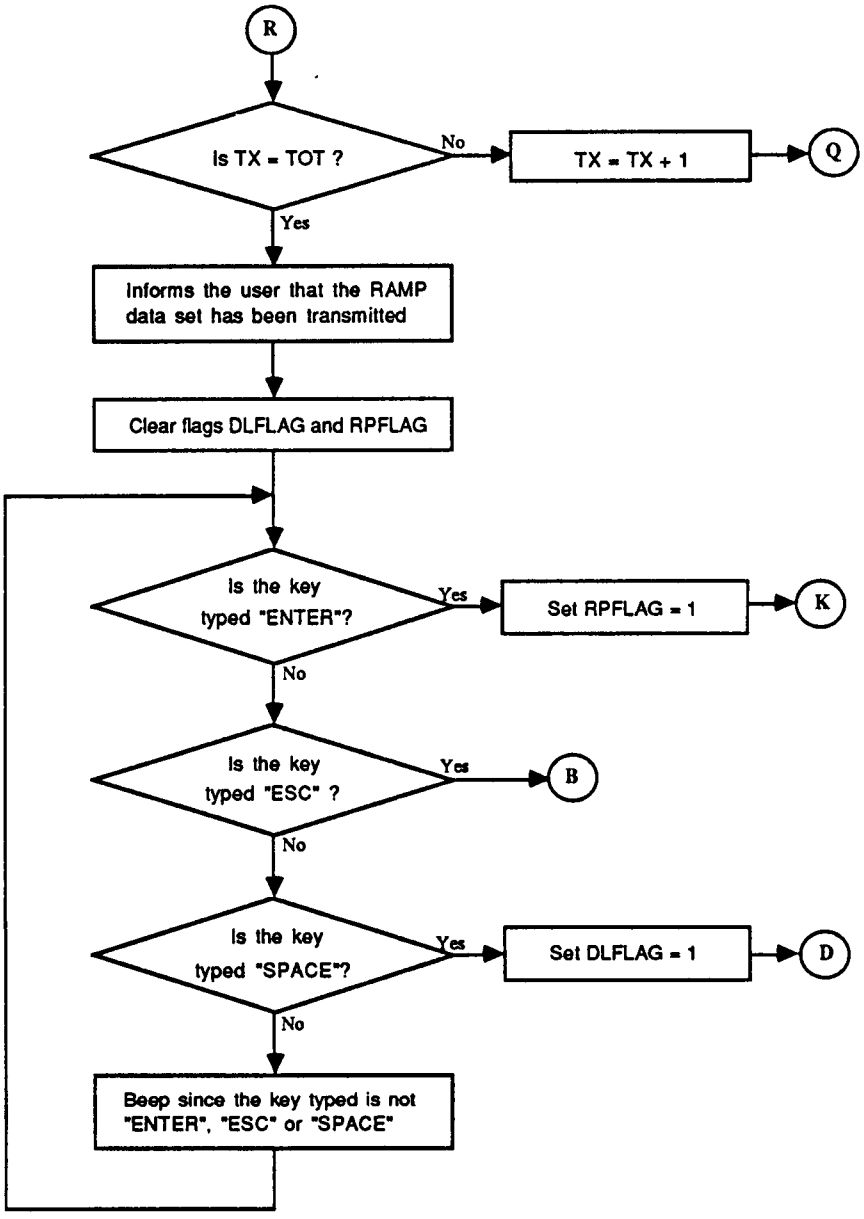
Flowchart of ROBOT3.BAS (continued)



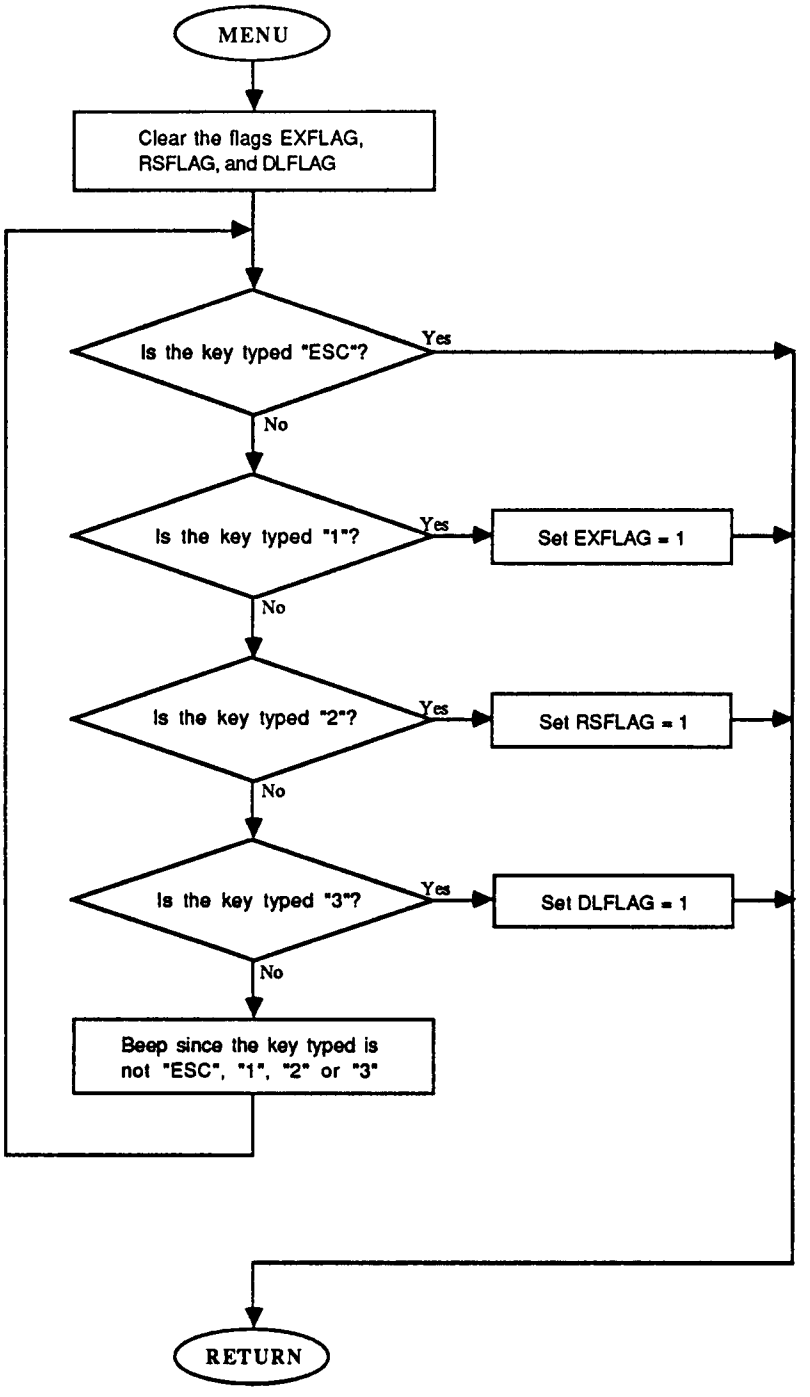
Flowchart of ROBOT3.BAS (continued)



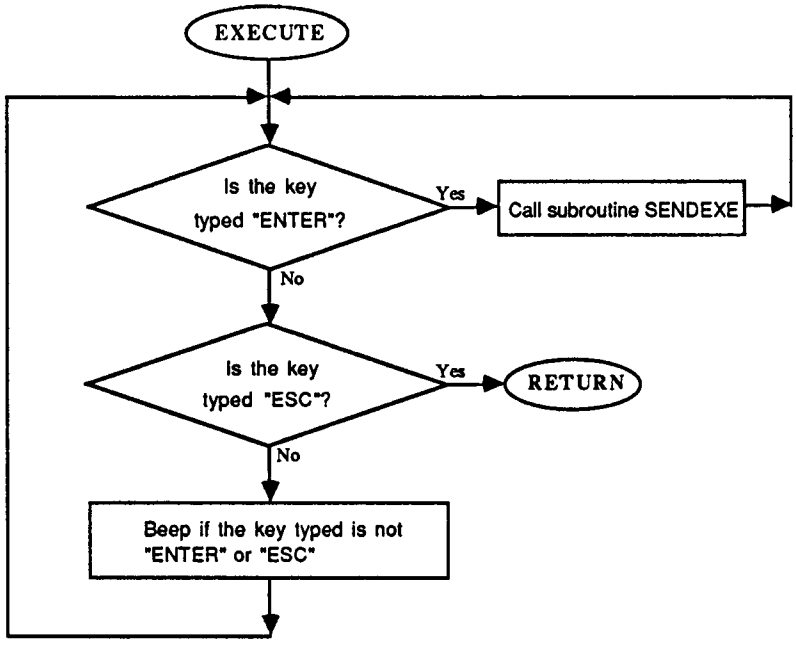
Flowchart of ROBOT3.BAS (continued)



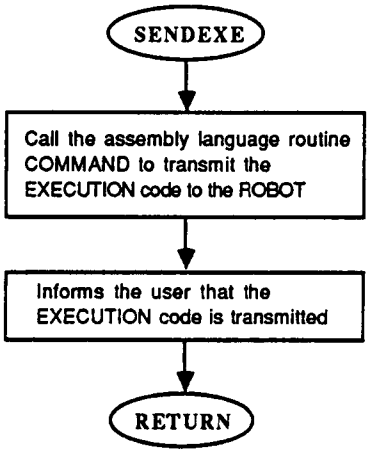
Flowchart of ROBOT3.BAS (Continued)



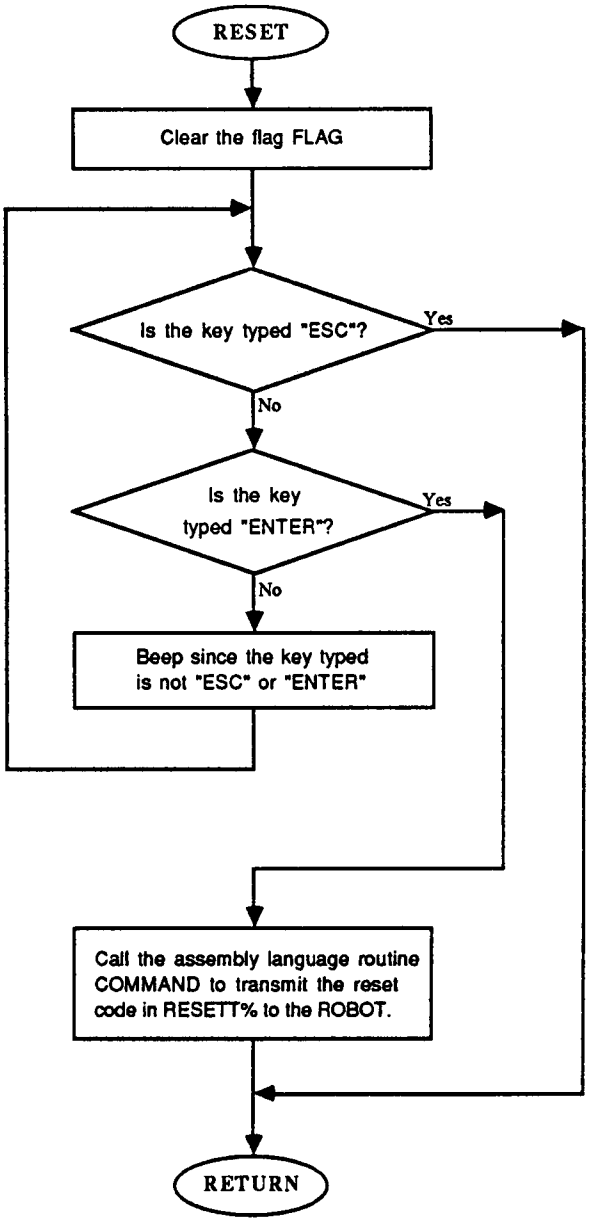
Flowchart of subroutine MENU of ROBOT3.BAS



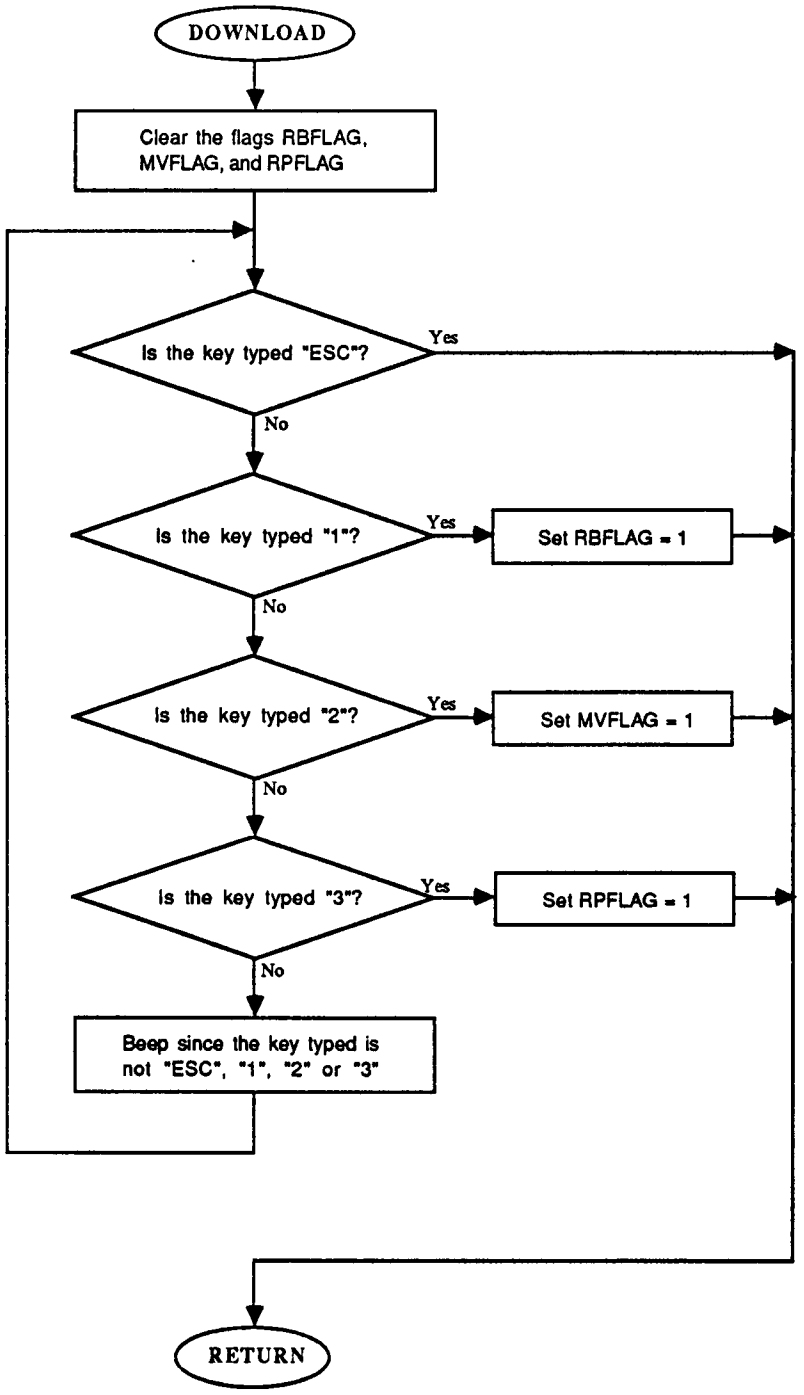
Flowchart of subroutine EXECUTE of ROBOT3.BAS



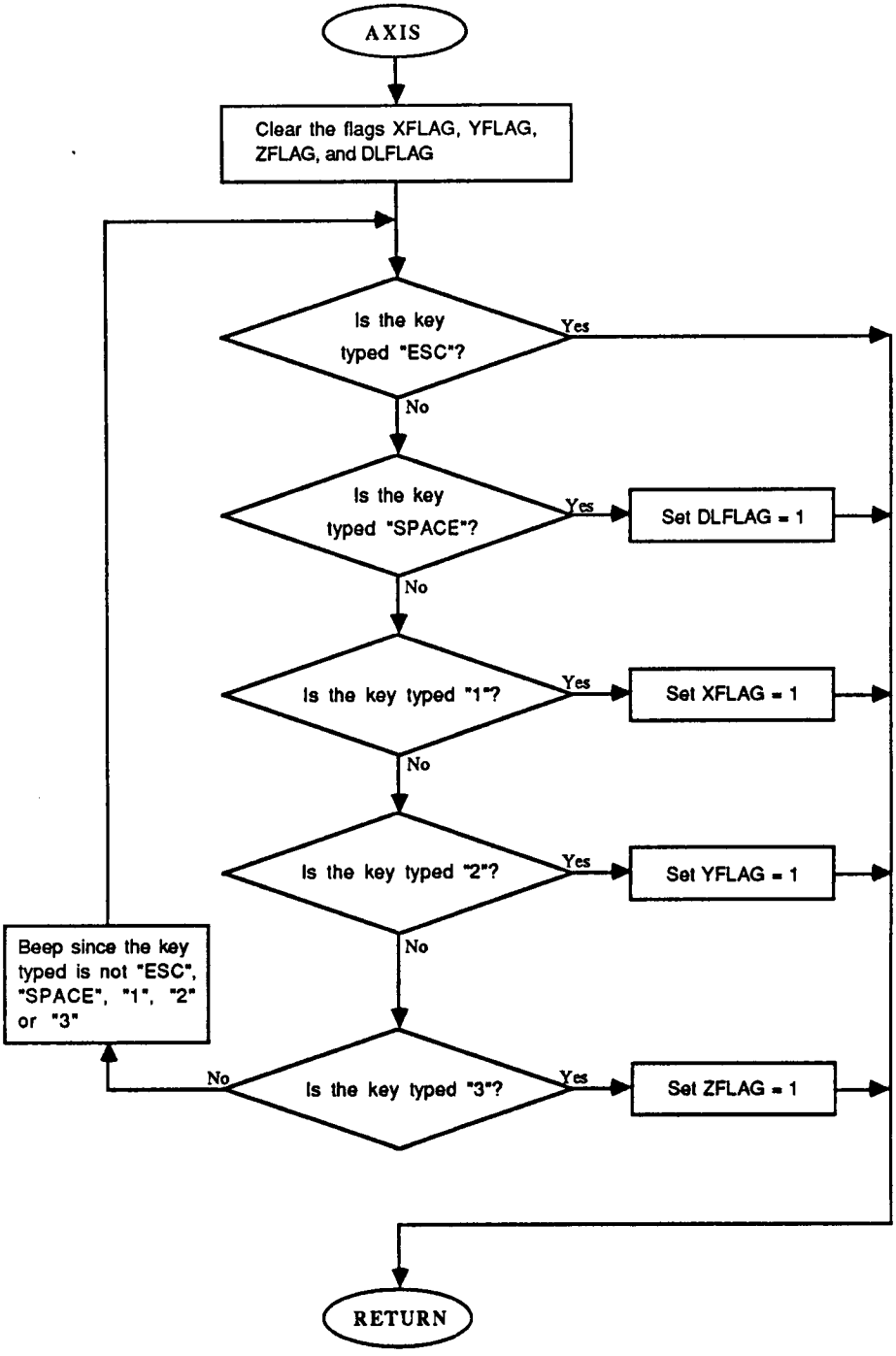
Flowchart of subroutine SENDEXE of ROBOT3.BAS



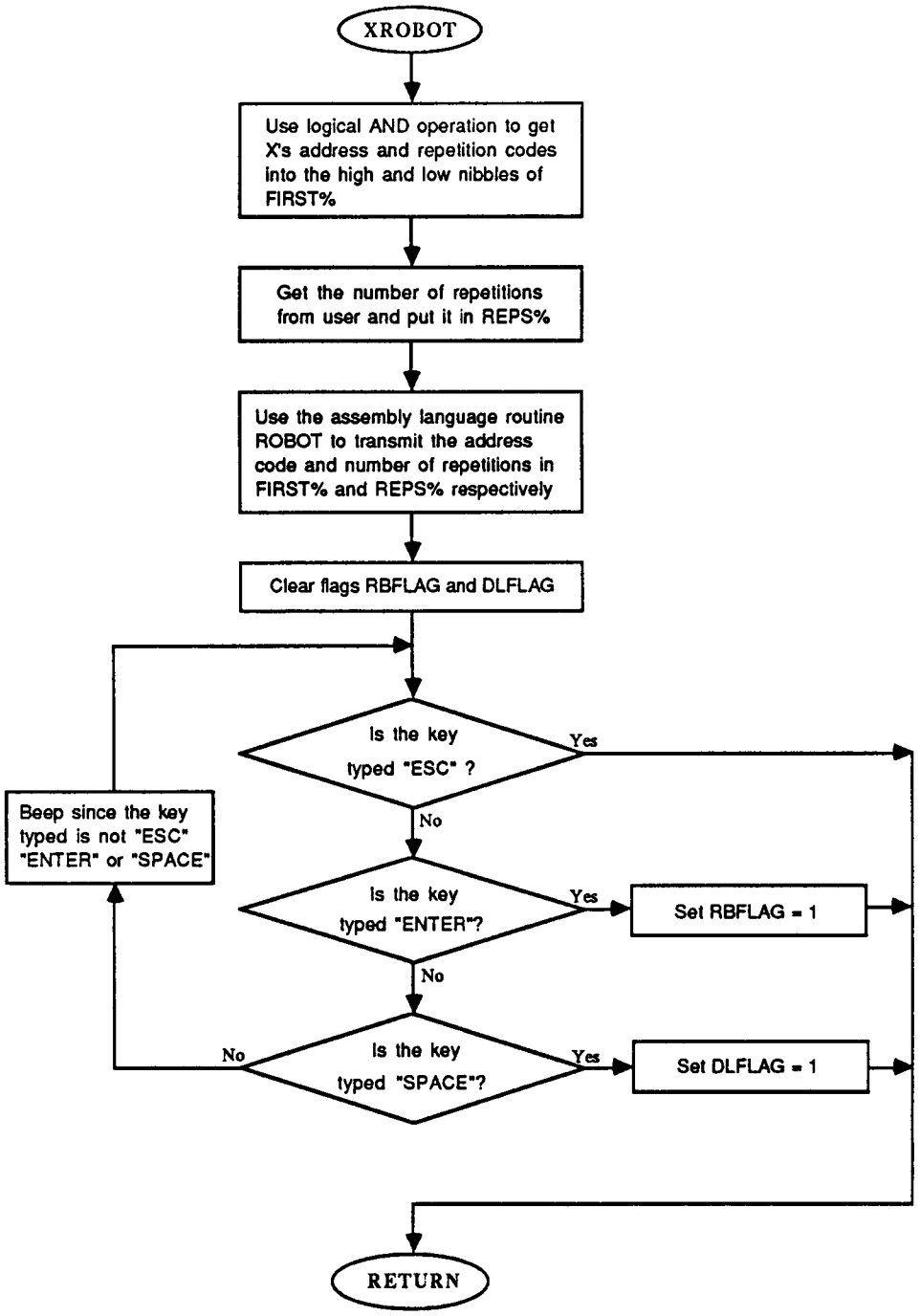
Flowchart of subroutine RESET of ROBOT3.BAS



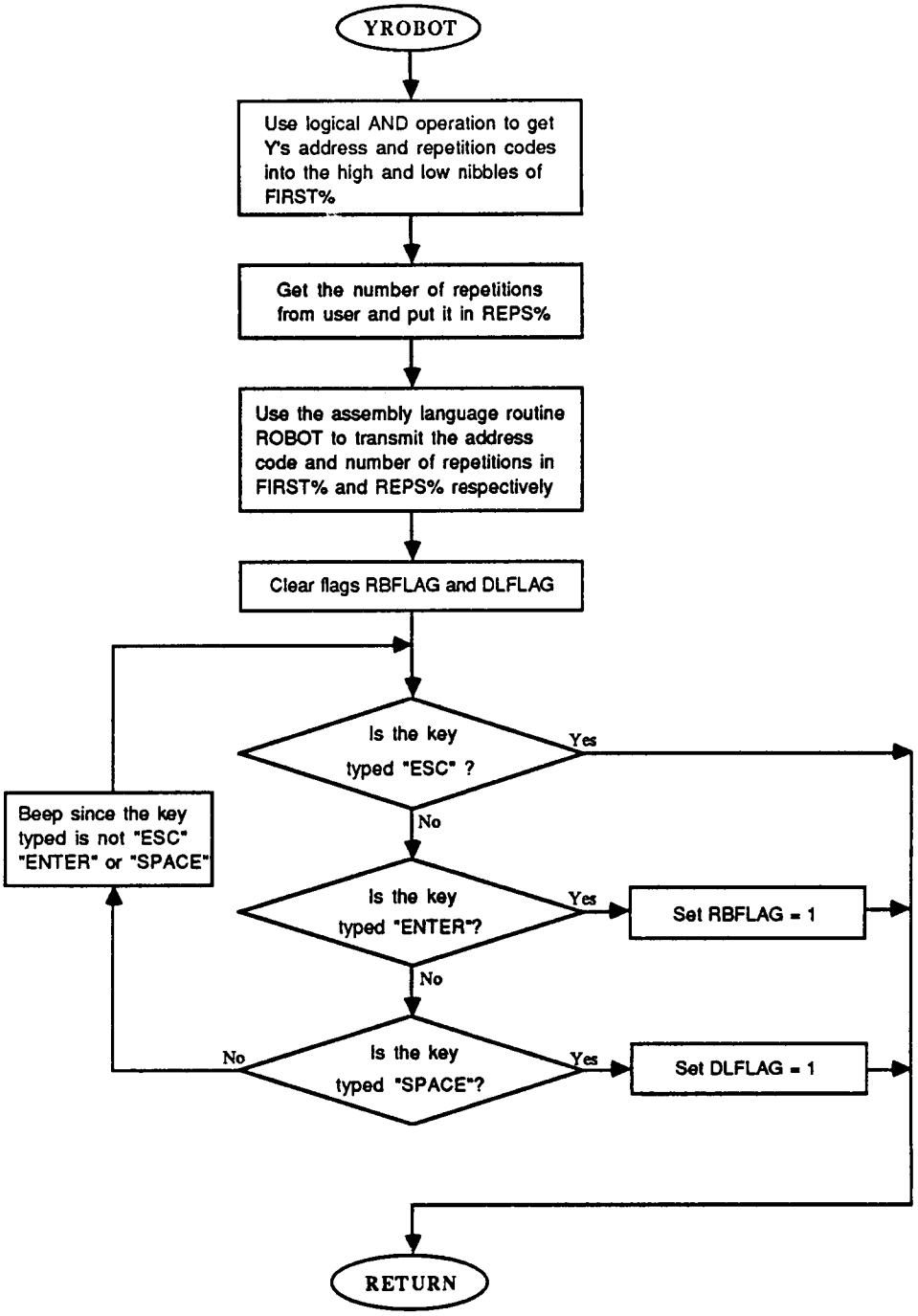
Flowchart of subroutine DOWNLOAD of ROBOT3.BAS



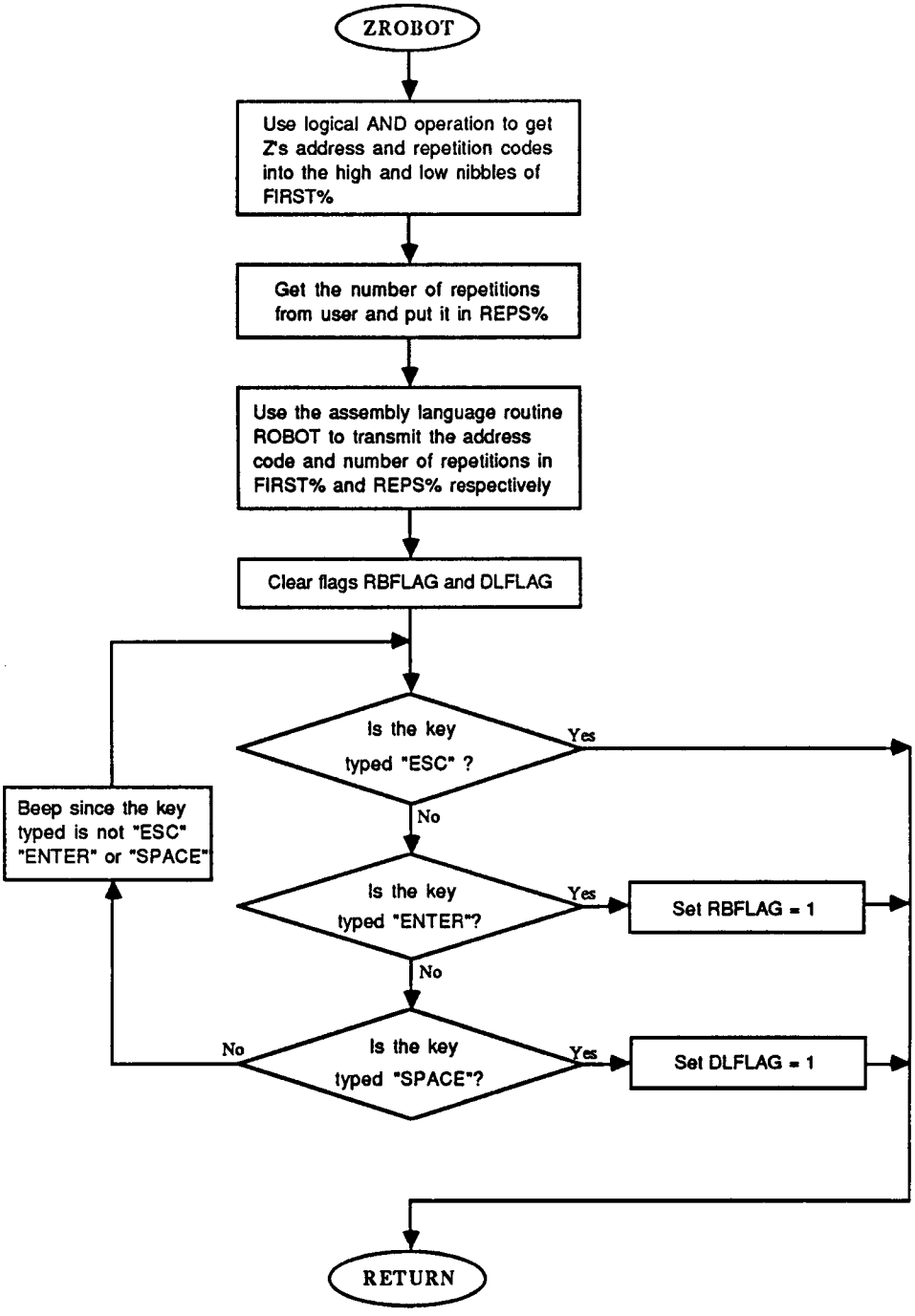
Flowchart of subroutine AXIS of ROBOT3.BAS



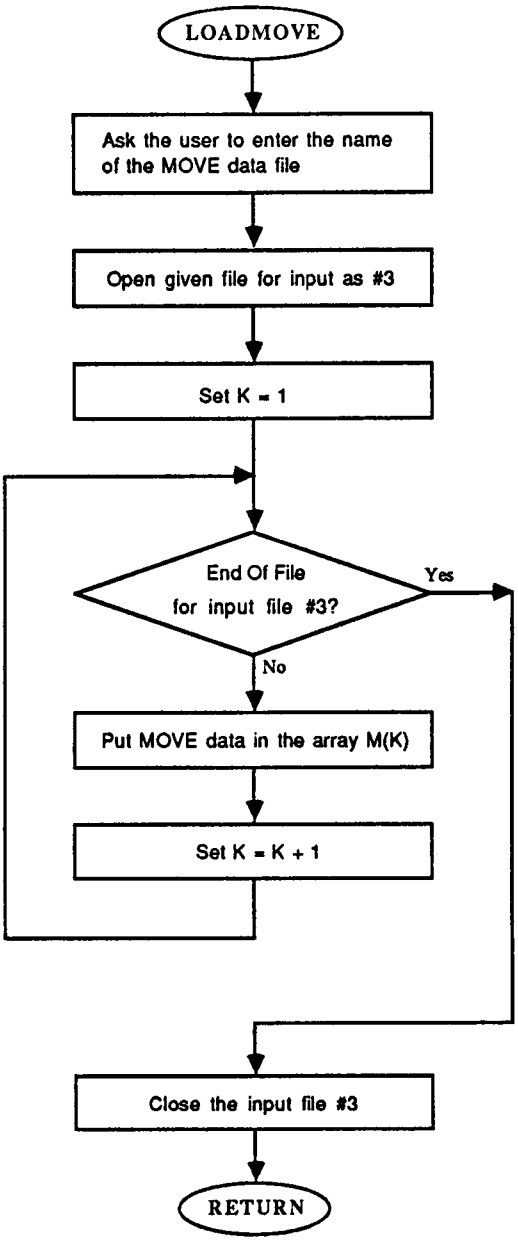
Flowchart of subroutine XROBOT of ROBOT3.BAS



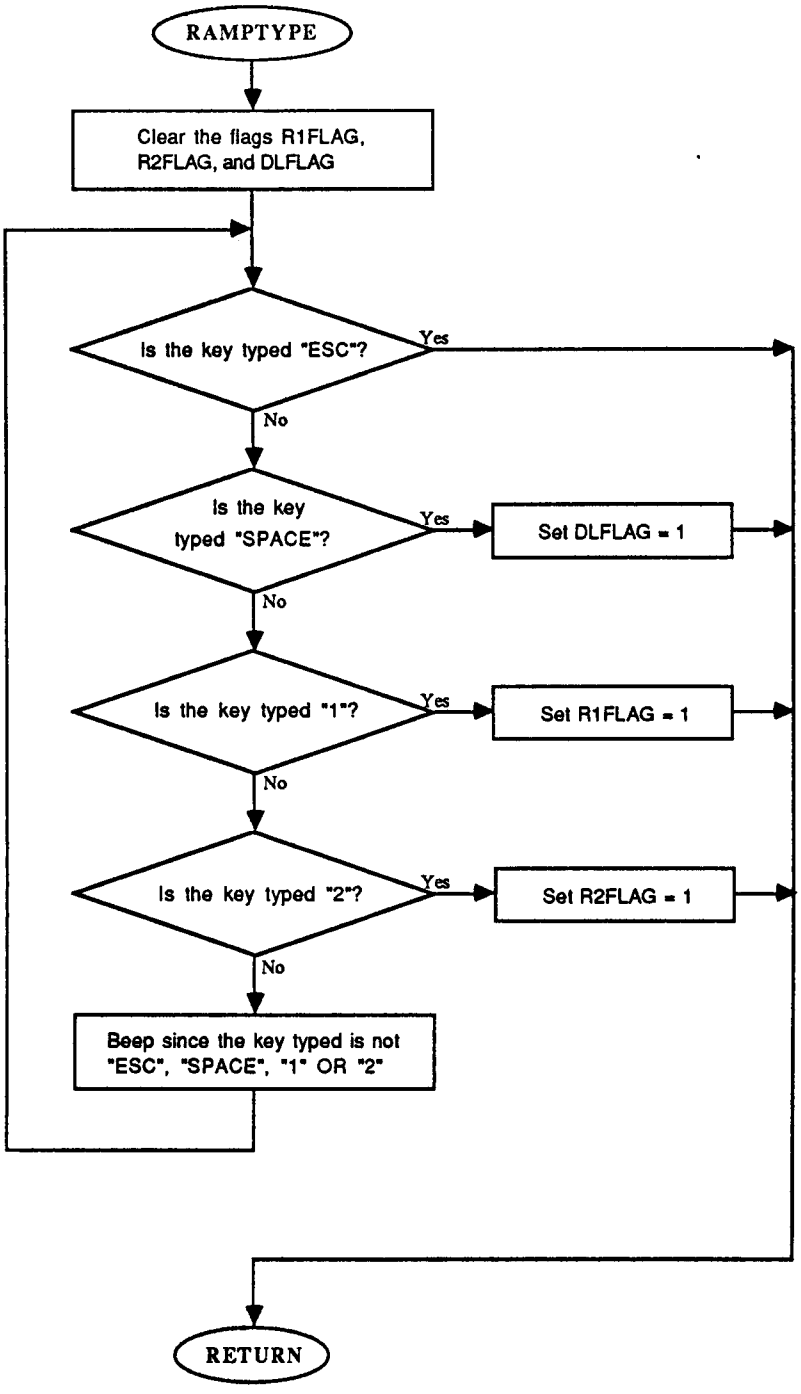
Flowchart of subroutine YROBOT of ROBOT3.BAS



Flowchart of subroutine ZROBOT of ROBOT3.BAS



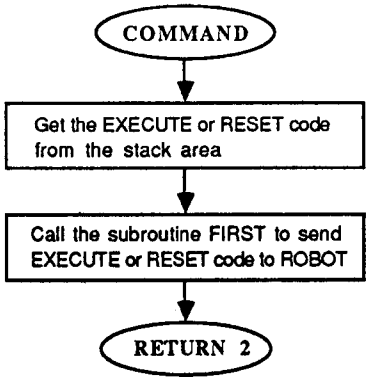
Flowchart of subroutine LOADMOVE of ROBOT3.BAS



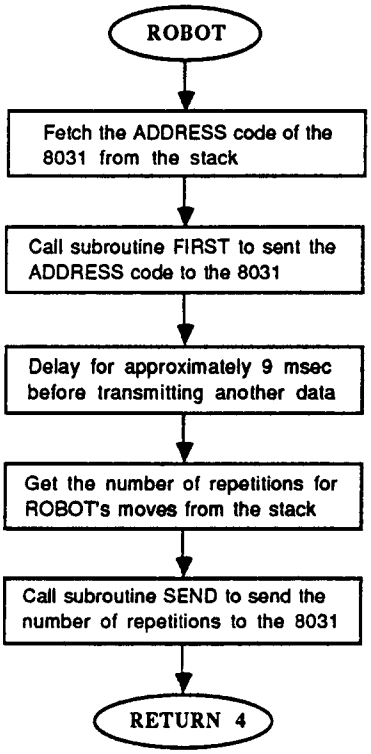
Flowchart of subroutine RAMPTYPE of ROBOT3.BAS

APPENDIX C

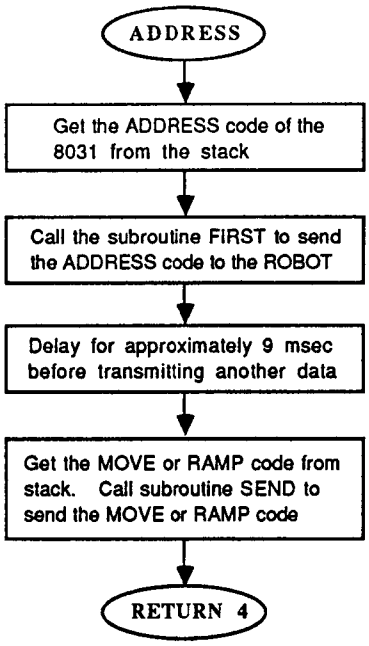
Flowcharts of the Assembly Language Program ROBOT



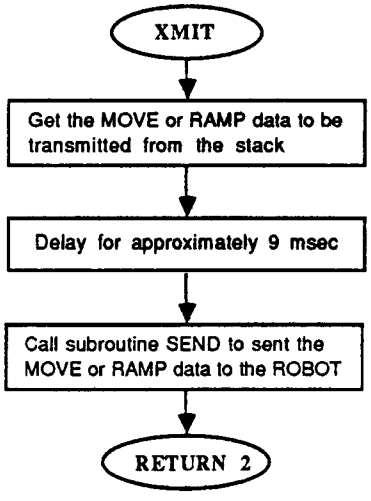
Flowchart of routine COMMAND of ROBOT.ASM



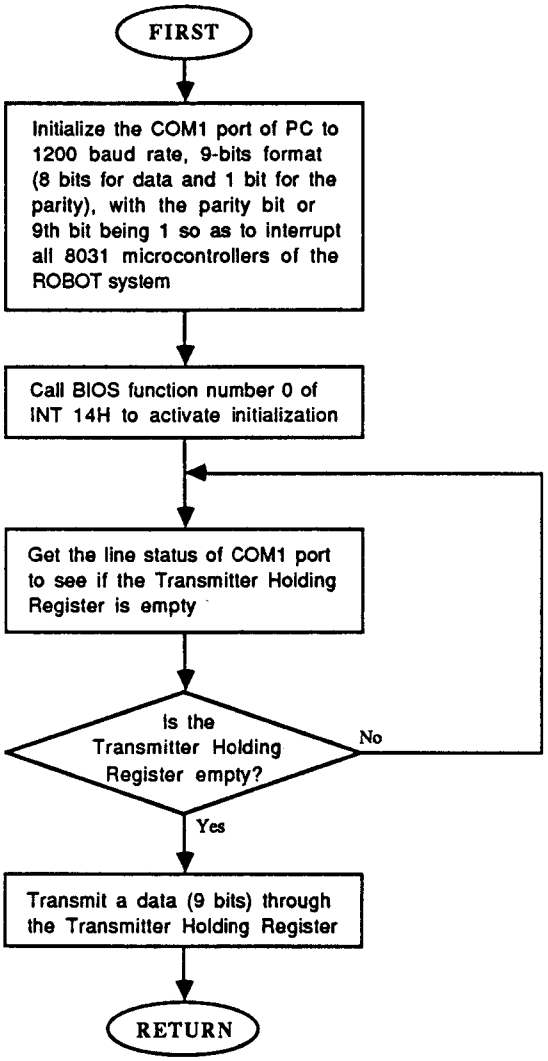
Flowchart of routine ROBOT of ROBOT.ASM



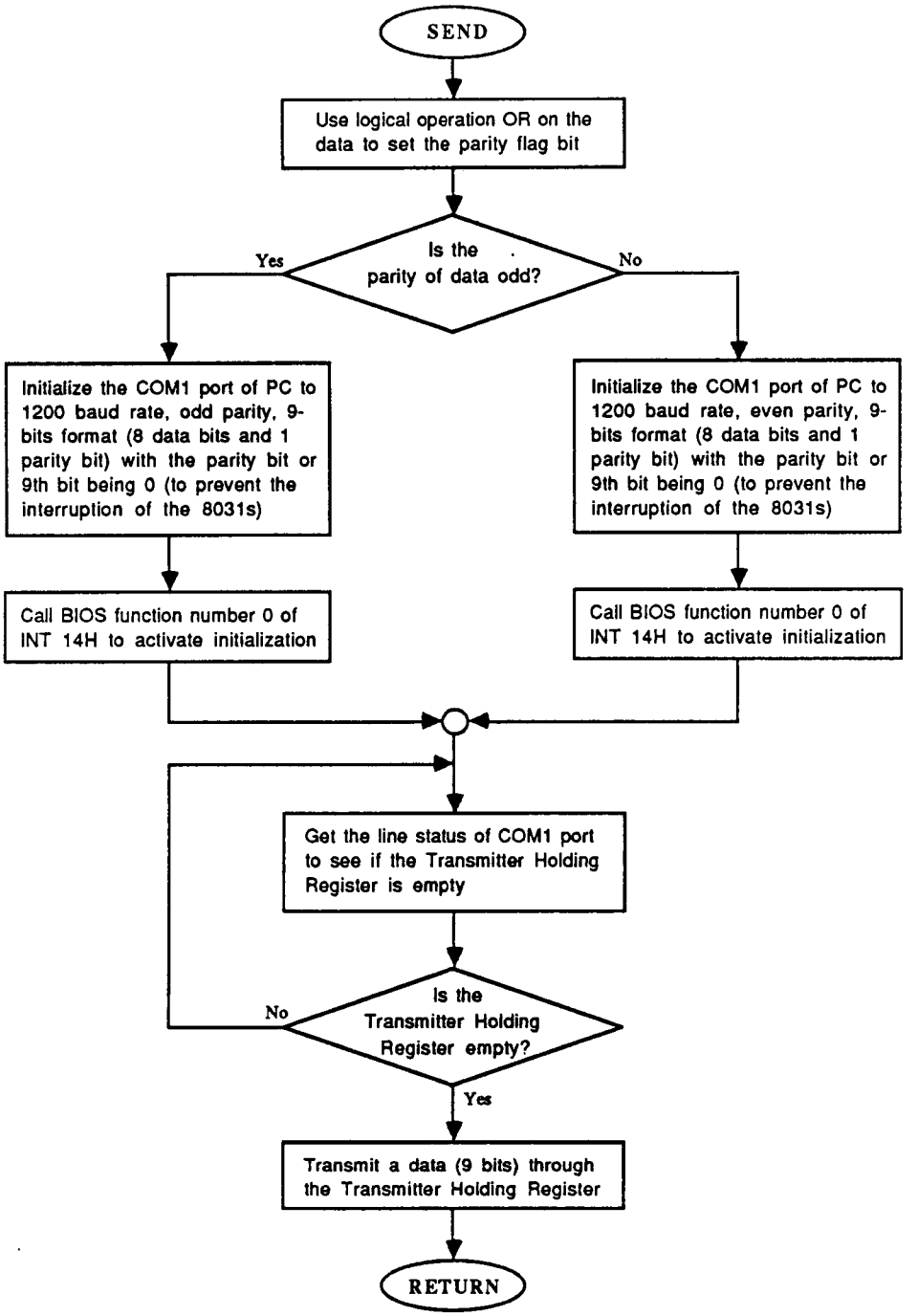
Flowchart of routine ADDRESS of ROBOT.ASM



Flowchart of routine XMIT of ROBOT.ASM



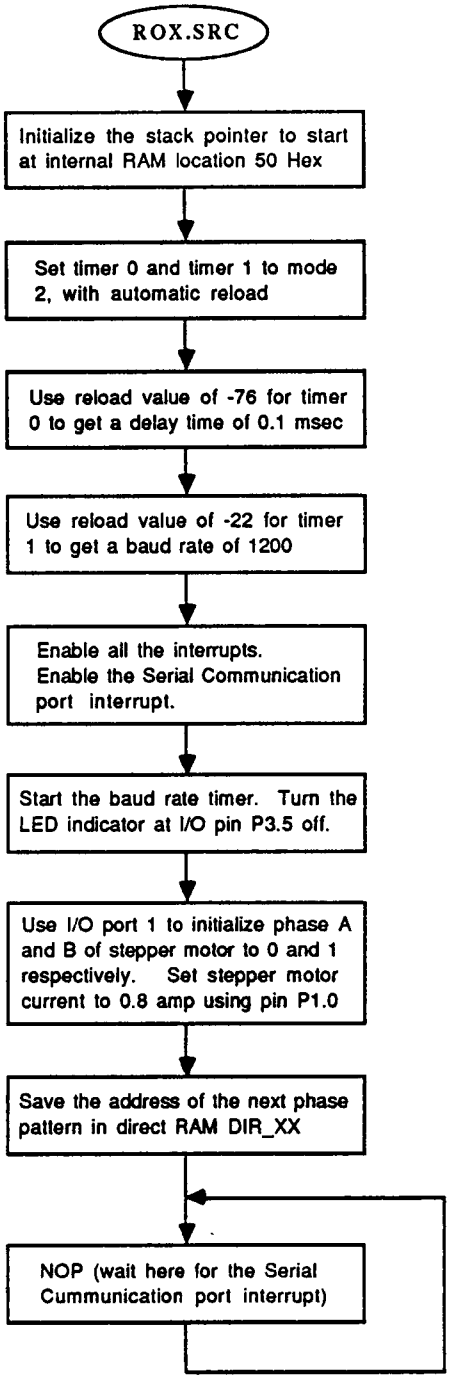
Flowchart of subroutine FIRST of ROBOT.ASM



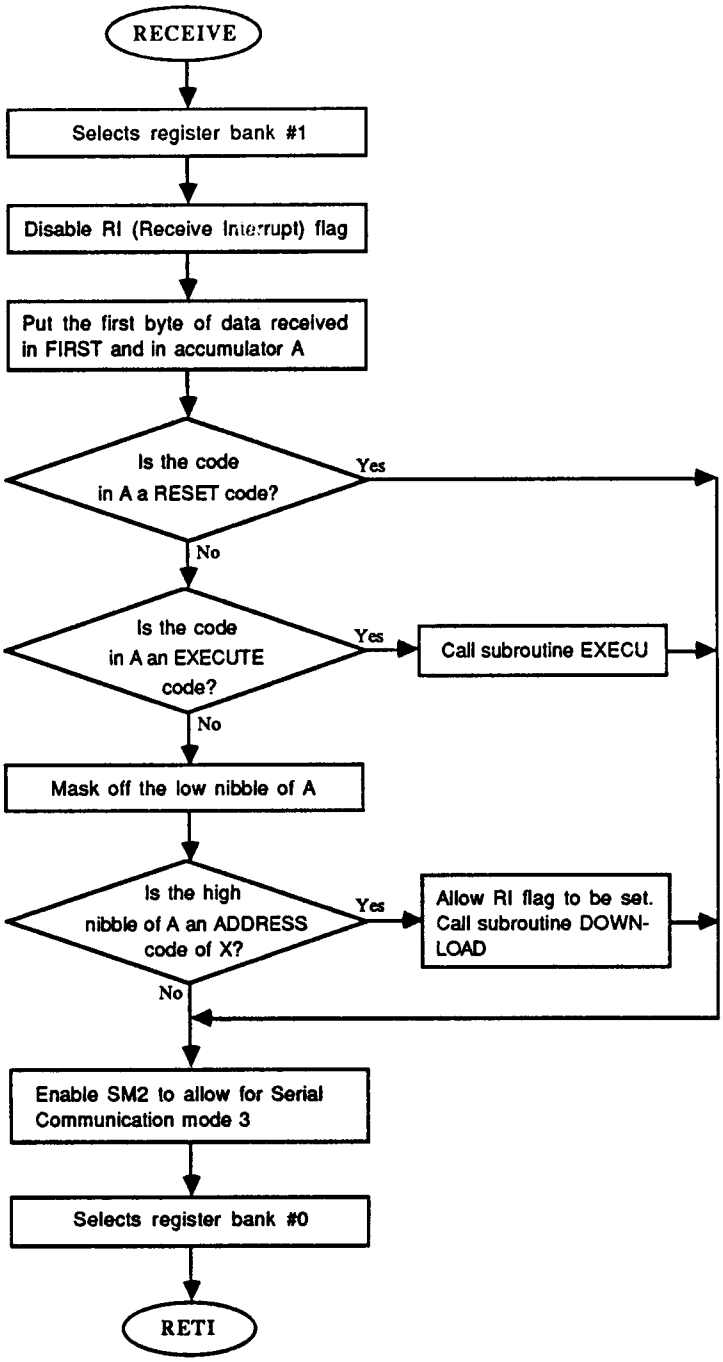
Flowchart of subroutine SEND of ROBOT.ASM

APPENDIX D

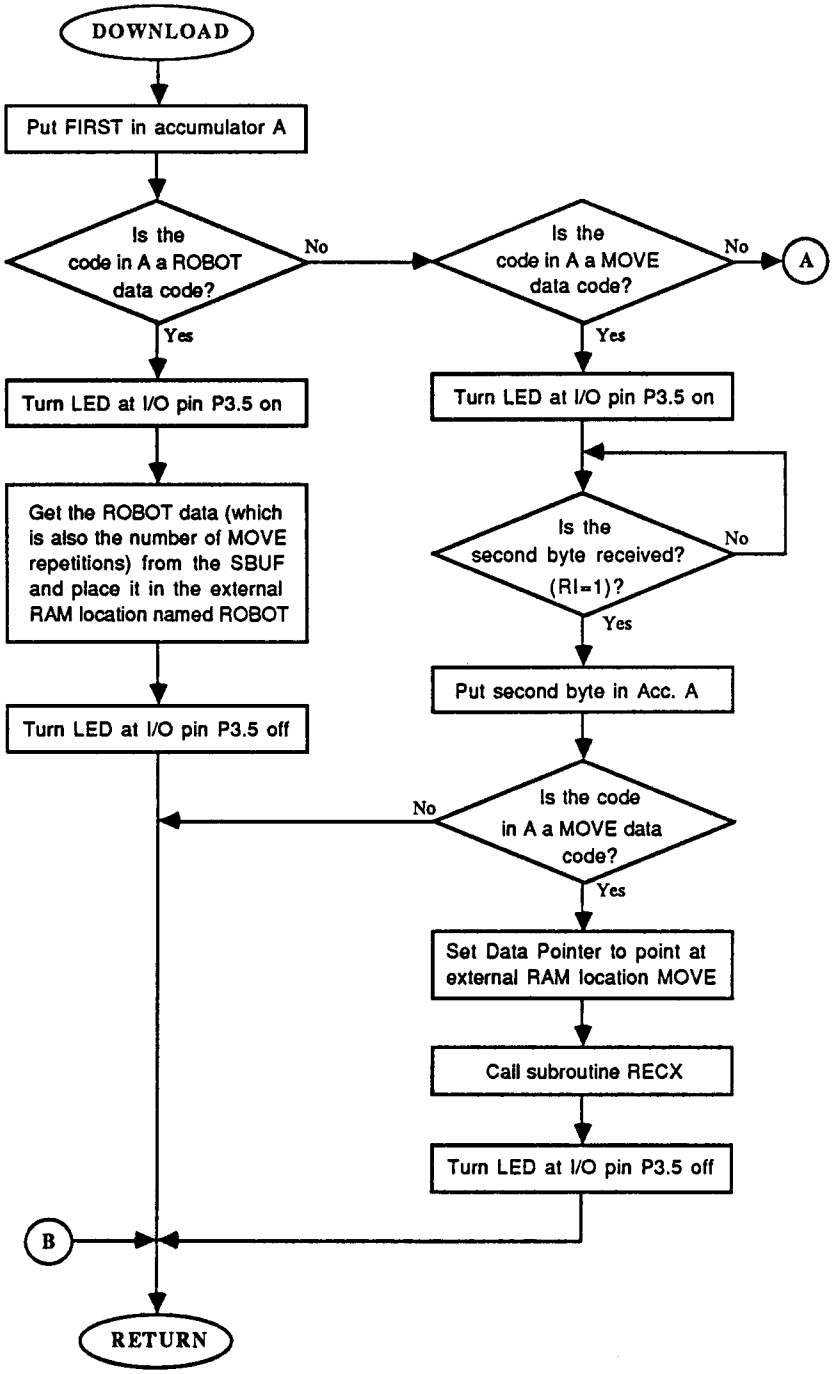
Flowcharts of the MCS-51 Assembly Language Program ROX



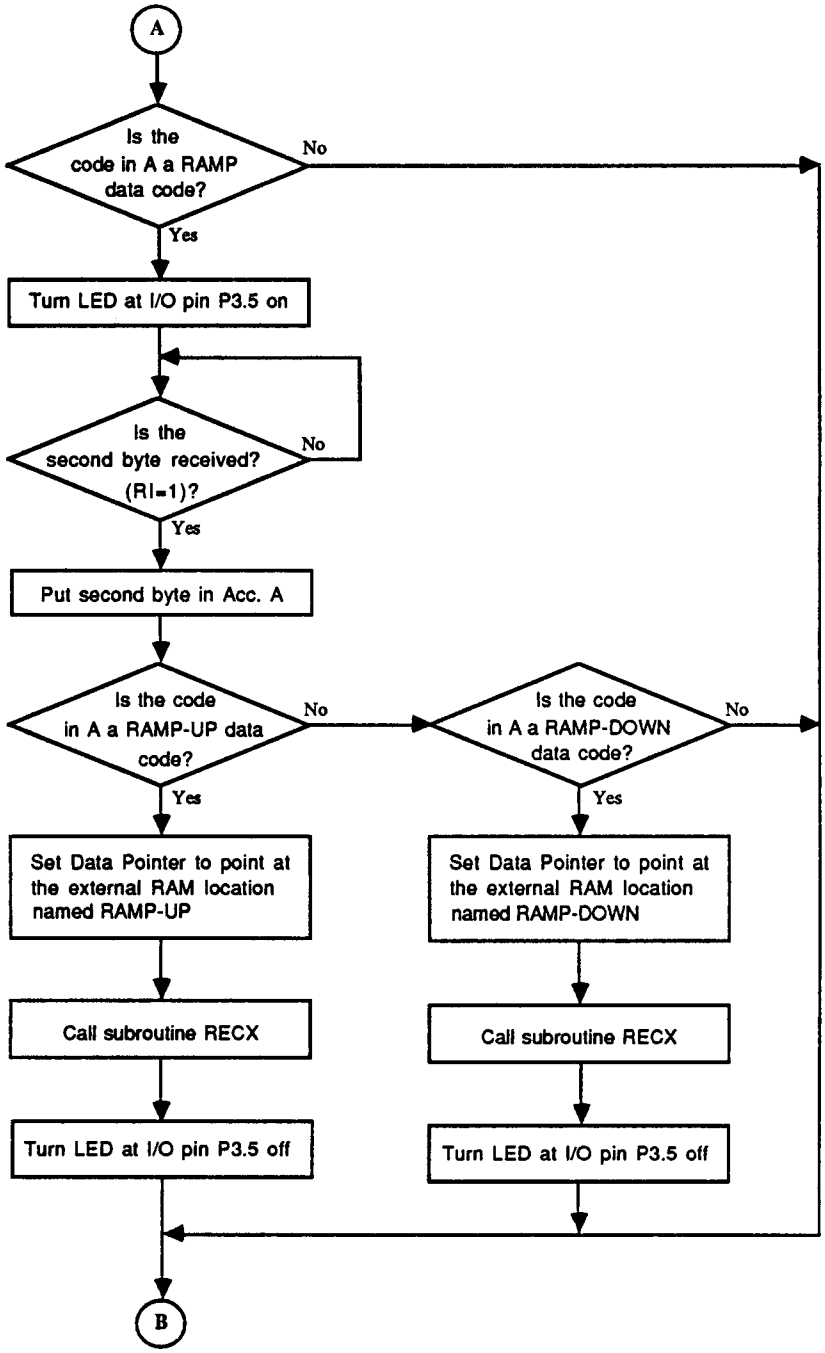
Flowchart of main routine of 8031 program ROX.SRC



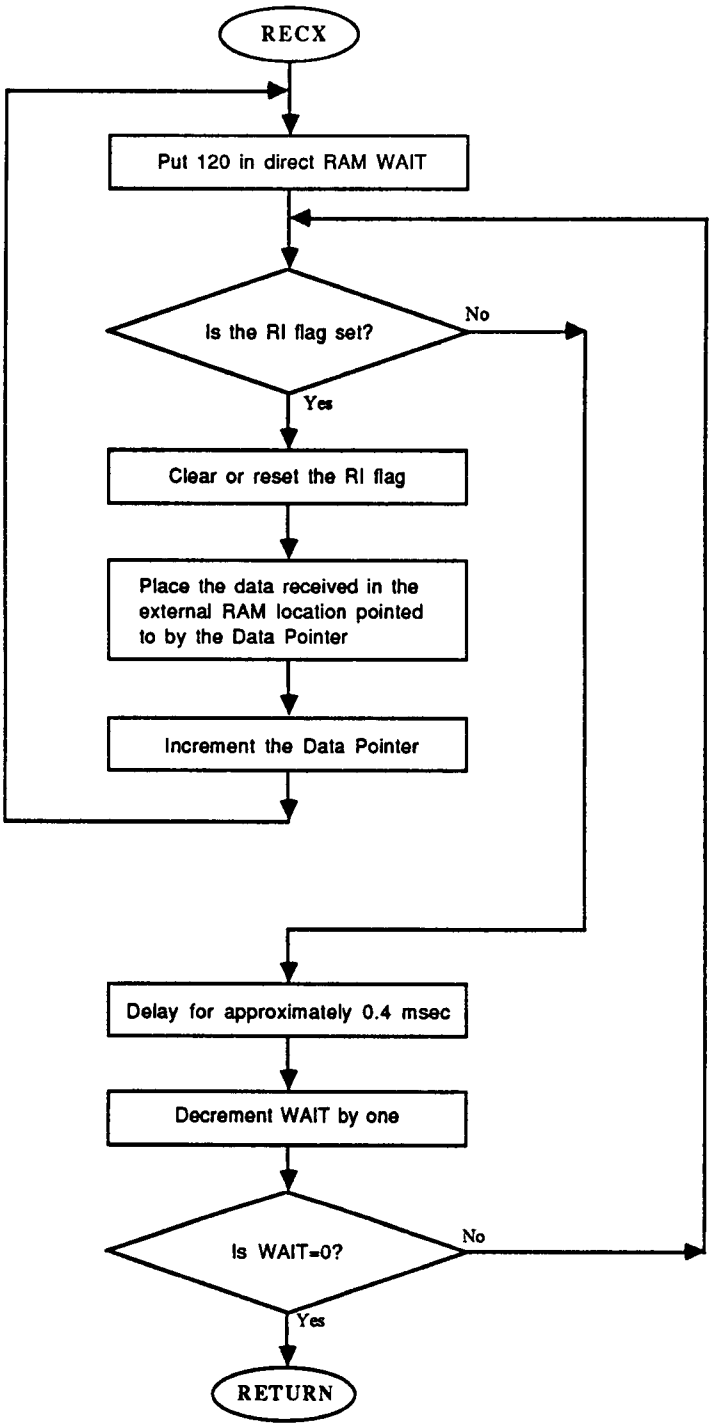
Flowchart of Interrupt Service Routine RECEIVE of ROX.SRC



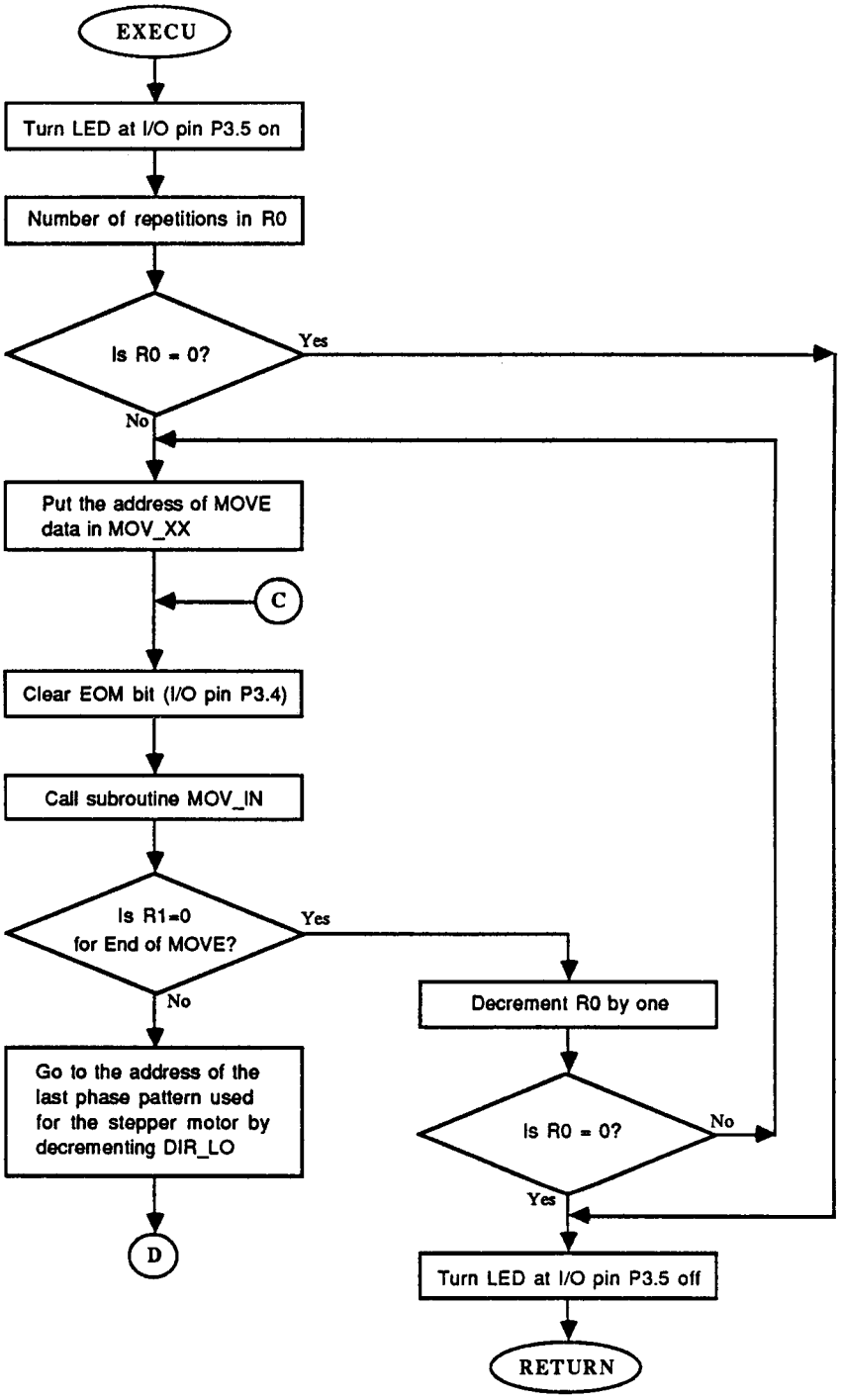
Flowchart of subroutine DOWNLOAD of ROX.SRC



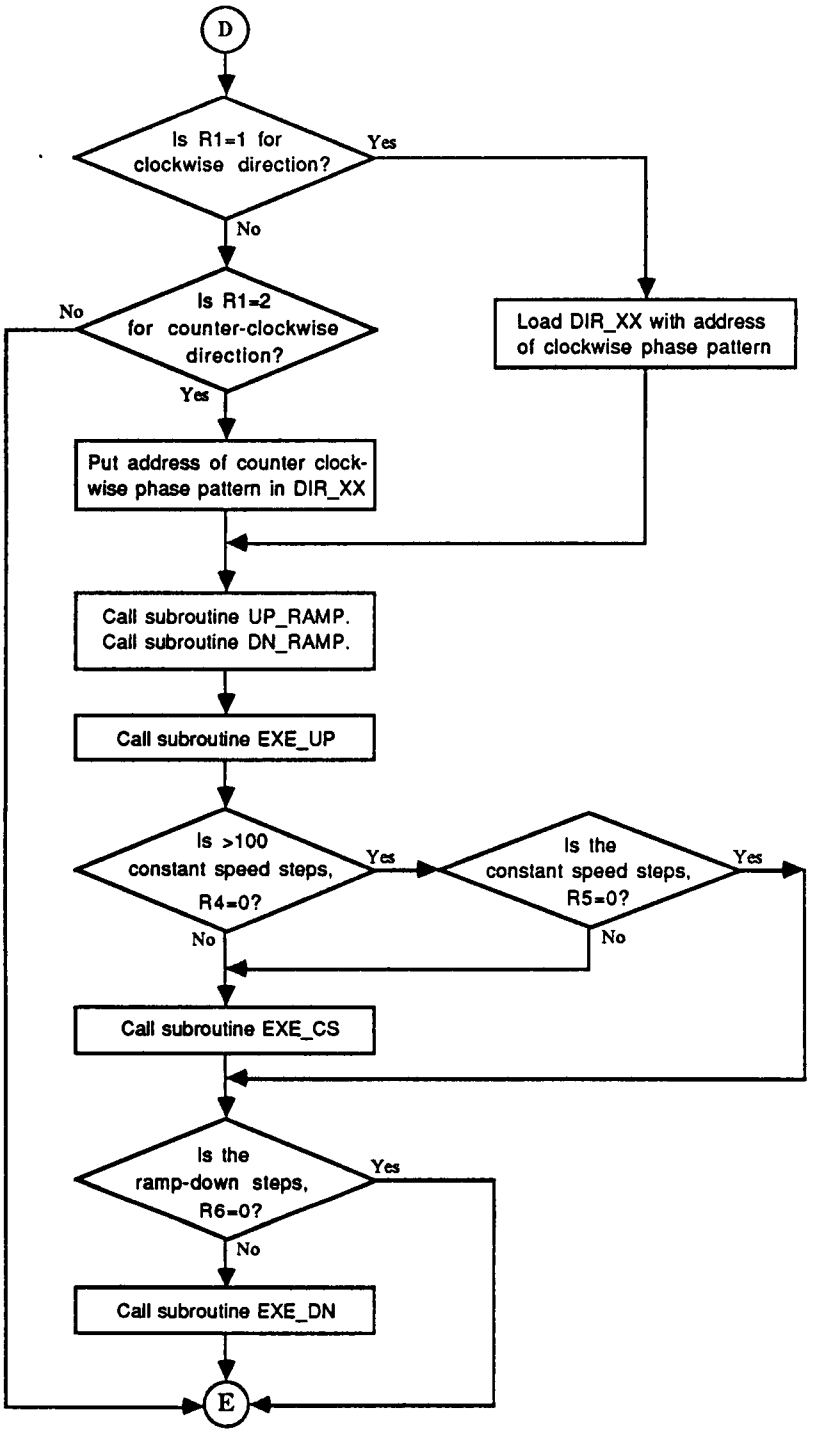
Flowchart of subroutine DOWNLOAD of ROX.SRC (continued)



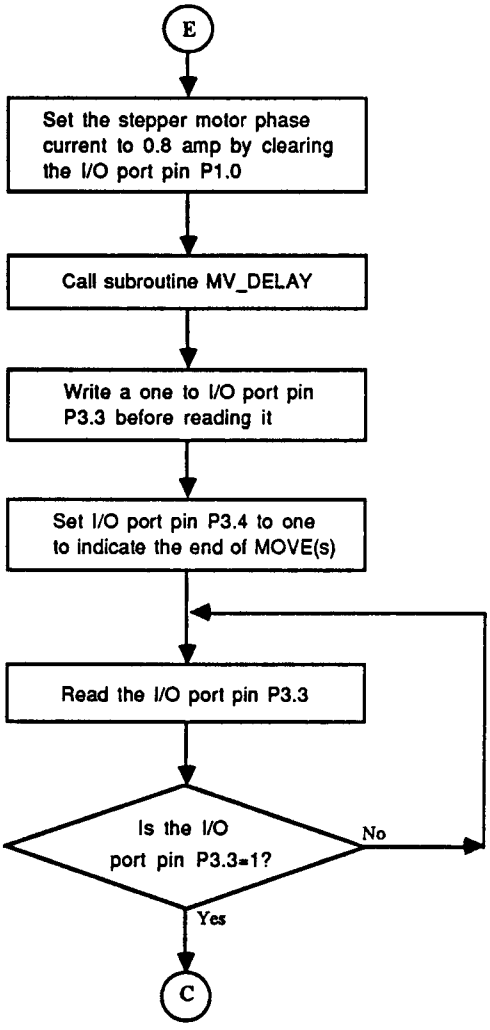
Flowchart of subroutine RECX of ROX.SRC



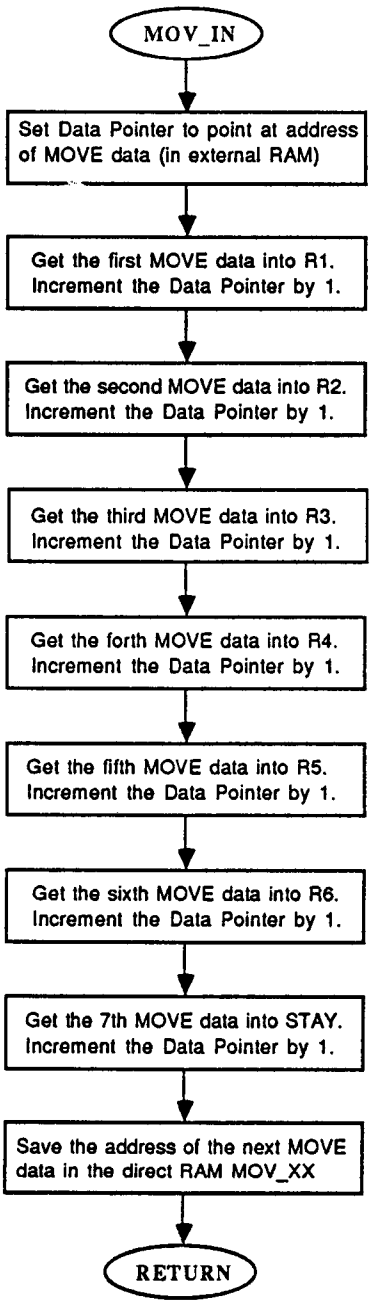
Flowchart of subroutine EXECU of ROX.SRC



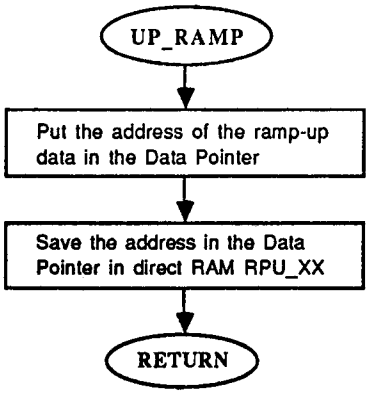
Flowchart of subroutine EXECU of ROX.SRC (continued)



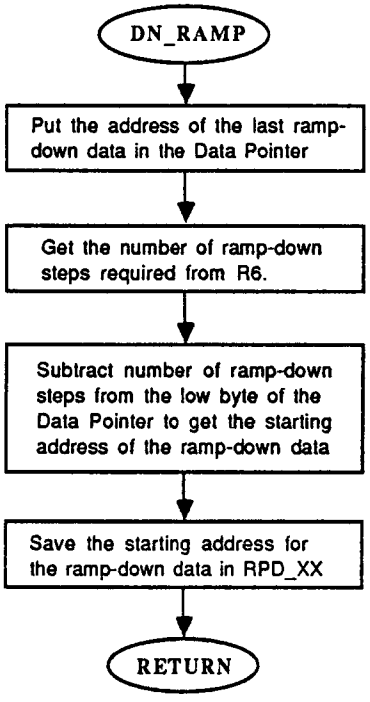
Flowchart of subroutine EXECU of ROX.SRC (Continued)



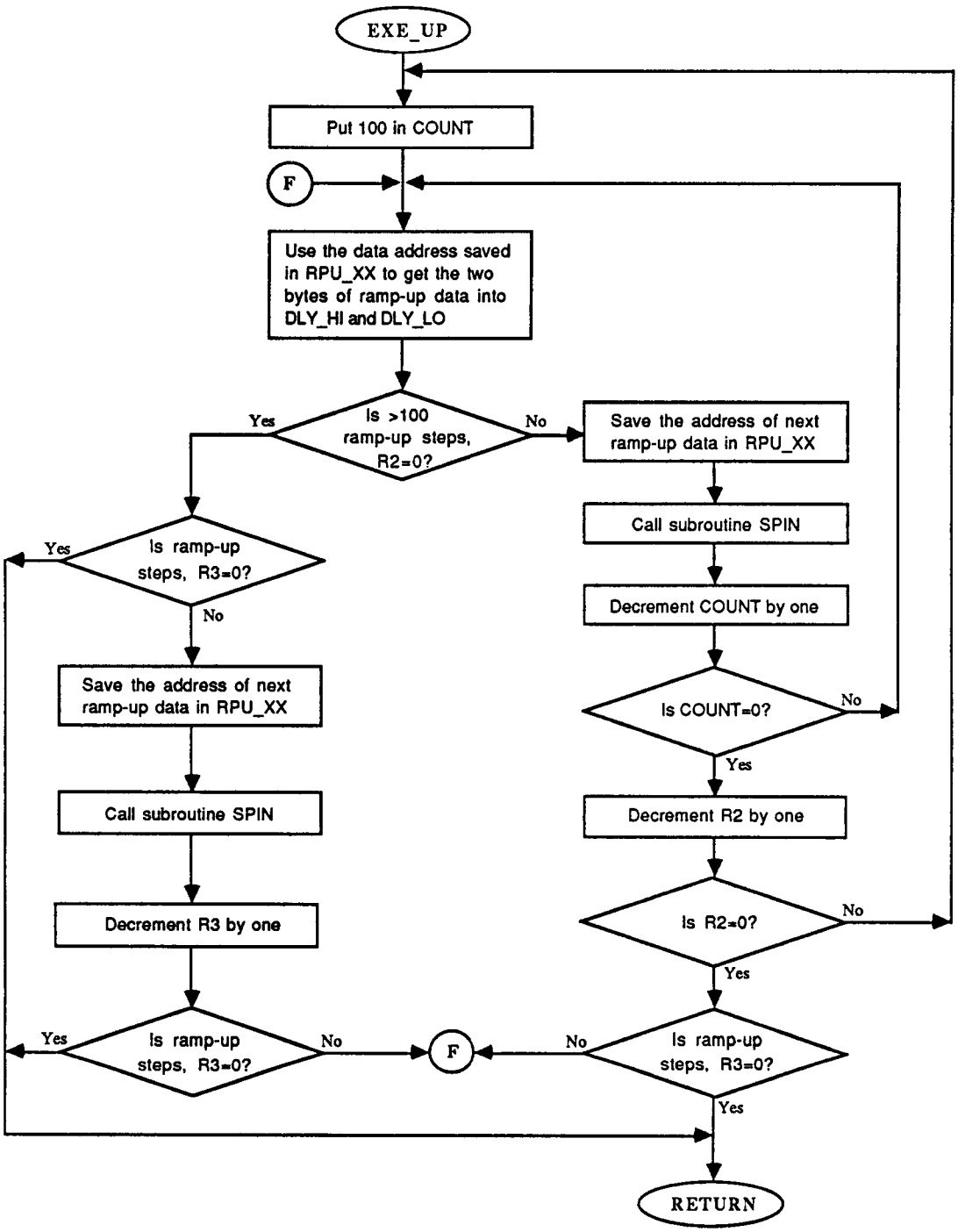
Flowchart of subroutine MOVE_IN of ROX.SRC



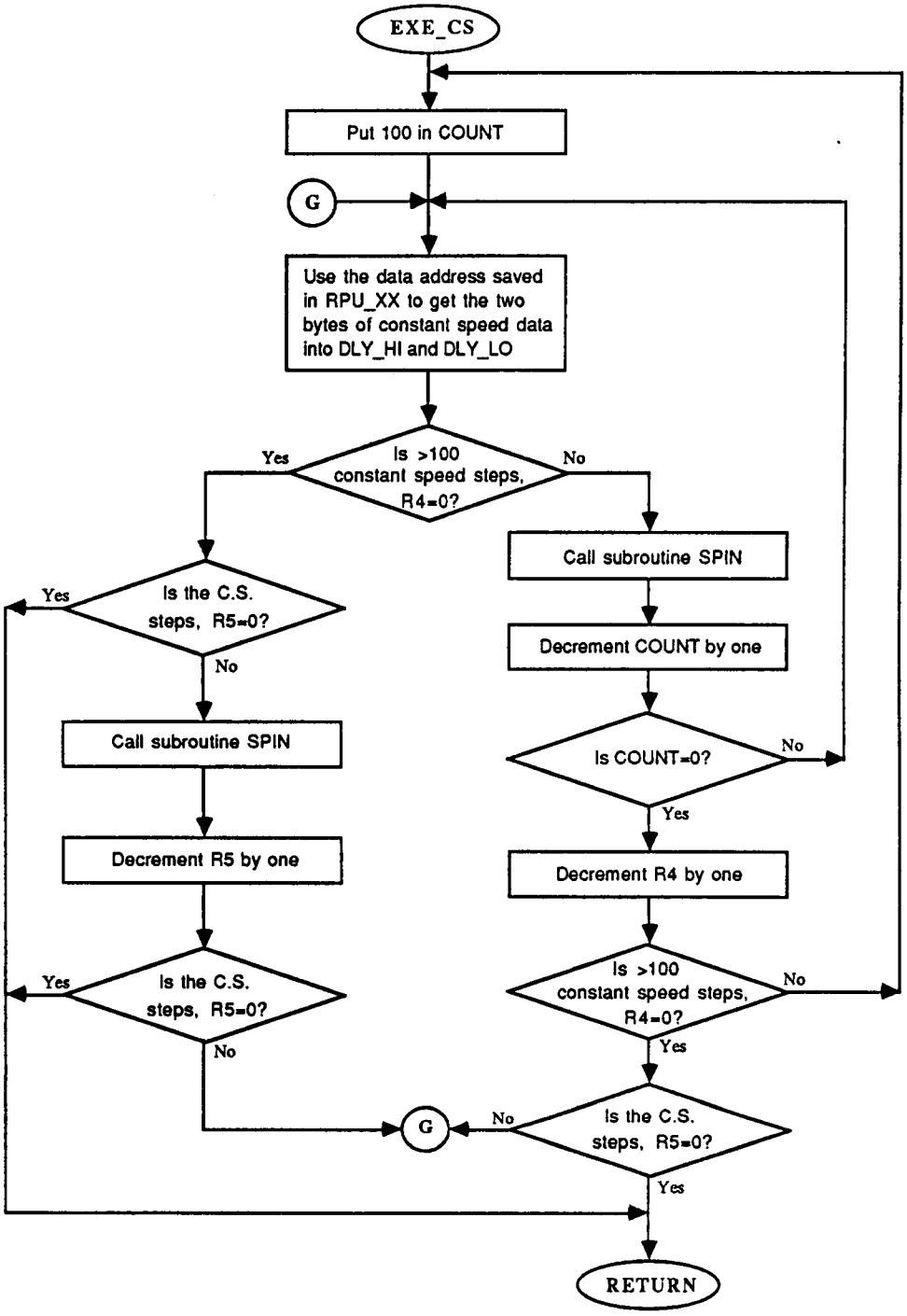
Flowchart of subroutine UP_RAMP of ROX.SRC



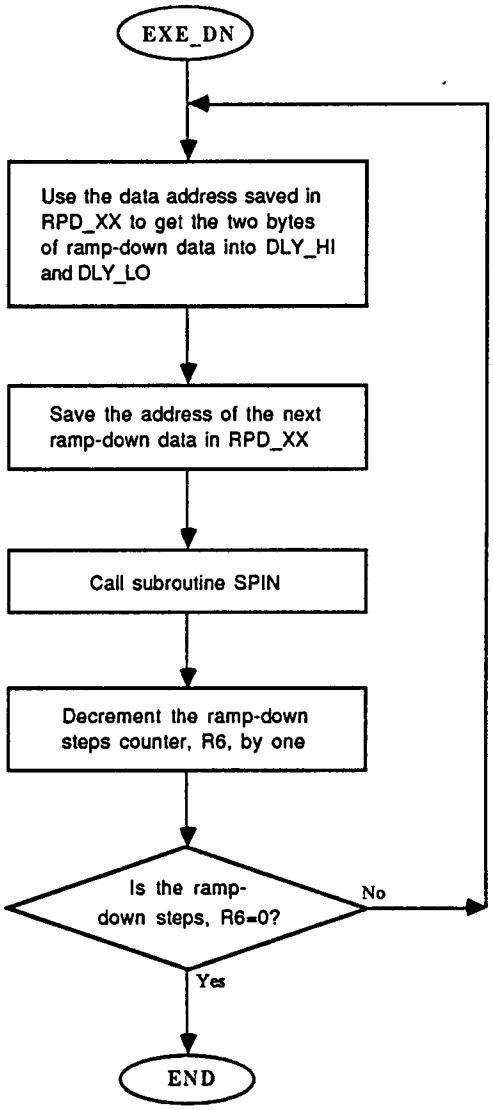
Flowchart of subroutine DN_RAMP of ROX.SRC



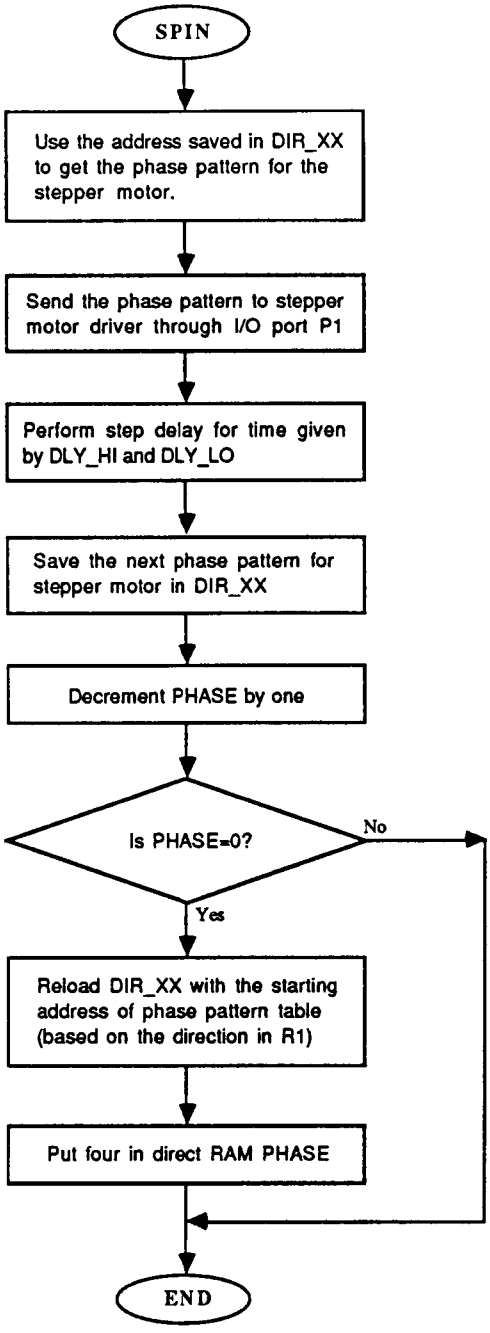
Flowchart of subroutine EXE_UP of ROX.SRC



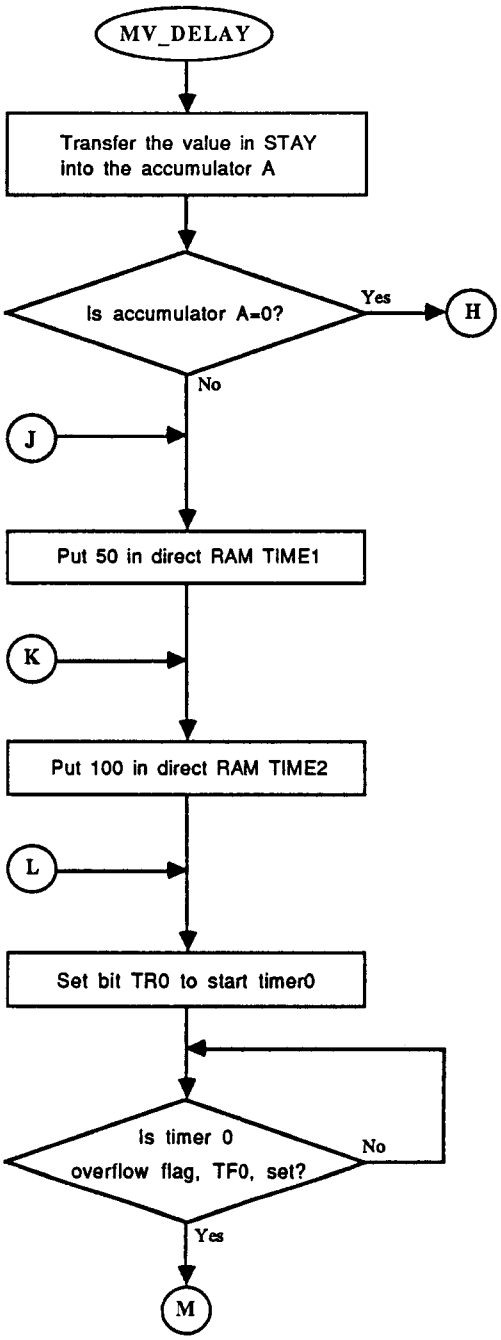
Flowchart of subroutine EXE_CS of ROX.SRC



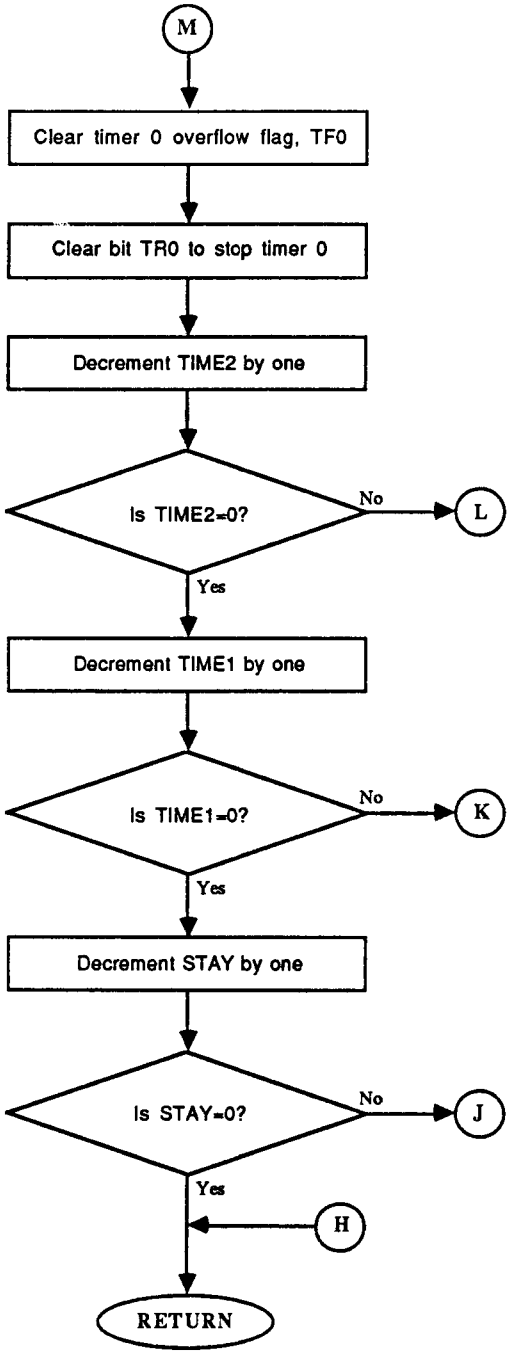
Flowchart of subroutine EXE_DN of ROX.SRC



Flowchart of subroutine SPIN of ROX.SRC



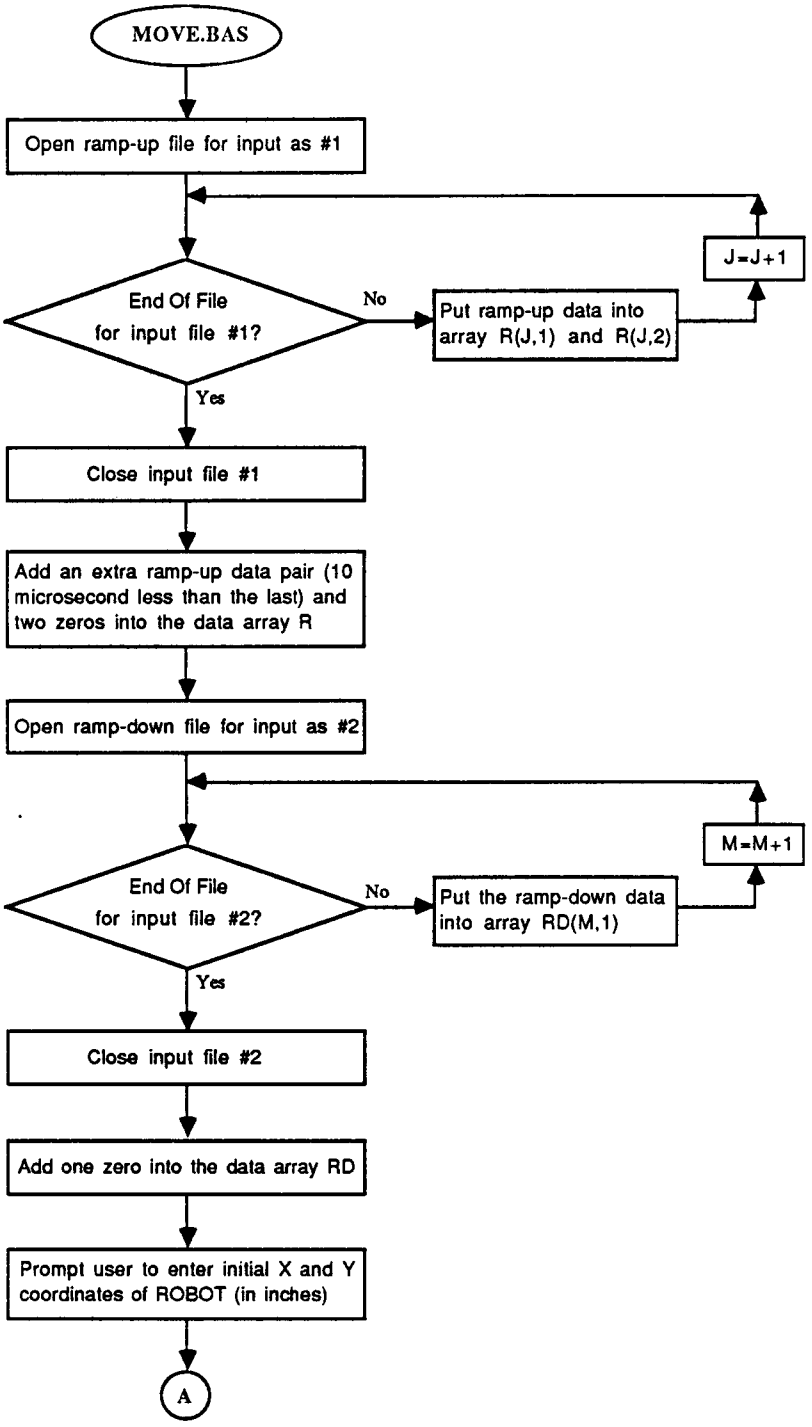
Flowchart of subroutine MV_DELAY of ROX.SRC



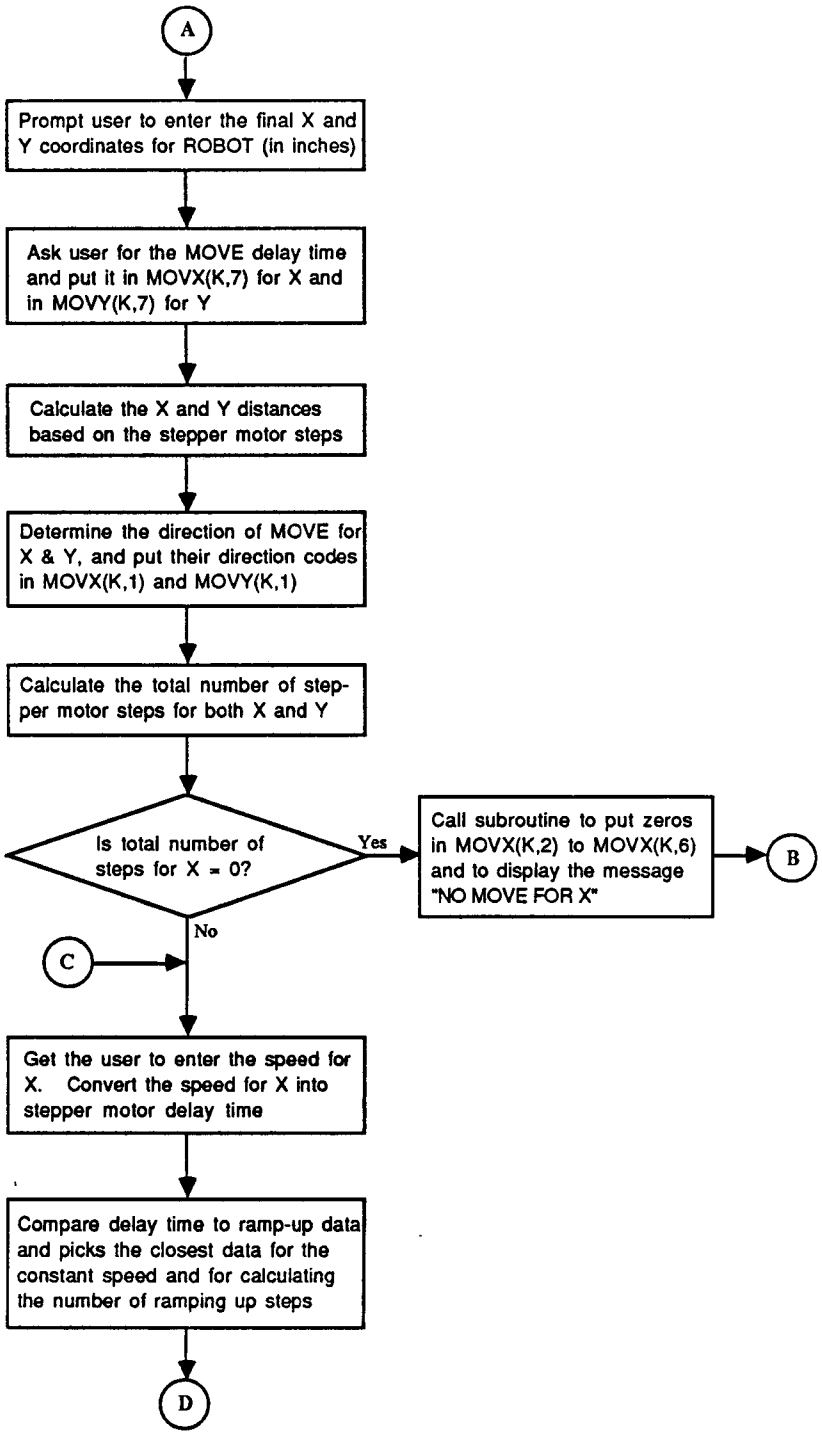
Flowchart of subroutine MV_DELAY of ROX.SRC (Continued)

APPENDIX E

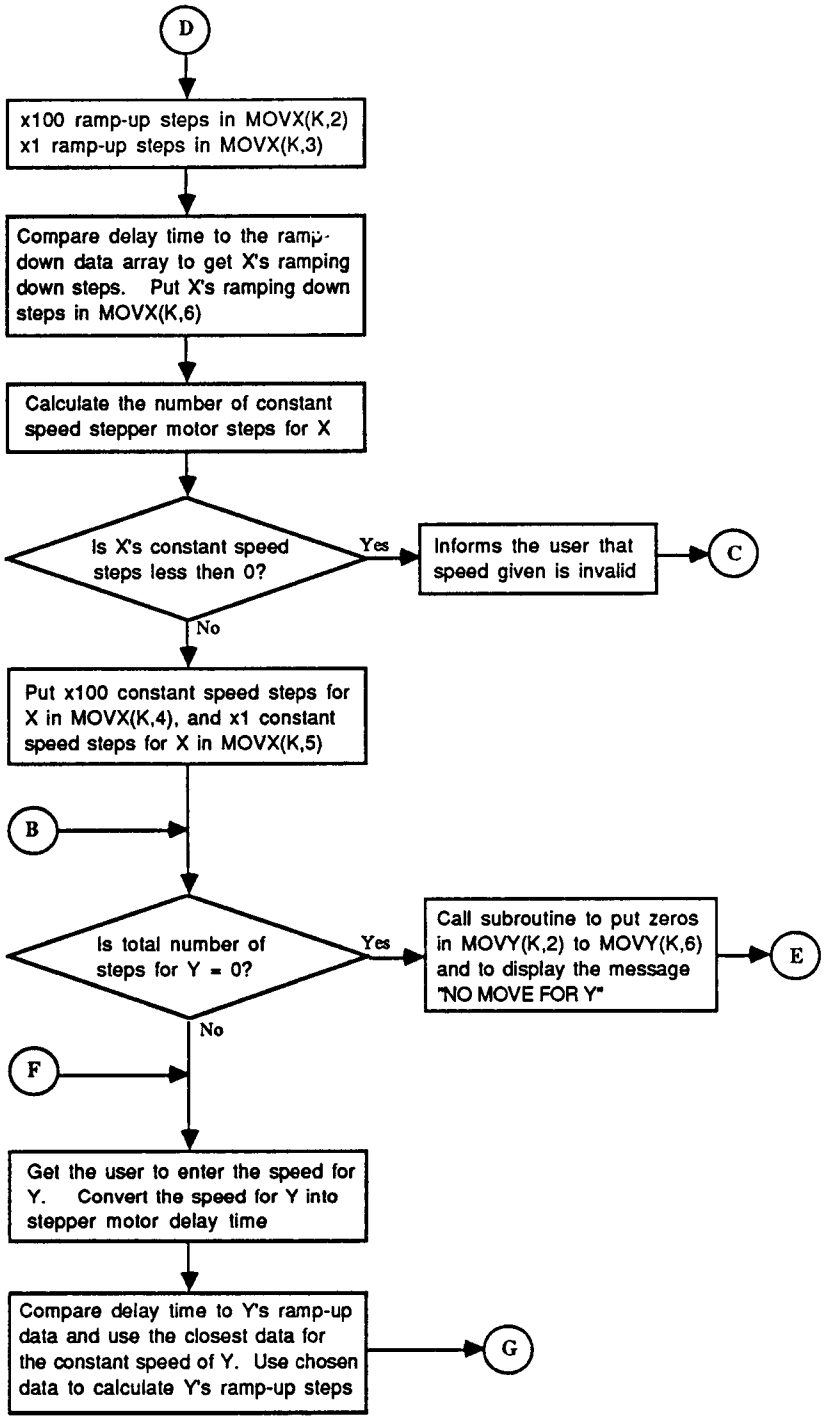
Flowcharts of the BASIC Program MOVE



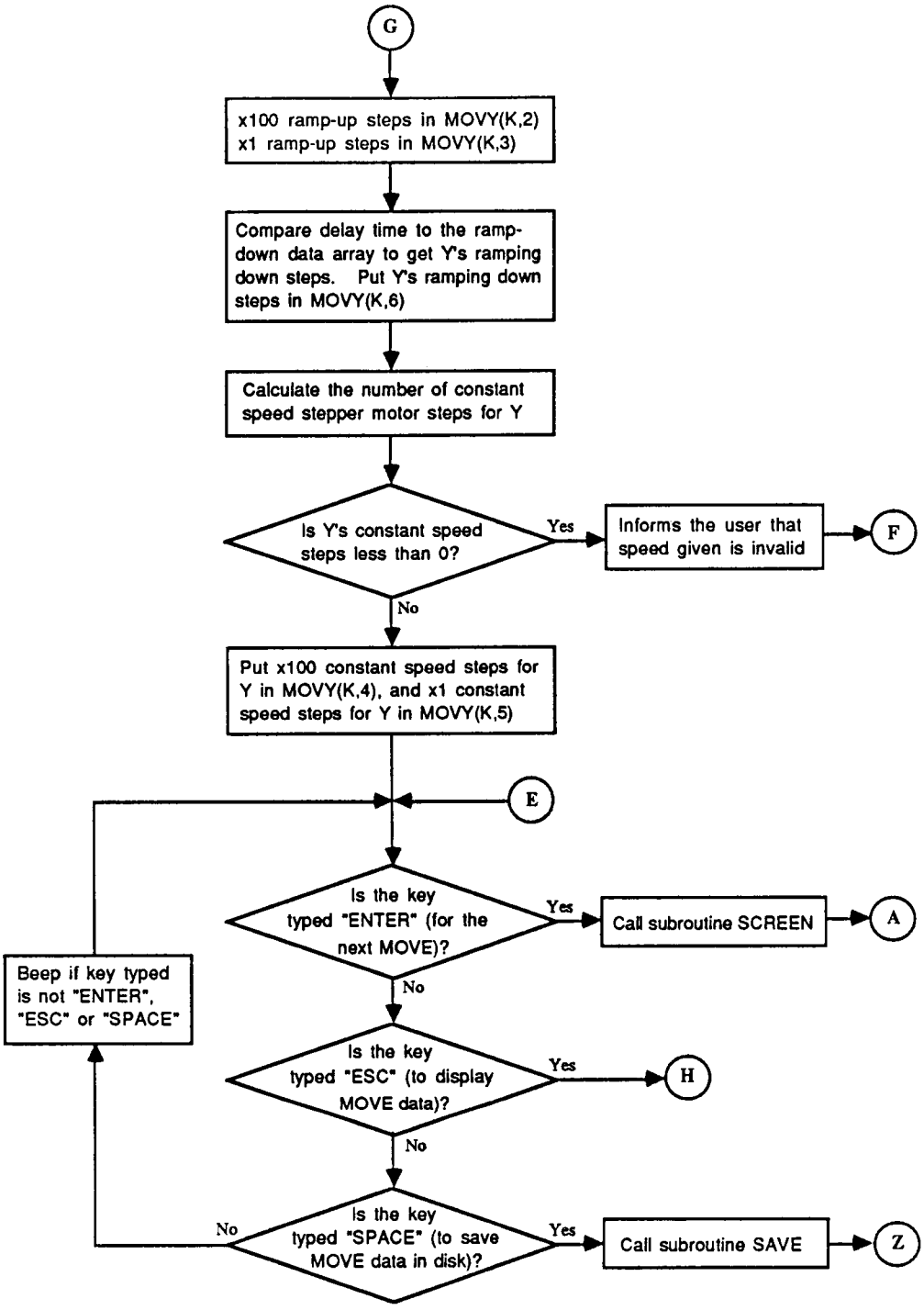
Flowchart of MOVE.BAS



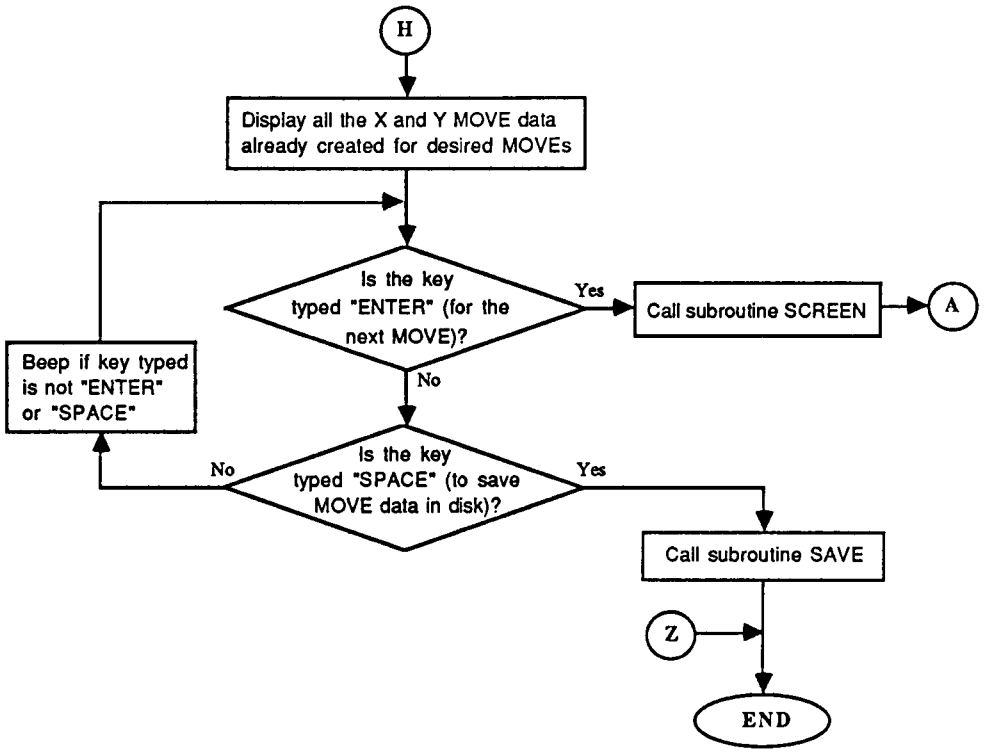
Flowchart of MOVE.BAS (Continued)



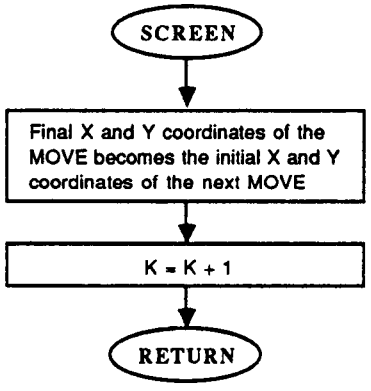
Flowchart of MOVE.BAS (Continued)



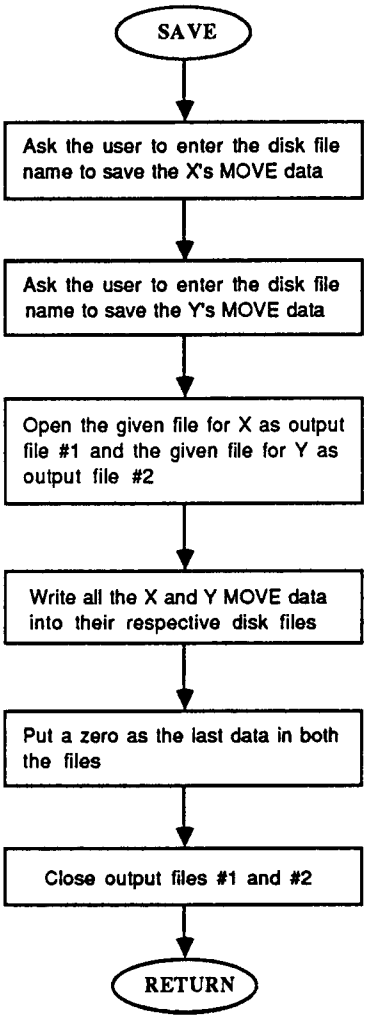
Flowchart of MOVE.BAS (Continued)



Flowchart of MOVE.BAS (Continued)



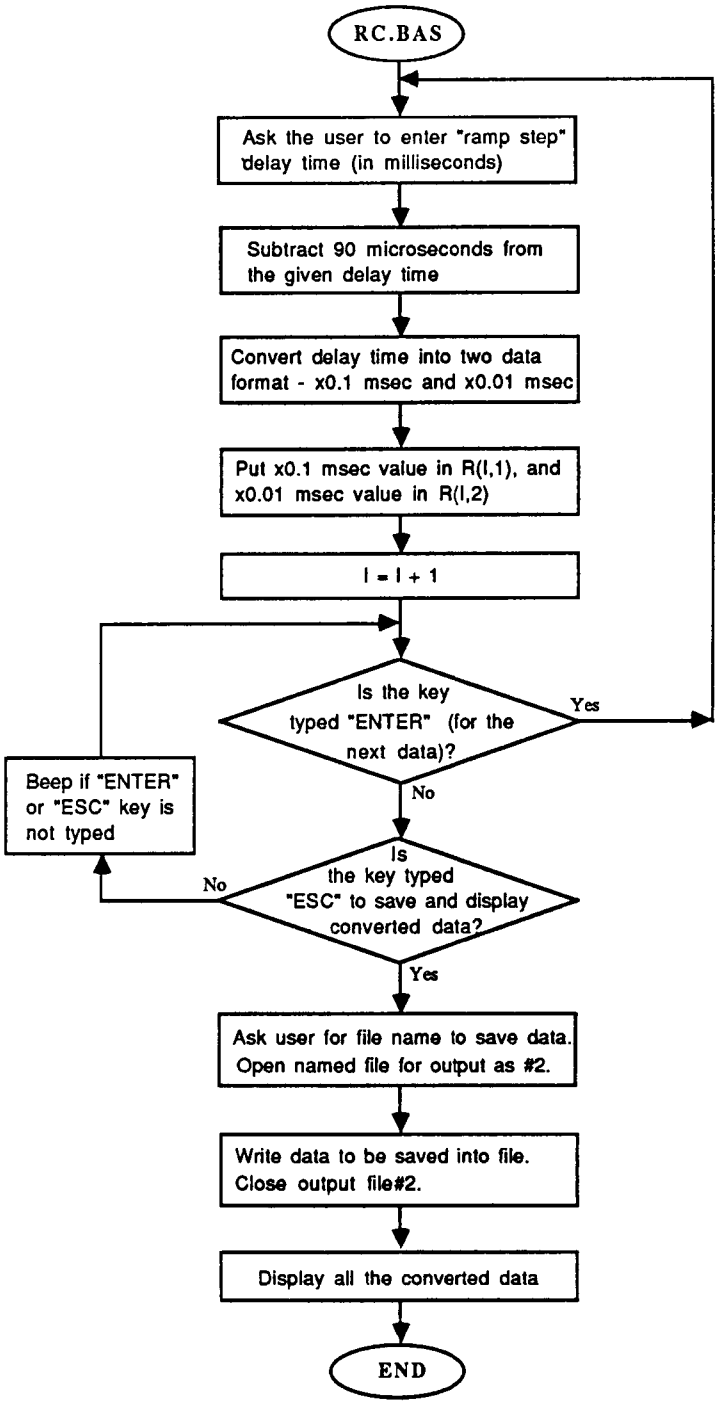
Flowchart of subroutine SCREEN of MOVE.BAS



Flowchart of subroutine SAVE of MOVE.BAS

APPENDIX F

Flowchart of the BASIC Program RC



Flowchart of RC.BAS

APPENDIX G

Listing of BASIC Program ROBOT3.BAS

```
10    '*************************************************************************
20    '**                                                                     **
30    '**                  ROBOT3.bas  by  Siak Thong Yeo                     **
40    '**                          ( 3/9/1988 )                               **
50    '**                                                                     **
60    '**       NOTE:    This program needs to be compiled by a BASIC        **
70    '**                Compiler because it involves the calling of         **
80    '**                Assembly Language subroutines to perform            **
90    '**                Serial transmission of data to the 8031.            **
100   '**                                                                     **
110   '*************************************************************************
120   '
130    OPTION BASE 1  'lowest value of array subscript is 1
140    DIM M(635), R(1145)
150   ADRSX%=31     'address of X-axis is hex 1F
160   ADRSY%=47     'address of Y-axis is hex 2F
170   ADRSZ%=79     'address of Z-axis is hex 4F
180   ROBOT%=246    'code for ROBOT data is hex F6
190   MOVE%=243     'code for MOVE data is hex F3 .
200   RAMP%=245     'code for RAMP data is hex F5
210   EXECU%=7         'code for move EXECUTION
220   RESETT%=224      'code for RESET
230   MOVEDAT%=10      'code for 1st MOVE data
240   RMPDAT1%=3       'code for RAMP-UP data
250   RMPDAT2%=4       'code for RAMP-DOWN RAMP data
260   CLS
270   COLOR 15,0
280   GOSUB 4240      'put window on screen
290   GOSUB 4740      'put title and name on screen
300       REM *** Informing the user which key to use  ***
310   LOCATE 20,30,0
320   COLOR 27,7 : PRINT " ENTER ";
330   COLOR 15,0 : PRINT "  to CONTINUE "
340   LOCATE 22,30 : COLOR 10,7: PRINT "  ESC  ";
350   COLOR 15,0  : PRINT "  to QUIT "
360   A$=INKEY$
370   IF A$= ""  THEN 360
380   IF LEN(A$) <> 1 THEN 360   'get key again if more than 1 character
390   IF ASC(A$) = 13 THEN FLAG=1 : GOTO 430   'ENTER so continue
400   IF ASC(A$) = 27 THEN 430                 'ESC so quit
410   BEEP
420   GOTO 360
430   IF FLAG <> 1 THEN GOTO 4960    'quit if flag is not set
440   GOSUB 1950
450   IF EXFLAG=1 THEN GOSUB 2180 : GOTO 440   'EXECUTE???
460   IF RSFLAG=1 THEN GOSUB 2460 : GOTO 440   'RESET???
470   IF DLFLAG=1 THEN GOSUB 2610 : GOTO 490   'DOWNLOAD???
480   GOTO 4960                      'Quit
490   IF RBFLAG=1 THEN GOSUB 2820 : GOTO 550   'ROBOT data??
500   IF MVFLAG=1 THEN GOSUB 2860 : GOTO 670   'MOVE data??
510   IF RPFLAG=1 THEN GOSUB 2900 : GOTO 1260  'RAMP data??
520   GOTO 440                       'goto MAIN MENU
530      REM  ***  This section of the program handles the down-  ***
540      REM  ***  loading of ROBOT data to the microcontrollers  ***
550   COLOR 11,0 : GOSUB 4370
560   GOSUB 2950
570   IF XFLAG=1  THEN GOSUB 3180 : GOTO 620   'X microcontroller??
580   IF YFLAG=1  THEN GOSUB 3430 : GOTO 620   'Y microcontroller??
590   IF ZFLAG=1  THEN GOSUB 3680 : GOTO 620   'Z microcontroller??
600   IF DLFLAG=1 THEN 470
610   GOTO 440                    'goto MAIN MENU
620   IF RBFLAG=1 THEN 490        'download another ROBOT data set
630   IF DLFLAG=1 THEN 470        'goto DOWNLOAD menu
```

```
640  GOTO 440
650      REM  ***  This section of the program handles the down- ***
660      REM  ***  loading of MOVE data to the microcontrollers  ***
670  COLOR 11,0 : GOSUB 4370
680  GOSUB 2950
690  IF XFLAG=1  THEN 740
700  IF YFLAG=1  THEN 810
710  IF ZFLAG=1  THEN 880
720  IF DLFLAG=1 THEN 470
730  GOTO 440                        'goto MAIN MENU
740  CLS : LOCATE 2,16 : COLOR 10,5
750  PRINT "  DOWNLOAD MOVE DATA FOR X-AXIS MICROCONTROLLER  "
760  FIRST%=ADRSX% AND MOVE%
770  GOSUB 3940                'to get MOVE data set from user
780  CLS : LOCATE 2,16 : COLOR 10,5
790  PRINT "  DOWNLOAD MOVE DATA FOR X-AXIS MICROCONTROLLER  "
800  GOTO 980
810  CLS : LOCATE 2,16 : COLOR 10,3
820  PRINT "  DOWNLOAD MOVE DATA FOR Y-AXIS MICROCONTROLLER  "
830  FIRST%=ADRSY% AND MOVE%
840  GOSUB 3940                'to get MOVE data set from user
850  CLS : LOCATE 2,16 : COLOR 10,3
860  PRINT "  DOWNLOAD MOVE DATA FOR Y-AXIS MICROCONTROLLER  "
870  GOTO 980
880  CLS : LOCATE 2,16 : COLOR 10,6
890  PRINT "  DOWNLOAD MOVE DATA FOR Z-AXIS MICROCONTROLLER  "
900  FIRST%=ADRSZ% AND MOVE%
910  GOSUB 3940                'to get MOVE data set from user
920  CLS : LOCATE 2,16 : COLOR 10,6
930  PRINT "  DOWNLOAD MOVE DATA FOR Z-AXIS MICROCONTROLLER  "
940     REM  ***  This part of the program transmit the MOVE  data  ***
950     REM  ***  set to the addressed microcontroller by calling   ***
960     REM  ***  the routines ADDRESS and XMIT of the assembly     ***
970     REM  ***  language program ROBOT.ASM.                       ***
980   SECAN%=MOVEDAT%
990  HX1$=HEX$(FIRST%)
1000  HX2$=HEX$(SECAN%)
1010  LOCATE 6,17 : COLOR 10,3
1020  PRINT "  ADDRESS code = ";HX1$;" Hex      DATA code = ";HX2$;" Hex  "
1030  CALL ADDRESS(FIRST%, SECAN%)
1040  FOR TX=1 TO K-1
1050  DATOUT%=M(TX)
1060  CALL XMIT(DATOUT%)
1070  NEXT
1080  LOCATE 12,25 : COLOR 15,3
1090  PRINT "   MOVE data set sent to robot   "
1100  LOCATE 18,20 : COLOR 11,7 : PRINT " ENTER ";
1110  COLOR 15,0 : PRINT " to download another set of MOVE data"
1120  LOCATE 20,20 : COLOR 11,7 : PRINT " SPACE ";
1130  COLOR 15,0 : PRINT " to return to DOWNLOAD menu"
1140  LOCATE 22,20 : COLOR 11,7 : PRINT "  ESC  ";
1150  COLOR 15,0 : PRINT " to return to MAIN MENU"
1160  DLFLAG=0 : MVFLAG=0
1170  Z1$=INKEY$
1180  IF Z1$="" THEN 1170
1190  IF LEN(Z1$) <> 1 THEN 1170
1200  IF ASC(Z1$) = 13 THEN MVFLAG=1 : GOTO 500
1210  IF ASC(Z1$) = 27 THEN 440
1220  IF ASC(Z1$) = 32 THEN DLFLAG=1 : GOTO 470
1230  BEEP : GOTO 1170
1240     REM  ***  This section of the program handles the down- ***
1250     REM  ***  loading of RAMP data to the microcontrollers  ***
1260  COLOR 11,0 : GOSUB 4370
```

```
1270  GOSUB 2950
1280  IF XFLAG=1  THEN 1330
1290  IF YFLAG=1  THEN 1400
1300  IF ZFLAG=1  THEN 1470
1310  IF DLFLAG=1 THEN 470
1320  GOTO 440                    'goto MAIN MENU
1330  CLS : LOCATE 2,16 : COLOR 14,5
1340  PRINT "  DOWNLOAD RAMP DATA FOR X-AXIS MICROCONTROLLER  "
1350  FIRST%=ADRSX% AND RAMP%
1360  GOSUB 4050              'select the RAMP? data for downloading
1370  CLS : LOCATE 2,16 : COLOR 14,5
1380  PRINT "  DOWNLOAD RAMP DATA FOR X-AXIS MICROCONTROLLER  "
1390  GOTO 1530
1400  CLS : LOCATE 2,16 : COLOR 14,3
1410  PRINT "  DOWNLOAD RAMP DATA FOR Y-AXIS MICROCONTROLLER  "
1420  FIRST%=ADRSY% AND RAMP%
1430  GOSUB 4050              'select the RAMP? data for downloading
1440  CLS : LOCATE 2,16 : COLOR 14,3
1450  PRINT "  DOWNLOAD RAMP DATA FOR Y-AXIS MICROCONTROLLER  "
1460  GOTO 1530
1470  CLS : LOCATE 2,16 : COLOR 14,6
1480  PRINT "  DOWNLOAD RAMP DATA FOR Z-AXIS MICROCONTROLLER  "
1490  FIRST%=ADRSZ% AND RAMP%
1500  GOSUB 4050              'select the RAMP? data for downloading
1510  CLS : LOCATE 2,16 : COLOR 14,6
1520  PRINT "  DOWNLOAD RAMP DATA FOR Z-AXIS MICROCONTROLLER  "
1530  IF R1FLAG=1 THEN SECAN%=RMPDAT1% : GOTO 1590
1540  IF R2FLAG=1 THEN SECAN%=RMPDAT2% : GOTO 1590
1550  IF DLFLAG=1 THEN 470
1560  GOTO 440
1570     REM  ***  This part of the program transmit the RAMP data  ***
1580     REM  ***  set from a file to the microcontroller          ***
1590  LOCATE 12,20 : COLOR 15,0
1600  INPUT "Name of RAMP data file:-  ", RDAT$
1610  OPEN RDAT$ FOR INPUT AS #2
1620  TOT=1
1630  WHILE NOT EOF(2)
1640  INPUT #2, R(TOT)        'puts RAMP data into an array
1650  TOT=TOT+1
1660  WEND
1670  CLOSE
1680  HY1$=HEX$(FIRST%)
1690  HY2$=HEX$(SECAN%)
1700  LOCATE 6,17 : COLOR 10,3
1710  PRINT "  ADDRESS code = ";HY1$;" Hex      DATA code = ";HY2$;" Hex  "
1720  CALL ADDRESS(FIRST%, SECAN%)
1730  FOR TX=1 TO TOT-1
1740  DATOUT%=R(TX)
1750  CALL XMIT(DATOUT%)
1760  NEXT
1770  LOCATE 12,20 : COLOR 15,0 : PRINT "      "
1780  LOCATE 12,25 : COLOR 15,3
1790  PRINT "   RAMP data set sent to robot   "
1800  LOCATE 18,20 : COLOR 11,7 : PRINT " ENTER ";
1810  COLOR 15,0 : PRINT " to download another set of RAMP data"
1820  LOCATE 20,20 : COLOR 11,7 : PRINT " SPACE ";
1830  COLOR 15,0 : PRINT " to return to DOWNLOAD menu"
1840  LOCATE 22,20 : COLOR 11,7 : PRINT "  ESC  ";
1850  COLOR 15,0 : PRINT " to return to MAIN MENU"
1860  DLFLAG=0 : RPFLAG=0
1870  Z2$=INKEY$
1880  IF Z2$="" THEN 1870
1890  IF LEN(Z2$) <> 1 THEN 1870
```

```
1900  IF ASC(Z2$) = 13 THEN RPFLAG=1 : GOTO 510
1910  IF ASC(Z2$) = 27 THEN 440
1920  IF ASC(Z2$) = 32 THEN DLFLAG=1 : GOTO 470
1930  BEEP : GOTO 1870
1940      REM ***  Subroutine MENU to put main menu on the screen  ***
1950  COLOR 7,0 : CLS
1960  COLOR 14,0 : GOSUB 4370
1970  LOCATE 2,32 : COLOR 14,3 : PRINT "   MAIN  MENU   "
1980  LOCATE 5,28 : COLOR 14,7 : PRINT " 1 ";
1990  COLOR  15,0 : PRINT "  to EXECUTE robot"
2000  LOCATE 7,28 : COLOR 14,7 : PRINT " 2 ";
2010  COLOR  15,0 : PRINT "  to RESET robot system"
2020  LOCATE 9,28 : COLOR 14,7 : PRINT " 3 ";
2030  COLOR  15,0  : PRINT "  for DOWNLOAD menu"
2040  LOCATE 20,28 : COLOR 14,7 : PRINT "  ESC  ";
2050  COLOR 11,0 : PRINT "  to QUIT"
2060  EXFLAG=0 : RSFLAG=0 : DLFLAG=0
2070  B$=INKEY$
2080  IF B$="" THEN 2070
2090  IF LEN(B$) <> 1 THEN 2070
2100  IF ASC(B$) = 27 THEN RETURN
2110  IF ASC(B$) = 49 THEN EXFLAG=1 : RETURN
2120  IF ASC(B$) = 50 THEN RSFLAG=1 : RETURN
2130  IF ASC(B$) = 51 THEN DLFLAG=1 : RETURN
2140  BEEP : GOTO 2070
2150     REM  ***  Subroutine EXECUTE to send the EXECUTION code to  ***
2160     REM  ***  the robot system (X, Y, and Z microcontrollers)   ***
2170     REM  ***  so that it will start to execute the move(s).     ***
2180  CLS
2190  LOCATE 2,10  : COLOR 11,2 : PRINT "   EXECUTION   "
2200  LOCATE 18,30 : COLOR 27,7 : PRINT " ENTER ";
2210  COLOR 10,0 : PRINT " to EXECUTE move"
2220  LOCATE 20,30 : COLOR 11,7 : PRINT "  ESC  ";
2230  COLOR 10,0 : PRINT " to return to MAIN MENU"
2240  C$=INKEY$
2250  IF C$="" THEN 2240
2260  IF LEN(C$) <> 1 THEN 2240
2270  IF ASC(C$) = 13 THEN GOSUB 2320 : GOTO 2240
2280  IF ASC(C$) = 27 THEN RETURN
2290  BEEP : GOTO 2240
2300     REM  ***  Subroutine that calls the assembly language    ***
2310     REM  ***  routine to transmit the EXECUTE command code.  ***
2320  LOCATE 12,22 : PRINT "                                      "
2330  CALL COMMAND(EXECU%)
2340  FOR DELAY=1 TO 500 : NEXT DELAY
2350  COLOR 15,0 : LOCATE 11,21
2360  FOR BOX=21 TO 62 : PRINT "▄"; : NEXT BOX
2370  LOCATE 12,21 : PRINT "▐"
2380  LOCATE 12,62 : PRINT "▌"
2390  LOCATE 13,21 : FOR BOX=21 TO 62 : PRINT "▀"; : NEXT BOX
2400  COLOR 14,1
2410  LOCATE 12,22 : PRINT "  EXECUTE command transmitted to ROBOT  "
2420  LOCATE 18,54 : COLOR 26,0 : PRINT "again"
2430  RETURN
2440     REM  ***  Subroutine RESET that allows for the RESET code  ***
2450     REM  ***  to be transmitted to all the microcontrollers    ***
2460  CLS
2470  LOCATE 2,10  : COLOR 14,2 : PRINT "  SYSTEM RESET  "
2480  LOCATE 18,30 : COLOR 30,7 : PRINT " ENTER ";
2490  COLOR 10,0 : PRINT " to RESET robot system"
2500  LOCATE 20,30 : COLOR 14,7 : PRINT "  ESC  ";
2510  COLOR 10,0 : PRINT " to return to MAIN MENU "
2520  FLAG=0
```

```
2530  D$=INKEY$
2540  IF D$="" THEN 2530
2550  IF LEN(D$) <> 1 THEN 2530
2560  IF ASC(D$) = 13 THEN CALL COMMAND(RESETT%) : RETURN
2570  IF ASC(D$) = 27 THEN RETURN
2580  BEEP : GOTO 2530
2590     REM  ***  Subroutine DOWNLOAD to put the download menu   ***
2600     REM  ***  (ROBOT, MOVE or RAMP data type) on the screen  ***
2610  COLOR 7,0 : CLS
2620  LOCATE 2,10 : COLOR 15,2
2630  PRINT "  DOWNLOAD MENU  "  : GOSUB 4360
2640  LOCATE 5,28 : COLOR 15,7 : PRINT " 1 ";
2650  COLOR 14,0  : PRINT "  to download ROBOT data"
2660  LOCATE 7,28 : COLOR 15,7 : PRINT " 2 ";
2670  COLOR  14,0 : PRINT "  to download MOVE data"
2680  LOCATE 9,28 : COLOR 15,7 : PRINT " 3 ";
2690  COLOR  14,0  : PRINT "  to download RAMP data"
2700  LOCATE 22,28 : COLOR 15,7 : PRINT "  ESC  ";
2710  COLOR 10,0   : PRINT "  to return to MAIN MENU"
2720  RBFLAG=0 : MVFLAG=0 : RPFLAG=0        'reset flags
2730  E1$=INKEY$
2740  IF E1$="" THEN GOTO 2730
2750  IF LEN(E1$) <> 1 THEN 2730
2760  IF ASC(E1$) = 27 THEN RETURN
2770  IF ASC(E1$) = 49 THEN RBFLAG=1 : RETURN
2780  IF ASC(E1$) = 50 THEN MVFLAG=1 : RETURN
2790  IF ASC(E1$) = 51 THEN RPFLAG=1 : RETURN
2800  BEEP : GOTO 2730
2810     REM  ***  Subroutine to put title on the screen (ROBOT)  ***
2820  CLS : LOCATE 2,10
2830  COLOR 11,4 : PRINT "  DOWNLOAD ROBOT DATA  "
2840  RETURN
2850     REM  ***  Subroutine to put title on the screen (MOVE)  ***
2860  CLS : LOCATE 2,10
2870  COLOR 11,5 : PRINT "  DOWNLOAD MOVE DATA  "
2880  RETURN
2890     REM  ***  Subroutine to put title on the screen (RAMP)  ***
2900  CLS : LOCATE 2,10
2910  COLOR 11,6 : PRINT "  DOWNLOAD RAMP DATA  "
2920  RETURN
2930     REM  ***  Subroutine AXIS that allows the user to select the  ***
2940     REM  ***  X, Y, or Z microcontroller for the data to be sent  ***
2950  LOCATE 5,25 : COLOR 11,7 : PRINT " 1 ";
2960  COLOR 15,0  : PRINT "  for X-axis Microcontroller"
2970  LOCATE 7,25 : COLOR 11,7 : PRINT " 2 ";
2980  COLOR 15,0  : PRINT "  for Y-axis Microcontroller"
2990  LOCATE 9,25 : COLOR 11,7 : PRINT " 3 ";
3000  COLOR 15,0   : PRINT "  for Z-axis Microcontroller"
3010  LOCATE 20,25 : COLOR 11,7 : PRINT " SPACE ";
3020  COLOR 15,0   : PRINT "  to return to DOWNLOAD menu"
3030  LOCATE 22,25 : COLOR 11,7 : PRINT "  ESC  ";
3040  COLOR 15,0   : PRINT "  to return to MAIN MENU"
3050  XFLAG=0 : YFLAG=0 : ZFLAG=0 : DLFLAG=0        'reset flags
3060  E2$=INKEY$
3070  IF E2$="" THEN GOTO 3060
3080  IF LEN(E2$) <> 1 THEN 3060
3090  IF ASC(E2$) = 27 THEN RETURN
3100  IF ASC(E2$) = 32 THEN DLFLAG=1 : RETURN
3110  IF ASC(E2$) = 49 THEN XFLAG=1  : RETURN
3120  IF ASC(E2$) = 50 THEN YFLAG=1  : RETURN
3130  IF ASC(E2$) = 51 THEN ZFLAG=1  : RETURN
3140  BEEP : GOTO 3060
3150     REM  ***  Subroutine XROBOT to get the number of repeti-  ***
```

```
3160     REM  ***  tions for the "moves" from the user to be sent  ***
3170     REM  ***  to the X-Microcontroller as ROBOT data          ***
3180  CLS : LOCATE 2,16 : COLOR 14,5
3190  PRINT "  DOWNLOAD ROBOT DATA FOR X-AXIS MICROCONTROLLER  "
3200  FIRST% = ADRSX% AND ROBOT%
3210  LOCATE 10,20 : COLOR 15,0
3220  INPUT "Number of repetitions"; REPS%
3230  CALL ROBOT(FIRST%, REPS%)
3240  LOCATE 14,20 : COLOR 0,2
3250  PRINT " ROBOT data downloaded to X-axis Microcontroller "
3260  LOCATE 18,20 : COLOR 15,7 : PRINT " ENTER ";
3270  COLOR 10,0 : PRINT "  to download another set of ROBOT data"
3280  LOCATE 20,20 : COLOR 15,7 : PRINT " SPACE ";
3290  COLOR 10,0 : PRINT "  to return to DOWNLOAD menu"
3300  LOCATE 22,20 : COLOR 15,7 : PRINT "  ESC  ";
3310  COLOR 10,0 : PRINT "  to return to MAIN MENU "
3320  RBFLAG=0 : DLFLAG=0              'reset flags
3330  E3$=INKEY$
3340  IF E3$="" THEN GOTO 3330
3350  IF LEN(E3$) <> 1 THEN 3330
3360  IF ASC(E3$) = 13 THEN RBFLAG=1 : RETURN
3370  IF ASC(E3$) = 27 THEN RETURN
3380  IF ASC(E3$) = 32 THEN DLFLAG=1 : RETURN
3390  BEEP : GOTO 3330
3400     REM  ***  Subroutine YROBOT to get the number of repeti-  ***
3410     REM  ***  tions for the "moves" from the user to be sent  ***
3420     REM  ***  to the Y-Microcontroller as ROBOT data          ***
3430  CLS : LOCATE 2,16 : COLOR 1,7
3440  PRINT "  DOWNLOAD ROBOT DATA FOR Y-AXIS MICROCONTROLLER  "
3450  FIRST% = ADRSY% AND ROBOT%
3460  LOCATE 10,20 : COLOR 15,0
3470  INPUT "Number of repetitions"; REPS%
3480  CALL ROBOT(FIRST%, REPS%)
3490  LOCATE 14,20 : COLOR 0,2
3500  PRINT " ROBOT data downloaded to Y-axis Microcontroller "
3510  LOCATE 18,20 : COLOR 15,7 : PRINT " ENTER ";
3520  COLOR 10,0 : PRINT "  to download another set of ROBOT data"
3530  LOCATE 20,20 : COLOR 15,7 : PRINT " SPACE ";
3540  COLOR 10,0 : PRINT "  to return to DOWNLOAD menu"
3550  LOCATE 22,20 : COLOR 15,7 : PRINT "  ESC  ";
3560  COLOR 10,0 : PRINT "  to return to MAIN MENU "
3570  RBFLAG=0 : DLFLAG=0
3580  E4$=INKEY$
3590  IF E4$="" THEN GOTO 3580
3600  IF LEN(E4$) <> 1 THEN 3580
3610  IF ASC(E4$) = 13 THEN RBFLAG=1 : RETURN
3620  IF ASC(E4$) = 27 THEN RETURN
3630  IF ASC(E4$) = 32 THEN DLFLAG=1 : RETURN
3640  BEEP : GOTO 3580
3650     REM  ***  Subroutine ZROBOT to get the number of repeti-  ***
3660     REM  ***  tions for the "moves" from the user to be sent  ***
3670     REM  ***  to the Z-Microcontroller as ROBOT data          ***
3680  CLS : LOCATE 2,16 : COLOR 12,1
3690  PRINT "  DOWNLOAD ROBOT DATA FOR Z-AXIS MICROCONTROLLER  "
3700  FIRST% = ADRSZ% AND ROBOT%
3710  LOCATE 10,20 : COLOR 15,0
3720  INPUT "Number of repetitions"; REPS%
3730  CALL ROBOT(FIRST%, REPS%)
3740  LOCATE 14,20 : COLOR 0,2
3750  PRINT " ROBOT data downloaded to Z-axis Microcontroller "
3760  LOCATE 18,20 : COLOR 15,7 : PRINT " ENTER ";
3770  COLOR 10,0 : PRINT "  to download another set of ROBOT data"
3780  LOCATE 20,20 : COLOR 15,7 : PRINT " SPACE ";
```

```
3790  COLOR 10,0 : PRINT "  to return to DOWNLOAD menu"
3800  LOCATE 22,20 : COLOR 15,7 : PRINT "  ESC  ";
3810  COLOR 10,0 : PRINT "  to return to MAIN MENU "
3820  RBFLAG=0 : DLFLAG=0
3830  E5$=INKEY$
3840  IF E5$="" THEN GOTO 3830
3850  IF LEN(E5$) <> 1 THEN 3830
3860  IF ASC(E5$) = 13 THEN RBFLAG=1 : RETURN
3870  IF ASC(E5$) = 27 THEN RETURN
3880  IF ASC(E5$) = 32 THEN DLFLAG=1 : RETURN
3890  BEEP : GOTO 3830
3900     REM  ***  Subroutine LOADMOVE that gets the name of the   ***
3910     REM  ***  MOVE data file from the user, and then put the  ***
3920     REM  ***  MOVE data into an array M(K) before they are    ***
3930     REM  ***  being transmitted to the microcontroller        ***
3940  LOCATE 12,20 : COLOR 15,0
3950  INPUT "Name of MOVE data file:-  ", MDAT$
3960  OPEN MDAT$ FOR INPUT AS #3
3970  K=1
3980  WHILE NOT EOF(3)
3990  INPUT #3, M(K)
4000  K=K+1
4010  WEND
4020  CLOSE
4030  RETURN
4040     REM  ***  Subroutine RAMPTYPE to display the RAMP data type  ***
4050  COLOR 14,0 : GOSUB 4490
4060  LOCATE 7,22 : COLOR 14,7 : PRINT " 1 ";
4070  COLOR 15,0 : PRINT " to download RAMP-UP data "
4080  LOCATE 11,22 : COLOR 14,7 : PRINT " 2 ";
4090  COLOR 15,0 : PRINT " to download RAMP-DOWN data"
4100  LOCATE 20,22 : COLOR 14,7 : PRINT " SPACE ";
4110  COLOR 15,0 : PRINT "  to return to DOWNLOAD menu"
4120  LOCATE 22,22 : COLOR 14,7 : PRINT "  ESC  ";
4130  COLOR 15,0 : PRINT "  to return to MAIN MENU"
4140  R1FLAG=0 : R2FLAG=0 : DLFLAG=0          'reset flags
4150  F2$=INKEY$
4160  IF F2$="" THEN 4150
4170  IF LEN(F2$) <> 1 THEN 4150
4180  IF ASC(F2$) = 27 THEN RETURN
4190  IF ASC(F2$) = 32 THEN DLFLAG=1 : RETURN
4200  IF ASC(F2$) = 49 THEN R1FLAG=1 : RETURN
4210  IF ASC(F2$) = 50 THEN R2FLAG=1 : RETURN
4220  BEEP : GOTO 4150
4230     REM  ***  Subroutine to draw a window on the screen  ***
4240  CLS : COLOR 15,0
4250  LOCATE 4,10: PRINT "╔";
4260  GOSUB 4600
4270  LOCATE 4,70 : PRINT "╗"
4280  FOR J=1 TO 10
4290  LOCATE 4+J,10 : PRINT "║" : : LOCATE ,70 : PRINT "║"
4300  NEXT J
4310  LOCATE 15,10 :PRINT "╚";
4320  GOSUB 4600
4330  LOCATE 15,70 : PRINT "╝"
4340  RETURN
4350     REM  ***  Subroutine to draw window on the screen  ***
4360  COLOR 15,0
4370  LOCATE 3,10: PRINT "╔";
4380  GOSUB 4600
4390  LOCATE 3,70 : PRINT "╗"
4400  FOR J=1 TO 7
4410  LOCATE 3+J,10 : PRINT "║" : : LOCATE ,70 : PRINT "║"
```

```
4420  NEXT J
4430  LOCATE 11,10 :PRINT "╚";
4440  GOSUB 4600
4450  LOCATE 11,70 : PRINT "╝"
4460  RETURN
4470     REM  ***  Subroutine to draw window on the screen  ***
4480  COLOR 15,0
4490  LOCATE 3,16: PRINT "╔";
4500  GOSUB 4660
4510  LOCATE 3,64 : PRINT "╗"
4520  FOR J=1 TO 12
4530  LOCATE 3+J,16 : PRINT "║" ; : LOCATE ,64 : PRINT "║"
4540  NEXT J
4550  LOCATE 15,16 :PRINT "╚";
4560  GOSUB 4660
4570  LOCATE 15,64: PRINT "╝"
4580  RETURN
4590     REM  ***  Subroutine to put "═" on the screen  ***
4600  FOR I=1 TO 59
4610  LOCATE ,10+I
4620  PRINT"═";
4630  NEXT
4640  RETURN
4650     REM  ***  Subroutine to put "═" on the screen  ***
4660  FOR I=1 TO 48
4670  LOCATE ,16+I
4680  PRINT"═";
4690  NEXT
4700  RETURN
4710     REM  *** Subroutine to put program's name and  ***
4720     REM  ***  writer's name with green and blue    ***
4730     REM  ***   background inside the window.       ***
4740  LOCATE 7,21 : COLOR 0,2
4750  GOSUB 4910
4760  LOCATE 8,21
4770  GOSUB 4910
4780  LOCATE 9,21 : COLOR 0,2 : PRINT "    ";
4790  COLOR 15,1 : PRINT "      Downloading Program       ";
4800  COLOR 0,2 : PRINT "    "
4810  LOCATE 10,21 : PRINT "    ";
4820  COLOR 15,1 : PRINT "       by Siak Thong Yeo        ";
4830  COLOR 0,2 : PRINT "    "
4840  LOCATE 11,21 : COLOR 0,2
4850  GOSUB 4910
4860  LOCATE 12,21
4870  GOSUB 4910
4880  COLOR 7,0
4890  RETURN
4900     REM  ***  Subroutine to put color strip on the screen  ***
4910  FOR I=1 TO 40
4920  PRINT " ";
4930  NEXT
4940  RETURN
4950     REM  ***  The end of the ROBOT program  ***
4960  COLOR 7,0
4970  CLS
4980  END
```

APPENDIX H

Listing of the Assembly Language Program ROBOT.ASM

```
                        ;*********************************************;
                        ;*********************************************;
                        ;                                             ;
                        ;                  ROBOT.ASM                  ;
                        ;                                             ;
                        ;  Subroutine to be called by basic program.  ;
                        ;  The job of this subroutine is to transmit  ;
                        ;  9-bit data (8 data bits and 1 parity bit)  ;
                        ;  through the Serial Comm. port COM1.   The  ;
                        ;  9 bit format is necessary in order to use  ;
                        ;  the multi-processors serial communication  ;
                        ;  feature of the 8031 microcomputer.         ;
                        ;                                             ;
                        ;*********************************************;
                        ;*********************************************;

                        TITLE   ASM. LANG. ROUTINES FOR ROBOT3.BAS

                        ;*********************************************
                        ;*                 Equates                   *
                        ;*********************************************

= 03F8                  THR     EQU     3F8H    ;Xmitter holding reg.
= 03FD                  LSTAT   EQU     3FDH    ;line status port no.

                        ;*********************************************
                        ;*            Define constants               *
                        ;*********************************************

0000                    CONST   SEGMENT   WORD   PUBLIC   'CONST'
0000  07                EXECU   DB    07H
0001  E0                RESETT  DB    0E0H
0002                    CONST   ENDS

                        ;*********************************************
                        ;*            Define variables               *
                        ;*********************************************

0000                    DATA    SEGMENT   WORD   PUBLIC   'DATA'
0000  ??                TEMP    DB      ?
0001                    DATA    ENDS

                        DGROUP  GROUP    DATA, CONST

                        ;*********************************************
                        ;*       Program source code segment         *
                        ;*********************************************
```

```
0000                            CODE    SEGMENT  BYTE   PUBLIC  'CODE'
                                        PUBLIC   COMMAND, ROBOT, ADDRESS, XMIT

                                        ASSUME   CS:CODE, DS:DGROUP

                                ;routine to send EXECUTE or RESET command code
                                ;to robot system (8031s)

0000                            COMMAND PROC    FAR
0000  55                                PUSH  BP
0001  8B EC                            MOV   BP, SP
0003  8B 5E 06                         MOV   BX, [BP]+6  ;get EXECUTE or RESET
0006  E8 004F R                        CALL  FIRST       ;code & then send it
0009  5D                                POP   BP
000A  CA 0002                           RET   2
000D                            COMMAND ENDP

                                ;routine to send the Address byte (includes
                                ;the data type codes), and then the two ROBOT
                                ;data (no. of reps & no. of moves)

000D                            ROBOT   PROC    FAR
000D  55                                PUSH  BP
000E  8B EC                             MOV   BP, SP
0010  8B 5E 08                         MOV   BX, [BP]+8    ;get Address byte
0013  E8 004F R                        CALL  FIRST         ;send Address byte
0016  B9 1194                           MOV   CX, 4500
0019  E2 FE                     HERE2:  LOOP  HERE2         ;delay for 9 msec
001B  8B 5E 06                         MOV   BX, [BP]+6    ;get no. of reps
001E  E8 0067 R                         CALL  SEND          ;send reps data
0021  5D                                POP   BP
0022  CA 0004                           RET   4
0025                            ROBOT   ENDP

                                ;routine to send Address byte (data type is
                                ;also encoded in this byte), then the MOVE
                                ;or RAMP type byte

0025                            ADDRESS PROC    FAR
0025  55                                PUSH  BP
0026  8B EC                            MOV   BP, SP
0028  8B 5E 08                         MOV   BX, [BP]+8
002B  E8 004F R                         CALL  FIRST       ;send address byte
002E  B9 1194                           MOV   CX, 4500
0031  E2 FE                     HERE1:  LOOP  HERE1       ;delay for 9 msec
0033  8B 5E 06                         MOV   BX, [BP]+6 ;send MOVE code or
0036  E8 0067 R                         CALL  SEND        ;RAMP type code
0039  5D                                POP   BP
003A  CA 0004                           RET   4
003D                            ADDRESS ENDP

                                ;routine to tranmit data byte to the addressed
                                ;microcontroller (8031)
```

```
003D                            XMIT    PROC    FAR
003D  55                                PUSH    BP
003E  8B EC                             MOV     BP, SP
0040  8B 5E 06                          MOV     BX, [BP]+6 ;get MOVE or RAMP data
0043  B9 1194                           MOV     CX, 4500   ;delay for 9 ms before
0046  E2 FE                     TEST2: LOOP    TEST2      ;reinitializing COM1
0048  E8 0067 R                        CALL    SEND
004B  5D                                POP     BP
004C  CA 0002                           RET     2
004F                            XMIT    ENDP

                                ;Subroutine to Xmit command/address byte with
                                ;the 9th (parity) bit equals to 1

004F                            FIRST   PROC    NEAR
004F  B4 00                             MOV     AH, 0      ;func. 0 of INT 14H to
0051  B0 9B                             MOV     AL, 9BH    ;initialize COM1 to
0053  BA 0000                           MOV     DX, 0      ;1200 BPS, even parity
0056  CD 14                             INT     14H        ;(9th bit = 1), 8 data
                                                           ;bits, 1 stop bit
0058  BA 03FD                           MOV     DX, LSTAT    ;get line status and
005B                            SEROUT1:                     ;see if the Xmitter
005B  EC                                IN      AL, DX       ;holding register is
005C  A8 20                             TEST    AL, 20H      ;empty
005E  74 FB                             JZ      SEROUT1      ;not empty - re-check
0060  BA 03F8                           MOV     DX, THR
0063  8A 07                             MOV     AL, [BX]     ;data in AL to be
0065  EE                                OUT     DX, AL       ;sent thru' THR
0066  C3                                RET
0067                            FIRST   ENDP


                                ;Subroutine to check parity before setting the
                                ;parity bit so as to force the 9th (parity) bit
                                ;to be always 0 while xmitting data

0067                            SEND    PROC    NEAR
0067  8A 07                             MOV     AL, [BX]
0069  0C 00                             OR      AL, 00H    ;use logical operation
                                                           ;to set Parity Flag
006B  24 FF                             AND     AL, 0FFH
006D  7B 0C                             JPO     ODD

                                ;parity of 8 bit data is even so init. the COM1
                                ;port to send data with even parity so that the
                                ;the 9th bit will be a zero
006F  B4 00                             MOV     AH, 0H   ;function number 0
0071  B0 9B                             MOV     AL, 9BH  ;1200 BPS, even parity,
                                                         ;1 stop bit, 8 data bits
0073  BA 0000                           MOV     DX, 0
0076  CD 14                             INT     14H    ;BIOS to initialize COM1
0078  EB 0A 90                          JMP     TXD

                                ;initialize the serial communication port
```

```
007B  B4 00                 ODD:    MOV   AH, 0H
007D  B0 8B                         MOV   AL, 8BH     ;1200 BPS, odd parity,
                                                      ;1 stop bit, 8 data bits
007F  BA 0000                       MOV   DX, 0
0082  CD 14                         INT   14H         ;BIOS to init. COM1


                            ;send a 9-bit data thru' serial port COM1
0084  BA 03FD               TXD:    MOV   DX, LSTAT    ;get line status &
0087                        SEROUT2:                       ;see if the Xmitter
0087  EC                           IN    AL, DX           ;holding register
0088  A8 20                       TEST  AL, 20H          ;is empty
008A  74 FB                 JZ      SEROUT2   ;not empty - re-check
008C  8A 07                 MOV     AL, [BX]    ;data in AL to be
008E  BA 03F8               MOV     DX, THR
0091  EE                    OUT     DX, AL           ;sent thru' THR
0092  C3                    RET
0093                        SEND    ENDP

0093                        CODE    ENDS

                                    END
```

APPENDIX I

Listing of the MCS-51 Assembly Language Program ROY.SRC

```
MCS-51 MACRO ASSEMBLER    ROY
                    03/28/88   PAGE    1


DOS 3.0 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN ROY.OBJ
ASSEMBLER INVOKED BY:  C:\M51\ASM51.EXE ROY.SRC


LOC  OBJ             LINE     SOURCE

                        1     ;***********************************************
                        2     ;**                                           **
                        3     ;**          ROY.SRC by SIAK THONG YEO        **
                        4     ;**                                           **
                        5     ;**   8031 ASSEMBLY LANGUAGE PROGRAM FOR THE  **
                        6     ;**   ROBOT SYSTEM'S Y-MICROCONTROLLER.  THIS **
                        7     ;**   PROGRAM RECEIVES DOWNLOAD, EXECUTE, AND **
                        8     ;**   RESET COMMANDS FROM THE IBM PC/AT WHICH **
                        9     ;**   IS THE ROBOT SYSTEM'S MAIN CONTROLLER.  **
                       10     ;**                                           **
                       11     ;**   DOWNLOAD -- The four types of data to be **
                       12     ;**               received by the 8031 are:-  **
                       13     ;**               RAMP-UP, RAMP-DOWN, MOVE,   **
                       14     ;**               and ROBOT.  (ROBOT is the   **
                       15     ;**               number of MOVE repetition.) **
                       16     ;**                                           **
                       17     ;**   EXECUTE  -- This command tells all 8031s **
                       18     ;**               to execute their moves (with **
                       19     ;**               the desired no. of reps.)   **
                       20     ;**                                           **
                       21     ;**   RESET    -- Initializes the Robot system **
                       22     ;**               to a known or predetermined **
                       23     ;**               location/coordinates (this  **
                       24     ;**               is not implemented at yet.) **
                       25     ;**                                           **
                       26     ;***********************************************
                       27
                       28     ;**-------------------------------------------**
                       29     ;**                EQUATES                    **
                       30     ;**-------------------------------------------**
                       31
  00E0                 32     RSET     EQU   0E0H  ;RESET code
  0007                 33     EXEC     EQU   07H   ;EXECUTE code
  0020                 34     Y_ADDR   EQU   20H   ;address of microcontroller
  0025                 35     Y_RAMP   EQU   25H   ;in hi nibble and type of
  0023                 36     Y_MOVE   EQU   23H   ;incoming data in lo nibble
  0026                 37     Y_RBOT   EQU   26H   ;of first byte received
  000A                 38     MOVDAT   EQU   0AH   ;code for MOVE data
  0003                 39     RU_DAT   EQU   03H   ;code for RAMP_UP data
  0004                 40     RD_DAT   EQU   04H   ;code for RAMP_DN data
                       41
                       42     ;**-------------------------------------------**
                       43     ;**     DATA INITIALIZATION IN INTERNAL RAM   **
                       44     ;**-------------------------------------------**
                       45
  0000                 46     FLAG     BIT   20H.0 ;bit0 of 20H is used as a
                       47                         ;flag for constant speed step
  0022                 48     PHASE    DATA  22H  ;keeps track of phase patterns
  0023                 49     WAIT     DATA  23H  ;delay for polling RI flag
  0024                 50     TEMP     DATA  24H  ;internal RAM location 24H
```

```
MCS-51 MACRO ASSEMBLER    ROY
                    03/28/88   PAGE    2


LOC  OBJ             LINE     SOURCE

                       51                          ;used for temporary storage
  0025                 52     COUNT    DATA  25H  ;C.S. step's factor of 100
  0026                 53     RPU_LO   DATA  26H  ;low byt  of ramp-up address
  0027                 54     RPU_HI   DATA  27H  ;high byte of ramp-up address
  0028                 55     DIR_LO   DATA  28H  ;direction address ptr - low
  0029                 56     DIR_HI   DATA  29H  ;direction address ptr - high
  002A                 57     STEP     DATA  2AH  ;contain no. of steps
  002B                 58     SPEED    DATA  2BH  ;delay value for speed
  002C                 59     HUNMS    DATA  2CH  ;factor for getting x10 ms
  002D                 60     TENMS    DATA  2DH  ;factor for getting x0.1 ms
  002E                 61     FIRST    DATA  2EH  ;holds first data received
  002F                 62     PAUSE    DATA  2FH  ;wait time for incoming data
  0030                 63     M_COUNT  DATA  30H  ;stores no. of moves requested
  0031                 64     DLY_REG  DATA  31H  ;holds delay value for speed
  0032                 65     MOV_LO   DATA  32H  ;holds low  addr. of MOVE data
  0033                 66     MOV_HI   DATA  33H  ;holds high addr. of MOVE data
  0034                 67     RPD_LO   DATA  34H  ;holds low  addr. of ramp down
  0035                 68     RPD_HI   DATA  35H  ;holds high addr. of ramp down
  0036                 69     DLY_LO   DATA  36H  ;holds factor of 0.01 ms delay
  0037                 70     DLY_HI   DATA  37H  ;holds factor of 0.1 ms delay
  0038                 71     TIME1    DATA  38H  ;to get 0.5 sec in MV_DELAY
  0039                 72     TIME2    DATA  39H  ;to get 10 ms in MV_DELAY
  003A                 73     STAY     DATA  3AH  ;to hold user inserted delay
                       74
                       75     ;*** Using the ORG directive to skip Main program
                       76     ;*** over interrupt vector area
0000                   77             ORG   0H
0000 802C              78             SJMP  MAIN
                       79
                       80     ;*** Interrupt vector of Serial Port interrupt
0023                   81             ORG   23H
0023 0153              82             AJMP  RECEIVE
                       83
                       84     ;**-----------------------------------------------**
                       85     ;**       DATA STORAGE INITIALIZATION AND       **
                       86     ;**       RESERVATION IN PROGRAM CODE AREA      **
                       87     ;**-----------------------------------------------**
                       88
0025 61                89     BUFFCW:  DB   61H      ;extra pattern for proper
                       90                            ;initialization of next MOVE
0026 31                91     CW_DR1:  DB   31H      ;phase patterns for the
0027 91                92     CW_DR2:  DB   91H      ;clockwise direction
0028 C1                93     CW_DR3:  DB   0C1H
0029 61                94     CW_DR4:  DB   61H      ;also used as an extra phase
                       95                            ;pattern (like BUFFCW above)
002A C1                96     CCW_DR1: DB   0C1H     ;phase patterns for the
002B 91                97     CCW_DR2: DB   91H      ;counter-clockwise
002C 31                98     CCW_DR3: DB   31H      ;direction
002D 61                99     CCW_DR4: DB   61H
                      100
                      101     ;::::::::::::::::::::::::::::::::::::::::::::::::
                      102     ;::                                            ::
                      103     ;::    Main program to set up the Stack loc.,  ::
                      104     ;::    timers' mode (timer0 for 0.1 ms delay   ::
                      105     ;::    and timer1 for 1200 baud rate), intrpt. ::
```

```
LOC  OBJ            SOURCE

                    ;::    mask registers, serial communication      ::
                    ;::    port for multiprocessors mode (3), and    ::
                    ;::    initializes the stepper motor's phase     ::
                    ;::    pattern to A=0 and B=1.   Then it will    ::
                    ;::    wait for a serial comm. intrpt to occur.  ::
                    ;::                                              ::
                    ;::::::::::::::::::::::::::::::::::::::::::::::::::
                    MAIN:
002E 75814F                 MOV   SP, #4FH  ;initialize stack ptr to
                                            ;internal RAM location 50H
0031 758922                 MOV   TMOD, #22H  ;timers 0 & 1 in mode 2
0034 758CB4                 MOV   THO, #(-76) ;auto reload value for
                                              ;0.1 msec per count-down
0037 758DEA                 MOV   TH1, #(-22) ;set timer1 for 1200 bps
003A 7598F0                 MOV   SCON, #0F0H ;multiprocessors serial
                                              ;communication (mode 3)
003D D2AF                   SETB  IE.7        ;enable all interrupts
003F D2AC                   SETB  IE.4        ;allow serial port intrpt
0041 D28E                   SETB  TR1         ;start baud rate timer
0043 C2B5                   CLR   P3.5
0045 759060                 MOV   P1, #60H    ;init. stepper motor,
                                              ;with current at 0.8 amp
0048 900026                 MOV   DPTR, #CW_DR1    ;put address of next
004B 858228                 MOV   DIR_LO, DPL      ;phase pattern in
004E 858329                 MOV   DIR_HI, DPH      ;direct RAM DIR_XX
0051 80FE                   SJMP  $                ;at half current

                    ;*************************************************
                    ;**                                             **
                    ;**   Interrupt Service Routine for the multi-  **
                    ;**   processors Serial Communication intrpt.   **
                    ;**   This routine will only take place when    **
                    ;**   the 9th bit of the data it received is a  **
                    ;**   1.  The first byte of data received will  **
                    ;**   be decoded for the EXECUTE command, the   **
                    ;**   RESET command, or the ADDRESS command     **
                    ;**   (returns to Main program with no action   **
                    ;**   taken if the first byte is none of the    **
                    ;**   above commands)                           **
                    ;**                                             **
                    ;*************************************************
                    RECEIVE:
0053 D2D3                   SETB  RSO         ;select register bank 1
0055 C298                   CLR   RI          ;clear Receive Intrpt flag
0057 E599                   MOV   A, SBUF     ;get the first byte in ACC
0059 F52E                   MOV   FIRST, A    ;save first byte in FIRST
005B B4E002                 CJNE  A,#RSET,EXE_RT  ;RESET code?
                    ;       ACALL RESETT      ;routine to initialise
                                              ;ROBOT to known location
005E 0170                   AJMP  RET_MAIN

                    EXE_RT:                   ;EXECUTION routine
0060 B40704                 CJNE  A,#EXEC,ADD_RT   ;EXECUTE code?
0063 11E8                   ACALL EXECU
0065 0170                   AJMP  RET_MAIN
```

```
LOC  OBJ            LINE     SOURCE

                      161
                      162     ADD_RT:                         ;ADDRESS routine
0067 54F0             163             ANL   A, #0F0H            ;mask off lo nibble
0069 B42004           164             CJNE  A,#Y_ADDR,RET_MAIN      ;data for Y?
006C C29D             165             CLR   SM2       ;allow RI flag to be set by
                      166                             ;received data (9th bit=0)
006E 1175             167             ACALL DOWNLOAD
                      168
                      169     RET_MAIN:
0070 D29D             170             SETB  SM2      ;Serial Communication mode 3
0072 C2D3             171             CLR   RS0
0074 32               172             RETI
                      173
                      174     ;... Subroutine to decode the second byte received
                      175     ;... for the type of data that will be received
                      176     ;... (i.e. MOVE, ROBOT, RAMP_UP, or RAMP_DN ) and
                      177     ;... store them in their respective external RAM
                      178     ;... locations
                      179
                      180     DOWNLOAD:
0075 E52E             181             MOV   A, FIRST     ;get first byte received
0077 B42610           182             CJNE  A,#Y_RBOT,DL_MOV    ;ROBOT data?
007A D2B5             183             SETB  P3.5                ;LED indicator on
007C 9000FF           184             MOV   DPTR, #ROBOT    ;get storage location
007F 3098FD           185             JNB   RI, $         ;loop here until data is
0082 C298             186             CLR   RI            ;received; then clear RI
0084 E599             187             MOV   A, SBUF       ;data received into ACC
0086 F0               188             MOVX  @DPTR, A      ;then into ext. RAM loc.
0087 C2B5             189             CLR   P3.5
0089 22               190             RET
                      191
                      192     DL_MOV:                 ;routine for getting MOVE data
008A B42314           193             CJNE  A,#Y_MOVE,DL_RMP    ;MOVE data??
008D D2B5             194             SETB  P3.5                ;turn LED on
008F 3098FD           195             JNB   RI, $     ;wait till 2nd byte is rec'd
0092 C298             196             CLR   RI        ;reset RI flag
0094 E599             197             MOV   A, SBUF
0096 B40A2A           198             CJNE  A,#MOVDAT,DL_RET
0099 900574           199             MOV   DPTR, #MOVE
009C 11C4             200             ACALL RECX
009E C2B5             201             CLR   P3.5      ;turn LED indicator off
00A0 22               202             RET
                      203
                      204     DL_RMP:                 ;routine for getting RAMP data
00A1 B4251F           205             CJNE  A,#Y_RAMP,DL_RET    ;RAMP data?
00A4 D2B5             206             SETB  P3.5                ;turn LED on
00A6 3098FD           207             JNB   RI, $     ;wait for RI flag to be set
00A9 C298             208             CLR   RI
00AB E599             209             MOV   A, SBUF
00AD B40308           210             CJNE  A,#RU_DAT,DL_RMP_D  ;RAMP_UP data?
00B0 900100           211             MOV   DPTR, #RAMP_UP    ;store RAMP_UP data
00B3 11C4             212             ACALL RECX              ;in external RAM
00B5 C2B5             213             CLR   P3.5              ;turn LED off
00B7 22               214             RET
                      215
```

```
LOC  OBJ             LINE     SOURCE

                      216     DL_RMP_D:
00B8 B40408           217             CJNE  A,#RD_DAT,DL_RET  ;RAMP_DN data?
00BB 900001           218             MOV   DPTR, #RAMP_DN    ;store RAMP_DN
00BE 11C4             219             ACALL RECX              ;data in ext. RAM
00C0 C2B5             220             CLR   P3.5              ;turn LED off
00C2 22               221             RET
                      222
                      223     DL_RET:
00C3 22               224             RET
                      225
                      226     ;... Subroutine to get MOVE or RAMP data from the
                      227     ;... SBUF into ACC and then into the corresponding
                      228     ;... external storage RAM location one byte at a
                      229     ;... time.   This routine need not know how many
                      230     ;... bytes is incoming because it will exit (from
                      231     ;... this routine) if no data is received after a
                      232     ;... 50 msec wait.
                      233
                      234     RECX:
00C4 752378           235             MOV   WAIT, #120  ;value to make 50 msec
                      236     RI_CHK:
00C7 209809           237             JB    RI, RECDAT  ;check RI for rec'd data
00CA 752F04           238             MOV   PAUSE, #4   ;value for 0.4 msec
00CD 11DB             239             ACALL DELAY       ;delay
00CF D523F5           240             DJNZ  WAIT, RI_CHK
00D2 22               241             RET
                      242
                      243     RECDAT:                 ;get data into external RAM
00D3 C298             244             CLR   RI
00D5 E599             245             MOV   A, SBUF     ;stores rec'd data in
00D7 F0               246             MOVX  @DPTR, A    ;external RAM location
00D8 A3               247             INC   DPTR        ;pointed by DPTR
00D9 80E9             248             SJMP  RECX        ;get next data, if any
                      249
                      250     ;... Subroutine DELAY for 0.1 msec delay
                      251     DELAY:
00DB D28C             252             SETB  TR0         ;starts timer0
00DD 308DFD           253             JNB   TF0, $
00E0 C28C             254             CLR   TR0
00E2 C28D             255             CLR   TF0         ;clear timer0 flag
00E4 D52FF4           256             DJNZ  PAUSE, DELAY
00E7 22               257             RET
                      258
                      259     ;... Subroutine EXECU to execute robot's move(s)
                      260     ;... specified by the ROBOT and MOVE Xdata
                      261
                      262     EXECU:
00E8 D2B5             263             SETB  P3.5
00EA 9000FF           264             MOV   DPTR, #ROBOT  ;get ROBOT address
00ED E0               265             MOVX  A, @DPTR
00EE F8               266             MOV   R0, A         ;no. of reps in R0
00EF B80002           267             CJNE  R0,#0,REPS    ;at least 1 rep?
00F2 21C6             268             AJMP  EXECU_RET     ;quit if R0=0
                      269
00F4 900574           270     REPS:   MOV   DPTR, #MOVE   ;get MOVE data address
```

```
LOC  OBJ             LINE     SOURCE

00F7 858232           271             MOV   MOV_LO, DPL  ;save MOVE data address
00FA 858333           272             MOV   MOV_HI, DPH  ;in direct RAM MOV_XX
                      273
                      274     M_DATA:
00FD C2B4             275             CLR   P3.4         ;clear EOM indicator
00FF 31CB             276             ACALL MOVE_IN      ;get MOVE data into regs.
0101 B90002           277             CJNE  R1,#0,CW1    ;check R1 for end of move
0104 21C4             278             AJMP  REP_CHK      ;if R1=0 goto to REP_CHK
                      279
                      280     CW1:
0106 1528             281             DEC   DIR_LO       ;go back to the addr. of
                      282                                ;the last phase pattern
0108 852882           283             MOV   DPL, DIR_LO ;get phase pattern (last
010B 852983           284             MOV   DPH, DIR_HI ;used) into the ACC
010E E4               285             CLR   A
010F 93               286             MOVC  A, @A+DPTR
                      287
0110 B90141           288             CJNE  R1,#1,CCW1   ;clockwise direction?
0113 B4C10E           289             CJNE  A,#0C1H,CW2   ;phase pattern C1H?
0116 900029           290             MOV   DPTR, #CW_DR4 ;addr. of 61H pattern
0119 858228           291             MOV   DIR_LO, DPL   ;saves address of 61H
011C 858329           292             MOV   DIR_HI, DPH   ;in direct RAM DIR_XX
011F 752201           293             MOV   PHASE, #1
0122 8074             294             SJMP  CHK_END
                      295     CW2:
0124 B4910E           296             CJNE  A,#91H,CW3    ;phase pattern 91H?
0127 900028           297             MOV   DPTR, #CW_DR3 ;addr. of C1H pattern
012A 858228           298             MOV   DIR_LO, DPL   ;saves address of C1H
012D 858329           299             MOV   DIR_HI, DPH   ;in direct RAM DIR_XX
0130 752202           300             MOV   PHASE, #2
0133 8063             301             SJMP  CHK_END
                      302     CW3:
0135 B4310E           303             CJNE  A,#31H,CW4    ;phase pattern 31H?
0138 900027           304             MOV   DPTR, #CW_DR2 ;addr. of 91H pattern
013B 858228           305             MOV   DIR_LO, DPL   ;saves address of 91H
013E 858329           306             MOV   DIR_HI, DPH   ;in direct RAM DIR_XX
0141 752203           307             MOV   PHASE, #3
0144 8052             308             SJMP  CHK_END
                      309     CW4:
0146 900026           310             MOV   DPTR, #CW_DR1  ;addr. of 31H pattern
0149 858228           311             MOV   DIR_LO, DPL    ;saves address of 31H
014C 858329           312             MOV   DIR_HI, DPH    ;in direct RAM DIR_XX
014F 752204           313             MOV   PHASE, #4
0152 8044             314             SJMP  CHK_END
                      315
0154 B9025E           316     CCW1:   CJNE  R1,#2,NO_MOV     ;counter cw dir.?
0157 B4310E           317             CJNE  A,#31H,CCW2      ;phase pattern 31H?
015A 90002D           318             MOV   DPTR, #CCW_DR4   ;addr. of CCW's 61H
015D 858228           319             MOV   DIR_LO, DPL      ;saves 61H (CCW)
0160 858329           320             MOV   DIR_HI, DPH      ;address in RAM
0163 752201           321             MOV   PHASE, #1
0166 8030             322             SJMP  CHK_END
                      323     CCW2:
0168 B4910E           324             CJNE  A,#91H,CCW3      ;phase pattern 91H?
016B 90002C           325             MOV   DPTR, #CCW_DR3   ;addr. of CCW's 31H
```

```
LOC  OBJ              LINE     SOURCE

016E 858228            326             MOV   DIR_LO, DPL      ;saves 31H (CCW)
0171 858329            327             MOV   DIR_HI, DPH      ;address in RAM
0174 752202            328             MOV   PHASE, #2
0177 801F              329             SJMP  CHK_END
                       330     CCW3:
0179 B4C10E            331             CJNE  A,#0C1H,CCW4     ;phase pattern C1H?
017C 90002B            332             MOV   DPTR, #CCW_DR2   ;addr. of CCW's 91H
017F 858228            333             MOV   DIR_LO, DPL      ;saves 91H (CCW)
0182 858329            334             MOV   DIR_HI, DPH      ;address in RAM
0185 752203            335             MOV   PHASE, #3
0188 800E              336             SJMP  CHK_END
                       337     CCW4:
018A 90002A            338             MOV   DPTR, #CCW_DR1   ;addr. of CCW's C1H
018D 858228            339             MOV   DIR_LO, DPL      ;saves C1H (CCW)
0190 858329            340             MOV   DIR_HI,DPH       ;address in RAM
0193 752204            341             MOV   PHASE, #4
0196 8000              342             SJMP  CHK_END
                       343
                       344     CHK_END:
0198 31EE              345             ACALL UP_RAMP        ;get ramp up address
019A 31F8              346             ACALL DN_RAMP        ;get ramp down addr.
019C 5123              347             ACALL EXE_UP         ;execute the ramping up
019E BC0005            348             CJNE  R4,#0,CS_NOP    ;>100 C.S. steps=0?
01A1 BD0004            349             CJNE  R5,#0,CS_CALL   ;unit C.S. steps=0?
01A4 8008              350             SJMP  DN_CHCK         ;no C.S. stepping
                       351
                       352     CS_NOP:
01A6 00                353             NOP
01A7 00                354             NOP
                       355     CS_CALL:
01A8 5167              356             ACALL EXE_CS     ;execute const. speed
01AA 00                357             NOP            ;4 NOPs added to make the
01AB 00                358             NOP            ;timing from SPIN in EXE_CS
01AC 00                359             NOP            ;to SPIN in EXE_DN 38 instr.
01AD 00                360             NOP            ;cycles long
                       361     DN_CHCK:
01AE BE0002            362             CJNE  R6,#0,DN_CALL    ;R.D. steps=0?
01B1 8002              363             SJMP  NO_MOV
                       364
                       365     DN_CALL:
01B3 51B6              366             ACALL EXE_DN     ;execute the ramping down
                       367     NO_MOV:
01B5 C290              368             CLR   P1.0       ;set current to 0.8 amp
01B7 7117              369             ACALL MV_DELAY   ;delay requested by user
01B9 D2B3              370             SETB  P3.3       ;write a 1 before reading
01BB D2B4              371             SETB  P3.4       ;turn EOM indicator on
01BD 30B3FD            372             JNB   P3.3, $    ;wait for all controllers
01C0 00                373             NOP              ;to finish their moves and
01C1 00                374             NOP              ;read their P3.4 pins
01C2 01FD              375             AJMP  M_DATA     ;get next MOVE data set
                       376
                       377     REP_CHK:
01C4 D803              378             DJNZ  R0, J_REPS  ;reduce repetitions by 1
                       379     EXECU_RET:                ;end of execution routine
01C6 C2B5              380             CLR   P3.5
```

```
LOC  OBJ              LINE     SOURCE

01C8 22                381             RET
                       382
                       383     J_REPS:                 ;do another repetition
01C9 01F4              384             AJMP  REPS
                       385
                       386
                       387     ;... Subroutine MOVE_IN to load end-of-move (0)/
                       388     ;... direction (1 or 2)/no-move (3) into R1, ramp-
                       389     ;... up steps (x100) into R2, ramp-up steps (unit)
                       390     ;... into R3, constant speed steps (x100) into R4,
                       391     ;... constant speed steps (unit) into R5, rampdown
                       392     ;... steps into R6, and the user specified delay
                       393     ;... time (x0.5 sec) into the direct RAM STAY.
                       394
                       395     MOVE_IN:
01CB 853282            396             MOV   DPL, MOV_LO  ;get address of MOVE?
01CE 853383            397             MOV   DPH, MOV_HI  ;data into DPTR
01D1 E0                398             MOVX  A, @DPTR
01D2 F9                399             MOV   R1, A        ;dir./e.o.m./n.m. into R1
01D3 A3                400             INC   DPTR
01D4 E0                401             MOVX  A, @DPTR
01D5 FA                402             MOV   R2, A        ;x100 ramp-up steps in R2
01D6 A3                403             INC   DPTR
01D7 E0                404             MOVX  A, @DPTR
01D8 FB                405             MOV   R3, A        ;unit ram-up steps in R3
01D9 A3                406             INC   DPTR
01DA E0                407             MOVX  A, @DPTR     ;x100 constant speed
01DB FC                408             MOV   R4, A        ;steps in R4
01DC A3                409             INC   DPTR
01DD E0                410             MOVX  A, @DPTR      ;unit constant speed
01DE FD                411             MOV   R5, A         ;steps in R5
01DF A3                412             INC   DPTR
01E0 E0                413             MOVX  A, @DPTR      ;unit ramp-down steps
01E1 FE                414             MOV   R6, A         ;in R6
01E2 A3                415             INC   DPTR
01E3 E0                416             MOVX  A, @DPTR      ;delay time (in sec.) by
01E4 F53A              417             MOV   STAY, A       ;user in direct RAM STAY
01E6 A3                418             INC   DPTR
01E7 858232            419             MOV   MOV_LO, DPL ;save address of next
01EA 858333            420             MOV   MOV_HI, DPH ;MOVE data set in MOV_XX
01ED 22                421             RET
                       422
                       423     ;... Subroutine UP_RAMP to load the ramp up data
                       424     ;... address into direct RAM RPU_XX
                       425
                       426     UP_RAMP:
01EE 900100            427             MOV   DPTR, #RAMP_UP
01F1 858226            428             MOV   RPU_LO, DPL
01F4 858327            429             MOV   RPU_HI, DPH
01F7 22                430             RET
                       431
                       432     ;... Subroutine DN_RAMP to calculate the start
                       433     ;... address of the 2-bytes ramp down data and
                       434     ;... then load that address into direct RAM RPD_XX
                       435
```

```
LOC  OBJ             LINE     SOURCE

                      436     DN_RAMP:
01F8 9000FE           437             MOV   DPTR, #R_D_L  ;get end address of
                      438                                 ;ramp down data in DPTR
01FB EE               439             MOV   A, R6         ;get no. of ramp-dn steps
01FC C3               440             CLR   C             ;clear Carry bit for sub.
01FD 23               441             RL    A             ;multiply by 2
01FE F5F0             442             MOV   B, A          ;put result in B register
0200 E582             443             MOV   A, DPL
0202 C3               444             CLR   C
0203 95F0             445             SUBB  A, B          ;calc. ramp-dn start addr.
0205 04               446             INC   A             ;point at ramp-dn addr.
0206 F534             447             MOV   RPD_LO, A     ;ramp-dn start addr.
0208 858335           448             MOV   RPD_HI, DPH   ;in direct RAM RPD_XX
020B 22               449             RET
                      450
                      451     ;... Subroutine GET_DIR to get address of phase
                      452     ;... pattern for the direction specified in R2
                      453     ;... into "direct" RAM storage DIR_LO and DIR_HI
                      454
                      455     GET_DIR:
020C B9010A           456             CJNE  R1,#1,MOV_CCW ;C.W. direction??
020F 900026           457             MOV   DPTR, #CW_DR1 ;address of pattern
                      458                                 ;for C.W. dir
0212 858228           459             MOV   DIR_LO, DPL   ;save 16-bit address
0215 858329           460             MOV   DIR_HI, DPH   ;in direct memory
0218 22               461             RET
                      462
                      463     MOV_CCW:
0219 90002A           464             MOV   DPTR, #CCW_DR1  ;address of pattern
                      465                                   ;for C.C.W move
021C 858228           466             MOV   DIR_LO, DPL
021F 858329           467             MOV   DIR_HI, DPH
0222 22               468             RET
                      469
                      470     ;... Subroutine EXE_UP to execute the ramping up
                      471     ;... part of a "move" by knowing the number of
                      472     ;... steps to ramp up (with each ramp-up data
                      473     ;... being equivalent to one step).   NOPs are
                      474     ;... widely used to achieve the proper timing
                      475     ;... for stepper-motor delays.
                      476
                      477     EXE_UP:  ;38 instr. cycles in-between SPIN routine
0223 752564           478             MOV   COUNT, #100   ;multiple of 100 steps
                      479     NEX_RU:
0226 752403           480             MOV   TEMP, #3      ;a total of 8 instr.
0229 D524FD           481             DJNZ  TEMP, $       ;cycles added
022C 852682           482             MOV   DPL, RPU_LO   ;get ramp-up data
022F 852783           483             MOV   DPH, RPU_HI   ;address into DPTR
0232 E0               484             MOVX  A, @DPTR      ;get x0.1 msec data
0233 F537             485             MOV   DLY_HI, A     ;x0.1 msec in DLY_HI
0235 A3               486             INC   DPTR
0236 E0               487             MOVX  A, @DPTR      ;get x0.01 msec data
0237 F536             488             MOV   DLY_LO, A     ;x0.01 msec in DLY_LO
0239 A3               489             INC   DPTR          ;pt. at next ramp-up dat
023A BA0004           490             CJNE  R2,#0,MUL_RU  ;>100 ramp-up steps?
```

```
LOC  OBJ              LINE     SOURCE

023D BB0018            491             CJNE  R3,#0,UNT_RU  ;0 ramp-up steps?
0240 22                492             RET                  ;"RET" if R2=0 & R3=0
                       493
                       494     MUL_RU:    ;for >100 ramp-up steps
0241 00                495             NOP              ;NOPs to make timing equal
0242 00                496             NOP              ;to the UNT_RU routine
0243 858226            497             MOV   RPU_LO, DPL   ;addr. of next ramp-up
0246 858327            498             MOV   RPU_HI, DPH   ;data in RPU_XX
0249 51D5              499             ACALL SPIN          ;advance one step
024B D52506            500             DJNZ  COUNT, NOP_RU ;more ramp-up steps?
024E DAD3              501             DJNZ  R2, EXE_UP    ;next 100 steps if R2
                       502                                 ;is not 0
0250 BB00D3            503             CJNE  R3,#0,NEX_RU  ;R3=0?
0253 22                504             RET                 ;RET if R2=0 & R3=0
                       505
                       506     NOP_RU:              ;total of 4 instr. cycles
0254 00                507             NOP          ;NOPs to compensate for timing
0255 00                508             NOP          ;difference of the other path
0256 80CE              509             SJMP  NEX_RU
                       510
                       511     UNT_RU:                         ;for <100 ramp-up steps
0258 858226            512             MOV   RPU_LO, DPL   ;addr. of next ramp-up
025B 858327            513             MOV   RPU_HI, DPH   ;data in RPU_XX
025E 51D5              514             ACALL SPIN          ;advance one step
0260 DBF2              515             DJNZ  R3, NOP_RU    ;R3=0 after decrement?
0262 00                516             NOP              ;adds 4 NOPs to make total
0263 00                517             NOP              ;of 8 instr. cycles after
0264 00                518             NOP              ;the SPIN routine to get
0265 00                519             NOP              ;proper timing
0266 22                520             RET              ;RET since R3=0
                       521
                       522     ;... Subroutine EXE_CS to execute constant speed
                       523     ;... stepping
                       524
                       525     EXE_CS:  ;38 instr. cycles in-betwn SPIN routine
0267 752564            526             MOV   COUNT, #100
                       527     NEX_CS:
026A 752401            528             MOV   TEMP, #1      ;4 instr. cycles added
026D D524FD            529             DJNZ  TEMP, $       ;for proper timing
0270 852682            530             MOV   DPL, RPU_LO   ;the last ramp-up data
0273 852783            531             MOV   DPH, RPU_HI   ;is now the c.s. data
0276 E0                532             MOVX  A, @DPTR      ;get x0.1 msec delay
0277 F537              533             MOV   DLY_HI, A     ;data in DLY_HI
0279 A3                534             INC   DPTR
027A E0                535             MOVX  A, @DPTR      ;get x0.01 msec delay
027B F536              536             MOV   DLY_LO, A     ;data in DLY_LO
027D BC0004            537             CJNE  R4,#0,MUL_CS  ;>100 c.s. steps?
0280 BD0026            538             CJNE  R5,#0,UNT_CS  ;0 const. speed step?
0283 22                539             RET                 ;"RET" if R4=0 & R5=0
                       540
                       541     MUL_CS:
0284 00                542             NOP                  ;add 2 NOPs for the
0285 00                543             NOP                  ;timing delays
0286 51D5              544             ACALL SPIN           ;advance by 1 step
0288 00                545             NOP
```

```
LOC  OBJ             LINE     SOURCE

0289 00               546             NOP
028A D52506           547             DJNZ  COUNT, NO1_CS  ;more c.s. steps?
028D DC12             548             DJNZ  R4, NO3_CS     ;next 100 steps if R4
                      549                                  ;is not 0
028F BD0007           550             CJNE  R5,#0,NO2_CS   ;R5=0?
0292 22               551             RET
                      552
                      553     NO1_CS:
0293 752401           554             MOV   TEMP, #1       ;total of 4 instr.
0296 D524FD           555             DJNZ  TEMP, $        ;cycles added
                      556     NO2_CS:
0299 752402           557             MOV   TEMP, #2       ;gives 6 instr. cycles
029C D524FD           558             DJNZ  TEMP, $        ;for timing delay
029F 80C9             559             SJMP  NEX_CS
                      560
                      561     NO3_CS:
02A1 752402           562             MOV   TEMP, #2       ;6 instr. cycles for
02A4 D524FD           563             DJNZ  TEMP, $        ;timing delay
02A7 80BE             564             SJMP  EXE_CS
                      565
                      566     UNT_CS:
02A9 51D5             567             ACALL SPIN           ;advance by 1 step
02AB 00               568             NOP
02AC 00               569             NOP
02AD DDE4             570             DJNZ  R5, NO1_CS     ;R5=0 after decrement?
02AF 752401           571             MOV   TEMP, #1       ;a total of 4 instr.
02B2 D524FD           572             DJNZ  TEMP, $        ;cycles added
02B5 22               573             RET                  ;"RET" since R5=0
                      574
                      575
                      576     ;... Subroutine EXE_DN to execute ramping down
                      577
                      578     EXE_DN:   ;38 instr. cycles in-betwn SPIN routine
02B6 853482           579             MOV   DPL, RPD_LO    ;address of ramp-down
02B9 853583           580             MOV   DPH, RPD_HI    ;in DPTR
02BC E0               581             MOVX  A, @DPTR       ;get x0.1 msec ramp-
02BD F537             582             MOV   DLY_HI, A      ;down data in DLY_HI
02BF A3               583             INC   DPTR
02C0 E0               584             MOVX  A, @DPTR       ;get x0.01 msec ramp-
02C1 F536             585             MOV   DLY_LO, A      ;down data in DLY_LO
02C3 A3               586             INC   DPTR
02C4 858234           587             MOV   RPD_LO, DPL    ;save address of next
02C7 858335           588             MOV   RPD_HI, DPH    ;ramp-dn data in RPD_XX
02CA 51D5             589             ACALL SPIN           ;advance by 1 step
02CC 752407           590             MOV   TEMP, #7       ;gives 12 instr. cycles
02CF D524FD           591             DJNZ  TEMP, $        ;for timing delay
02D2 DEE2             592             DJNZ  R6, EXE_DN     ;more ramp-down steps?
02D4 22               593             RET
                      594
                      595     ;... Subroutine SPIN to output phase pattern to
                      596     ;... port P1, make a delay, and then increment
                      597     ;... the pointer DPTR to the next phase pattern.
                      598     ;... 2 NOPs have been added to make total timing
                      599     ;... "overhead" equal 79 instr. cycles or 94.8
                      600     ;... usec (need to subtract 90 usec from actual
```

```
LOC  OBJ             LINE     SOURCE

                      601     ;... delay time prior to downloading the RAM)
                      602     ;... Overhead = 38 (SPIN to SPIN) + 41 (within
                      603     ;   . SPIN and including 5 from the SM_DELAY
                      604     ;... subroutine) = 79 instruction cycles
                      605
                      606     SPIN:           ;a total of 41 instruction cycles
02D5 00               607             NOP
02D6 00               608             NOP
02D7 852882           609             MOV   DPL, DIR_LO   ;DPTR points at
02DA 852983           610             MOV   DPH, DIR_HI   ;phase pattern
02DD E4               611             CLR   A
02DE 93               612             MOVC  A,@A+DPTR     ;get phase pattern
02DF A3               613             INC   DPTR          ;points at next pattern
02E0 F590             614             MOV   P1, A         ;output phase pattern
02E2 51FC             615             ACALL SM_DELAY
02E4 858228           616             MOV   DIR_LO, DPL   ;save address of next
02E7 858329           617             MOV   DIR_HI, DPH   ;pattern in direct RAM
02EA D52206           618             DJNZ  PHASE, RET_SPIN
02ED 510C             619             ACALL GET_DIR       ;reload phase pattern
02EF 752204           620             MOV   PHASE, #4     ;reload phase counter
02F2 22               621             RET
                      622
                      623     RET_SPIN:               ;this routine is set up for
02F3 00               624             NOP             ;16 instr. cycles so as to
02F4 00               625             NOP             ;make the timing (delay)
02F5 752405           626             MOV   TEMP, #5  ;similar to the other path
02F8 D524FD           627             DJNZ  TEMP, $   ;that requires a phase
02FB 22               628             RET             ;pattern reload
                      629
                      630     ;... Subroutine SM_DELAY for the stepper motor
                      631     ;... delays. (With a 5 cycles overhead)
                      632
                      633     SM_DELAY:           ;0.0996 msec achieved instead
02FC D28C             634             SETB  TR0   ;of 0.1 msec
02FE 308DFD           635             JNB   TF0, $    ;gives 0.0912 msec here
0301 C28C             636             CLR   TR0
0303 C28D             637             CLR   TF0           ;clear timer0 flag
0305 D537F4           638             DJNZ  DLY_HI, SM_DELAY
0308 E4               639             CLR   A                 ;overhead of 1
0309 B53601           640             CJNE  A,DLY_LO, DLAY    ;overhead of 2
030C 22               641             RET                     ;overhead of 2
                      642
                      643     DLAY:           ;to give 0.01 msec delay
030D 00               644             NOP     ;with an error of -0.0004 msec
030E 00               645             NOP     ;i.e. the actual time achieved
030F 00               646             NOP     ;is only 0.0096.  A carefully
0310 00               647             NOP     ;selected crystal frequency
0311 00               648             NOP     ;will eliminate this error
0312 00               649             NOP
0313 D536F7           650             DJNZ  DLY_LO, DLAY
0316 22               651             RET
                      652
                      653     ;... Subroutine MV_DELAY for "end-of-move" delays
                      654     ;... requested by user, in factor of 0.5 second up
                      655     ;... to a maximum of 127.5 sec or 2.125 min. No
```

```
LOC  OBJ             LINE     SOURCE

                      656     ;... delay for Y microcontroller if STAY=0.
                      657     MV_DELAY:
0317 E53A             658             MOV   A, STAY
0319 B40002           659             CJNE  A,#0,SECON     ;delay time=0?
031C 8018             660             SJMP  MV_D_RET
                      661
                      662     SECON:
031E 753832           663             MOV   TIME1, #50     ;to get 0.5 second
                      664     ONE_S:
0321 753964           665             MOV   TIME2, #100    ;to get 10 msec
                      666     TEN_MS:
0324 D28C             667             SETB  TR0
0326 308DFD           668             JNB   TF0, $     ;loop till TF0=1
0329 C28C             669             CLR   TR0
032B C28D             670             CLR   TF0
032D D539F4           671             DJNZ  TIME2, TEN_MS
0330 D538EE           672             DJNZ  TIME1, ONE_S
0333 D53AE8           673             DJNZ  STAY, SECON
                      674     MV_D_RET:
0336 22               675             RET
                      676
                      677
                      678     ;**-----------------------------------------------**
                      679     ;**    EXTERNAL DATA STORAGE INITIALIZATION      **
                      680     ;**    AND RESERVATION IN THE EXTERNAL RAM       **
                      681     ;**-----------------------------------------------**
                      682
                      683     ;External RAM size used is only 2K byte (2016)
                      684
----                  685             XSEG  AT  1
                      686
0001                  687     RAMP_DN:  DS    253     ;a total of 127 ramp down
00FE                  688     R_D_L:    DS    1       ;steps (with 2 bytes each)
00FF                  689     ROBOT:    DS    1       ;MOVE reptetition value
0100                  690     RAMP_UP:  DS    1140    ;max. of 570 ramp-up steps
0574                  691     MOVE:     DS    630     ;maximum of 90 MOVEs
                      692
                      693             END             ; ^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^
```

APPENDIX J

Listing of the BASIC Program MOVE.BAS

```
10   '
20   '
30   '          MOVE.bas (MOVE data conversion program)
40   '
50   '
60   '
70   '  Definition of labels:-
80   '  R     - ramp-up step delay time (1 for x0.1 msec, 2 for x0.01 msec)
90   '  RD    - ramp-down step delay time (   - " -     ,      - " -      )
100  '  XSTEP - total number of steps for X
110  '  YSTEP - total number of steps for Y
120  '  MOVX  - the seven parameters for X
130  '  MOVY  - the seven parameters for Y
140  '  MOV(K,1) - direction (1 or 2) / no move (3) / end of move (0)
150  '  MOV(K,2) - ramp-up steps (x100)
160  '  MOV(K,3) - ramp-up steps (x1)
170  '  MOV(K,4) - constant speed steps (x100)
180  '  MOV(K,5) - constant speed steps (x1)
190  '  MOV(K,6) - ramp-down steps (x1)
200  '  MOV(K,7) - delay time added by user (x0.5 sec)
210  '
220  '  INPUT FILES NEEDED:-
230  '          1.   Rampup   -- ramp-up data (2 bytes per data format)
240  '          2.   Rampdown -- ramp-down data (2 bytes per data format)
250  '
260  OPTION BASE 1
270  COLOR 15,0 : CLS
280  DEFINT A-Z
290  DIM R(600,2), RD(200,2), XSTEP%(100), YSTEP%(100)
300  DIM MOVX(100,7), MOVY(100,7)
310  LOCATE 2,10 : COLOR 14,4
320  PRINT "   PROGRAM TO CREATE MOVE DATA FOR X AND Y MICROCONTROLLERS   "
330  COLOR 15,0
340  RDAT$="RAMPUP": RDDAT$="RAMPDOWN"       'ramp-up and ramp-down files
350  OPEN RDAT$ FOR INPUT AS #1
360  J=1 : K=1 : STP!=.0157  'each step is equivalent to 0.0157 inch
370  WHILE NOT EOF(1)
380  INPUT #1, R(J,1) : INPUT #1, R(J,2)
390  J=J+1
400  WEND
410  CLOSE
420  R(J,1)=R(J-1,1)  :  R(J,2)=R(J-1,2)-1
430  J=J+1
440  R(J,1)=0 : R(J,2)=0           ': PRINT J
450  M=1
460  OPEN RDDAT$ FOR INPUT AS #2
470  WHILE NOT EOF(2)
480  INPUT #2, RD(M,1) : INPUT #2, RD(M,2)
490  M=M+1
500  WEND
510  CLOSE
520  RD(M,1)=0            'add 0 for "end of data" indication
530  LOCATE 6,6
540  PRINT "Initial X coord. (inches)= 0 "
550  LOCATE 6,33 : INPUT "",X1!
560  X1%=X1!/STP!        'find equivalent number of steps for given location
570  X1!=X1%*STP!        'computes the exact location based on the steps
580  LOCATE 6,6 : PRINT "Actual initial X  coord. =";X1!
590  LOCATE 6,44
600  PRINT "Initial Y coord. (inches)= 0"
610  LOCATE 6,71 : INPUT "",Y1!
620  Y1%=Y1!/STP!        'find equivalent number of steps for given location
630  Y1!=Y1%*STP!        'computes the exact location based on the steps
```

```
640  LOCATE 6,44 : PRINT "Actual initial Y  coord. =";Y1!
650  LOCATE 7,6
660  INPUT "Final X coord.   (inches)= ", X2!
670  X2%=X2!/STP!       'find equivalent number of steps for given location
680  X2!=X2%*STP!       'computes the exact location based on the steps
690  LOCATE 7,6 : PRINT "Actual  final  X  coord. =";X2!
700  LOCATE 7,44
710  INPUT "Final Y coord.   (inches)= ", Y2!
720  Y2%=Y2!/STP!       'find equivalent number of steps for given location
730  Y2!=Y2%*STP!       'computes the exact location based on the steps
740  LOCATE 7,44 : PRINT "Actual  final  Y  coord. =";Y2!
750  XDAT!=X2!-X1!
760  YDAT!=Y2!-Y1!
770  LOCATE 8,6  : INPUT "X's delay time (x0.5 sec)= ", MOVX(K,7)
780  LOCATE 8,44 : INPUT "Y's delay time (x0.5 sec)= ", MOVY(K,7)
790  LOCATE 9,6  : PRINT "Actual  distance  for  X =";XDAT!
800  LOCATE 9,44 : PRINT "Actual  distance  for  Y =";YDAT!
810  MOVX(K,1)=3 : MOVY(K,1)=3
820  IF XDAT!>0! THEN MOVX(K,1)=1
830  IF XDAT!<0! THEN MOVX(K,1)=2 : XDAT!=-XDAT!
840  IF YDAT!>0! THEN MOVY(K,1)=1
850  IF YDAT!<0! THEN MOVY(K,1)=2 : YDAT!=-YDAT!
860  XSTEP!=XDAT!/STP! : YSTEP!=YDAT!/STP!
870  XSTEP%(K)=XSTEP!  : YSTEP%(K)=YSTEP!
880  IF XSTEP%(K)=0 THEN MOVX(K,1)=3
890  IF YSTEP%(K)=0 THEN MOVY(K,1)=3
900  LOCATE 10,6  : PRINT "Direction of MOVE  for X =";MOVX(K,1)
910  LOCATE 10,44 : PRINT "Direction of MOVE  for Y =";MOVY(K,1)
920  LOCATE 11,6  : PRINT "Total number of X steps  =";XSTEP%(K)
930  LOCATE 11,44 : PRINT "Total number of Y steps  =";YSTEP%(K)
940  IF XSTEP%(K)=0 THEN GOSUB 2350 : GOTO 1340
950  LOCATE 12,6  : INPUT "X's Constant Speed (in/s)= ", XCS!
960  LOCATE 15,2  : PRINT "                                                "
970  CSX!=STP!/XCS!*1000  'get delay time of c.s.
980  CX%=CSX!*100         'change delay time to x10 microsec
990  CX%=CX%-9            'subtract overhead of 90 microsec
1000  CXHI%=CX%/10 : CXLO%=CX% MOD 10
1010  IF CXLO%>4 THEN CXHI%=CXHI%-1
1020     REM   ***  This part of program calculates the required number  ***
1030     REM   ***  of ramp-up steps for X in the form of x100 and x1.   ***
1040  FOR H=1 TO J    'to calculate ramp-up steps
1050  IF R(H,1)=0 OR R(H+1,1)=0 THEN GOSUB 2030 : GOTO 1890
1060  IF CXHI%<R(H,1) THEN GOTO 1100
1070  A=0
1080  GOSUB 2100          'check if R(H) or R(H+1) is closer to CX
1090  IF FLAG=0 THEN 1110
1100  NEXT H
1110  RUX=H-1                    'ramp up steps is H-1
1120  LOCATE 14,6  : PRINT "User's time:  Hi=";CXHI%
1130  LOCATE 14,29 : PRINT "Lo=";CXLO%
1140  LOCATE 15,6  : PRINT "Chosen time:  Hi=";R(H,1)
1150  LOCATE 15,29 : PRINT "Lo=";R(H,2)
1160  MOVX(K,2)=RUX/100 : MOVX(K,3)=RUX MOD 100
1170  IF MOVX(K,3)>49 THEN MOVX(K,2)=MOVX(K,2)-1
1180     REM   ***  Required number of ramp-down steps calculated here  ***
1190  FOR H=1 TO M                'to calculate ramp-down steps
1200 TEMP3=RD(H,1)*10 + RD(H,2)
1210 TEMP4=RD(H+1,1)*10 + RD(H+1,2)
1220 LO=ABS(TEMP3 - CTEMP)
1230 HI=ABS(TEMP4 - CTEMP)
1240 IF LO>HI OR LO=HI THEN NEXT H
1250 MOVX(K,6)=M-H        'no. of ramp down steps
1260 CSX=XSTEP%(K) - RUX - MOVX(K,6)  'total constant speed steps for X
```

```
1270 IF CSX<0 THEN GOSUB 2210 : GOTO 950
1280 MOVX(K,4)=CSX/100 : MOVX(K,5)=CSX MOD 100
1290 IF MOVX(K,5)>49 THEN MOVX(K,4)=MOVX(K,4)-1
1300 LOCATE 16,6 : PRINT "R.U. steps: x100=";MOVX(K,2);"   x1=";MOVX(K,3)
1310 LOCATE 17,6 : PRINT "C.S. steps: x100=";MOVX(K,4)
1320 LOCATE 17,29 : PRINT "x1=";MOVX(K,5)
1330 LOCATE 18,6 : PRINT "R.D. steps:   x1=";MOVX(K,6)
1340 IF YSTEP%(K)=0 THEN GOSUB 2420 : GOTO 1750
1350 LOCATE 12,44 : INPUT "Y's constant Speed (in/s)= ", YCS!
1360 LOCATE 15,40 : PRINT "                                    "
1370 CSY!=STP!/YCS!*1000  'get delay time of constant step for Y
1380 CY%=CSY!*100         'change delay time to x10 microsec
1390 CY%=CY%-9            'subtract overhead of 90 microseconds
1400 CYHI%=CY%/10 : CYLO%=CY% MOD 10
1410 IF CYLO%>4 THEN CYHI%=CYHI%-1
1420 CXHI%=CYHI% : CXLO%=CYLO%
1430    REM   ***  This part of program calculates the required number  ***
1440    REM   ***  of ramp-up steps for Y in the form of x100 and x1.   ***
1450 FOR H=1 TO J    'to calculate ramp-up steps
1460 IF R(H,1)=0 OR R(H+1,1)=0 THEN GOSUB 2030 : GOTO 1890
1470 IF CYHI%<R(H,1) THEN GOTO 1510
1480 A=38
1490 GOSUB 2100      'check if R(H) or R(H+1) is a closer step delay time
1500 IF FLAG=0 THEN 1520
1510 NEXT H
1520 RUY=H-1                  'ramp up steps is H-1
1530 LOCATE 14,44 : PRINT "User's time:  Hi=";CYHI%
1540 LOCATE 14,67 : PRINT "Lo=";CYLO%
1550 LOCATE 15,44 : PRINT "Chosen time:  Hi=";R(H,1)
1560 LOCATE 15,67 : PRINT "Lo=";R(H,2)
1570 MOVY(K,2)=RUY/100 : MOVY(K,3)=RUY MOD 100
1580 IF MOVY(K,3)>49 THEN MOVY(K,2)=MOVY(K,2)-1
1590    REM   ***  Required number of ramp-down steps calculated here  ***
1600 FOR H=1 TO M                'to calculate ramp-down steps
1610 TEMP3=RD(H,1)*10 + RD(H,2)
1620 TEMP4=RD(H+1,1)*10 + RD(H+1,2)
1630 LO=ABS(TEMP3 - CTEMP)
1640 HI=ABS(TEMP4 - CTEMP)
1650 IF LO>HI OR LO=HI THEN NEXT H
1660 MOVY(K,6)=M-H       'no. of ramp down steps
1670 CSY=YSTEP%(K) - RUY - MOVY(K,6)  'total constant speed steps for Y
1680 IF CSY<0 THEN GOSUB 2470 : GOTO 1350
1690 MOVY(K,4)=CSY/100 : MOVY(K,5)=CSY MOD 100
1700 IF MOVY(K,5)>49 THEN MOVY(K,4)=MOVY(K,4)-1
1710 LOCATE 16,44 : PRINT "R.U. steps: x100=";MOVY(K,2);"   x1=";MOVY(K,3)
1720 LOCATE 17,44 : PRINT "C.S. steps: x100=";MOVY(K,4)
1730 LOCATE 17,67 : PRINT "x1=";MOVY(K,5)
1740 LOCATE 18,44 : PRINT "R.D. steps:   x1=";MOVY(K,6)
1750 LOCATE 21,6 : COLOR 11,7 : PRINT " ENTER ";
1760 COLOR 11,0 : PRINT " for next MOVE data"
1770 LOCATE 21,44 : COLOR 11,7 : PRINT "  ESC  ";
1780 COLOR 11,0 : PRINT " to display MOVE data"
1790 LOCATE 23,22 : COLOR 11,7 : PRINT " SPACE ";
1800 COLOR 11,0 : PRINT " to save MOVE data in a disk file"
1810 COLOR 15,0
1820 AA$=INKEY$
1830 IF AA$="" THEN GOTO 1820
1840 IF LEN(AA$)<>1 THEN 1820
1850 IF ASC(AA$) = 13 THEN GOSUB 1920 : GOTO 650
1860 IF ASC(AA$) = 27 THEN 2590
1870 IF ASC(AA$) = 32 THEN GOSUB 2970 : GOTO 1890
1880 BEEP : GOTO 1820
1890 COLOR 7,0 : END
```

```
1900     REM  ***  Subroutine (SCREEN) to get the screen display  ***
1910     REM  ***  ready for entering the next set of MOVE data   ***
1920 COLOR 15,0
1930 X1!=X2! : Y1!=Y2!
1940 K=K+1
1950 CLS
1960 LOCATE 2,10 : COLOR 14,4
1970 PRINT "   PROGRAM TO CREATE MOVE DATA FOR X AND Y MICROCONTROLLERS    "
1980 COLOR 15,0
1990 LOCATE 6,6  : PRINT "Initial X coord. (inches)=";X1!
2000 LOCATE 6,44 : PRINT "Initial Y coord. (inches)=";Y1!
2010 RETURN
2020     REM  ***  Inform user speed is out of range and quits  ***
2030 CLS : LOCATE 11,24 : PRINT "Speed selected is OUT OF RANGE "
2040 LOCATE 13,20
2050 COLOR 14,9 : PRINT "  MAXIMUM SPEED ALLOWED IS 35 IN/SEC  "
2060 COLOR 7,0
2070 RETURN
2080     REM   ***  Routine to check if the present delay value R(H) or the
2090     REM   ***  next R(H+1) is to be used as the last ramp-up data
2100 FLAG=0
2110 TEMP1=R(H,1)*10 + R(H,2)
2120 TEMP2=R(H+1,1)*10 + R(H+1,2)
2130 CTEMP=CXHI%*10 + CXLO%
2140 UND= ABS(TEMP1 - CTEMP)
2150 OVR= ABS(TEMP2 - CTEMP)
2160 IF UND < OVR THEN RETURN
2170 FLAG=1 : COLOR 10,0
2180 LOCATE 15,6+A : PRINT "U=";UND
2190 LOCATE 15,16+A : PRINT "O=";OVR
2200 COLOR 15,0 : RETURN
2210     REM  ***  Routine to inform user of speed invalidity for X  ***
2220 LOCATE 12,32 : PRINT "          "
2230 FOR AG=1 TO 7
2240 LOCATE 14,AG*5
2250 PRINT "     "
2260 LOCATE 15,AG*5
2270 PRINT "     "
2280 NEXT AG
2290 LOCATE 15,2 : COLOR 14,5
2300 PRINT " X's speed in MOVE(";K;")";"is not valid! "
2310 COLOR 15,0
2320 RETURN
2330     REM   ***  Subroutine to put zeros in MOVX(K,2) to MOVX(K,6)  ***
2340     REM   ***  and then display "NO MOVE FOR X MICROCONTROLLER"   ***
2350 FOR Q=2 TO 6
2360 MOVX(K,Q)=0
2370 NEXT Q
2380 LOCATE 16,6 : PRINT "NO MOVE FOR X MICROCONTROLLER"
2390 RETURN
2400     REM   ***  Subroutine to put zeros in MOVY(K,2) to MOVY(K,6)  ***
2410     REM   ***  and then display "NO MOVE FOR Y MICROCONTROLLER"   ***
2420 FOR P=2 TO 6
2430 MOVY(K,P)=0
2440 NEXT P
2450 LOCATE 16,44 : PRINT "NO MOVE FOR Y MICROCONTROLLER"
2460 RETURN
2470     REM  ***  Routine to inform user of speed invalidity for Y  ***
2480 LOCATE 12,70 : PRINT "          "
2490 FOR AG=8 TO 15
2500 LOCATE 14,AG*5
2510 PRINT "     "
2520 LOCATE 15,AG*5
```

```
2530 PRINT "      "
2540 NEXT AG
2550 LOCATE 15,41 : COLOR 14,5
2560 PRINT " Y's speed in MOVE(";K;")";"is not valid! "
2570 COLOR 15,0
2580 RETURN
2590    REM  ***  Routine to display MOVE data on screen  ***
2600 CLS
2610 LOCATE 1,11 : COLOR 14,1 : PRINT "              X's MOVEs           "
2620 LOCATE 1,47 : COLOR 14,1 : PRINT "              Y's MOVEs           "
2630 FOR N=1 TO K
2640 LOCATE ,2
2650 COLOR 14,0 : PRINT "MOVE";N;
2660 COLOR 15,0
2670 LOCATE ,11 : PRINT MOVX(N,1);
2680 LOCATE ,15 : PRINT MOVX(N,2);
2690 LOCATE ,19 : PRINT MOVX(N,3);
2700 LOCATE ,24 : PRINT MOVX(N,4);
2710 LOCATE ,29 : PRINT MOVX(N,5);
2720 LOCATE ,34 : PRINT MOVX(N,6);
2730 LOCATE ,39 : PRINT MOVX(N,7);
2740 LOCATE ,47 : PRINT MOVY(N,1);
2750 LOCATE ,51 : PRINT MOVY(N,2);
2760 LOCATE ,55 : PRINT MOVY(N,3);
2770 LOCATE ,60 : PRINT MOVY(N,4);
2780 LOCATE ,65 : PRINT MOVY(N,5);
2790 LOCATE ,70 : PRINT MOVY(N,6);
2800 LOCATE ,75 : PRINT MOVY(N,7)
2810 NEXT N
2820 LOCATE ,22 : PRINT ""
2830 LOCATE ,22 : COLOR 11,7 : PRINT " ENTER ";
2840 COLOR 11,0 : PRINT " for next MOVE data"
2850 LOCATE ,22 : PRINT ""
2860 LOCATE ,22 : COLOR 11,7 : PRINT " SPACE ";
2870 COLOR 11,0 : PRINT " to save MOVE data in a disk file"
2880 AA$=INKEY$
2890 IF AA$="" THEN GOTO 2880
2900 IF LEN(AA$)<>1 THEN 2880
2910 IF ASC(AA$) = 13 THEN GOSUB 1920 : GOTO 650
2920 IF ASC(AA$) = 32 THEN GOSUB 2970 : GOTO 1890
2930 BEEP : GOTO 2880
2940 RETURN
2950    REM  ***  Subroutine (SAVE) to save both X and Y MOVE data set  ***
2960    REM  ***  in disk files with names to be given by the user       ***
2970 CLS : LOCATE 2,10 : COLOR 14,4
2980 PRINT "   PROGRAM TO CREATE MOVE DATA FOR X AND Y MICROCONTROLLERS   "
2990 COLOR 15,0
3000 LOCATE 10,20 : INPUT "File name to store X's MOVE data:- ",XFILE$
3010 LOCATE 12,20 : INPUT "File name to store Y's MOVE data:- ",YFILE$
3020 OPEN XFILE$ FOR OUTPUT AS #1
3030 OPEN YFILE$ FOR OUTPUT AS #2
3040 FOR S=1 TO K           'to save all the MOVEs
3050 FOR T=1 TO 6           'the first six MOVE data
3060 PRINT #1, MOVX(S,T);
3070 PRINT #2, MOVY(S,T);
3080 NEXT T
3090 PRINT #1, MOVX(S,7) : PRINT #2, MOVY(S,7)
3100 NEXT S
3110 PRINT #1, 0 : PRINT #2, 0
3120 CLOSE
3130 RETURN
```

APPENDIX K

Listing of the BASIC Program RC.BAS

```
10   '
20   '      RC.BAS --- To convert ramp delay time into 2-byte data
30   '    (with program overhead of 8031 taken into consideration)
40   '    for the 8031 microcontrollers of the ROBOT system.  The
50   '    data will then be stored in disk file for future use.
60   '
70   '
80   OPTION BASE 1
90   DEFINT A-Z
100  DIM R(600,2), RSTP!(600)
110  I=1
120  CLS : LOCATE 3,8 : COLOR 14,3
130  PRINT " PROGRAM TO CONVERT RAMP DELAY TIME INTO 2-BYTE DATA FOR THE 8031s
140  IF I>1 THEN GOSUB 520
150  COLOR 15,0 : LOCATE 11,10
160  INPUT "Ramp step delay time in msec = ", RSTP!(I)
170  RSTP%=RSTP!(I)*100
180  RU%=RSTP%-9                    'subtract 90 microseconds due to "overhead"
190  RHI%=RU%/10 : RLO%=RU% MOD 10
200  IF RLO%>4 THEN RHI%=RHI%-1
210  R(I,1)=RHI% : R(I,2)=RLO%
220  LOCATE 13,21 : PRINT "x0.1 msec value   = ";R(I,1);"   "
230  LOCATE 14,21 : PRINT "x0.01 msec value  = ";R(I,2)
240  LOCATE 16,21 : PRINT "No. of entries    = ";I : I=I+1
250  LOCATE 18,20 : COLOR 14,7 : PRINT " ENTER ";
260  COLOR 14,0 : PRINT " to convert next RAMP data"
270  LOCATE 20,20 : COLOR 14,7 : PRINT "  ESC  ";
280  COLOR 14,0 : PRINT " to save and display ramp data"
290  A$=INKEY$
300  IF A$="" THEN 290
310  IF LEN(A$)<>1 THEN 290
320  IF ASC(A$) = 13 THEN 120
330  IF ASC(A$) = 27 THEN GOSUB 360 : GOTO 350
340  BEEP : GOTO 290
350  END
360  COLOR 15,0 : CLS : LOCATE 3,20
370  INPUT "Enter file name to save RAMP data: ", RR$
380  OPEN RR$ FOR OUTPUT AS #2
390  FOR L=1 TO I-1
400  PRINT #2, R(L,1); R(L,2);
410  NEXT L
420  CLOSE
430  LOCATE 5,10
440  PRINT "Delay Time", "x0.1 value", "   x0.01 value"
450  LOCATE ,10 : PRINT "  (msec)  ", "  (msec)    ","      (msec)"
460  FOR L=1 TO I-1
470  LOCATE ,12 : PRINT RSTP!(L);
480  LOCATE ,31 : PRINT R(L,1);
490  LOCATE ,50 : PRINT R(L,2)
500  NEXT
510  RETURN
520  LOCATE 9,10 : COLOR 14,1
530  PRINT " Entry number";I-1;"=";RSTP!(I-1);"msec "
540  RETURN
```

VITA

Siak Thong Yeo was born in Singapore on May 17, 1958. He served his national service for two and a half years in the military (Singapore Armed Forces). After his military service, he came to the United States in September of 1980 and attended the University of Tennessee, Knoxville. In Winter of 1984, he graduated with highest honors with a Bachelor of Science degree in Electrical Engineering. He also graduated as the top student from the College of Engineering that quarter.

He started graduate school in the Spring of 1984. A graduate assistantship was awarded to him by the Electrical Engineering department in the Fall of 1984. He also worked on two research projects, of which the last one became the subject of his thesis. He has been initiated into the Tau Beta Phi Honor Society and Eta Kappa Nu Honor Society.