Spring 2002

Embedded Software Programming to Develop a Command Line User Interface for Monitoring and Debugging a Manually Driven Gas Regulator Control System

Syed N. Hyder
Old Dominion University

EMBEDDED SOFTWARE PROGRAMMING TO DEVELOP A COMMAND LINE USER INTERFACE FOR MONITORING AND DEBUGGING A MANUALLY DRIVEN GAS REGULATOR CONTROL SYSTEM

by

Syed N. Hyder
B.E. November 1997,
N.E.D. University, Pakistan

A Thesis submitted to the Faculty of
Old Dominion University in Partial Fulfillment of
the Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
May 2002

Approved by:

Glenn A. Gerdin (Director)

Vishnu K. Lakdawala (Member)

James F. Leathrum, Jr. (Member)

ABSTRACT

EMBEDDED SOFTWARE PROGRAMMING TO DEVELOP A COMMAND LINE USER INTERFACE FOR MONITORING AND DEBUGGING A MANUALLY DRIVEN GAS REGULATOR CONTROL SYSTEM

Syed N. Hyder
Old Dominion University, 2002
Director: Dr. Glenn A. Gerdin

This thesis presents a complete embedded programming model and software codes for a command-line user interface for CONCOA's gas regulator control system. Control Corporation of America (CONCOA) manufactures high-pressure gas regulators, which mechanically control the pressure at their outlets. Since the control system is based on mechanical regulators, adding or stopping gas flow from the system can cause manifold fluctuation that could further cause the pressure to rise or fall with time. The main motivation for developing a command line user interface is to provide a centralized computer control to monitor and debug the gas regulator control system electronically by using a stand-alone IBM-compatible personal computer. This project was of a considerable significance for CONCOA to increase the accuracy of the gas regulator and to operate their gas regulator system electronically and from a central control location.

A user interface is developed in assembly language for Motorola's 68HC12 microcontroller on an M68EVB912B32 Evaluation Board, which enables users to communicate interactively with CONCOA's gas regulator system. The software written for this project provides CONCOA a complete system to monitor and debug their hardware using a set of commands consists of read (monitor), write (modify) and debug system operations. The debug program uses a set of commands to modify and dump any RAM and ROMs locations respectively, and is recommended only for expert users of the system, since it provides a way to access the RAM locations which are impossible using the monitor program's command set. Mainly the two well integrated monitor and control programs are used to carry out all the operational tasks. This thesis explores only the

monitor program that runs in the foreground as the main program for CONCOA system. The control program, works in the background, is used to perform AD and DA conversions to store the output pressure in the microcontroller's RAM and to calculate an equivalent voltage signal for a new set point pressure for the output respectively.

Finally, the successful testing of this software with the company's hardware provides a very strong base to suggest the further studies of this project.

This thesis is dedicated to my parents who gave me their love and prayed days and nights for my successes

ACKNOWLEDGMENTS

I would like to thank Dr. Glenn A. Gerdin for his technical support, patience, and guidance while I was completing the work for my master's degree.

I would also like to thank Dr. Leathrum and Dr. Lakdawala for being a part of my defense committee. I appreciate their time spent critiquing my thesis.

Finally, I would like to thank my beloved wife Nadia for her full support and encouragement for the completion of this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview:

The goal of this thesis[1] research was to provide an electronic control to a gas regulator system for CONCOA Corporation. To accomplish this task, a control and monitor software program was needed to run a hardware control system (as shown in figure 1.1) by using a stand-alone personal computer to control and monitor the gas pressure at the output of CONCOA's gas regulator system. The end goal of this system was to control the outlet pressure of CONCOA's gas regulator system.

1.2 What was the problem?

Control Corporation of America (CONCOA) manufactures high-pressure gas regulators, which mechanically controls the pressure at its outlet. The conventional mechanical gas regulators, even under normal operating conditions, cannot control the outlet gas pressure exactly.

As we know from the ideal gas law that: PV=NkT, provided the volume of the gas is kept constant.

*Where N = number of gas molecules, k = Boltzman's constant = 1.38 * 10-23 J/K.*

When the gas is drawn from a high-pressure gas cylinder, 'N' decreases and so does the gas pressure 'P'. This gas is stored in high-pressure (up to 2000 psi) cylinders to reduce gas volumes.

[1] The journal model for this thesis is IEEE TRANSACTIONS on DIELECTRICS and Electrical Insulation.

A gas regulator enables one to set the outlet pressure at its low-pressure side so that as the gas is used up, the high-pressure side falls but the outlet pressure remains fixed. For mechanical regulators, at steady flow rate, the outlet pressure gradually falls (single-stage regulator) or rises (double-stage regulator) with falling source pressure. Moreover, with an increasing flow rate, both the regulator types let the outlet pressure to fall. Phenomenon that could also change the outlet pressure includes excursions caused by the addition or subtraction of gas sources or sinks and/or downstream from regulator. So the actual problem was the control of the outlet pressure to within a ±0.5% full scale of the set point pressure at the gas PO manifold as shown in figure 1.1.

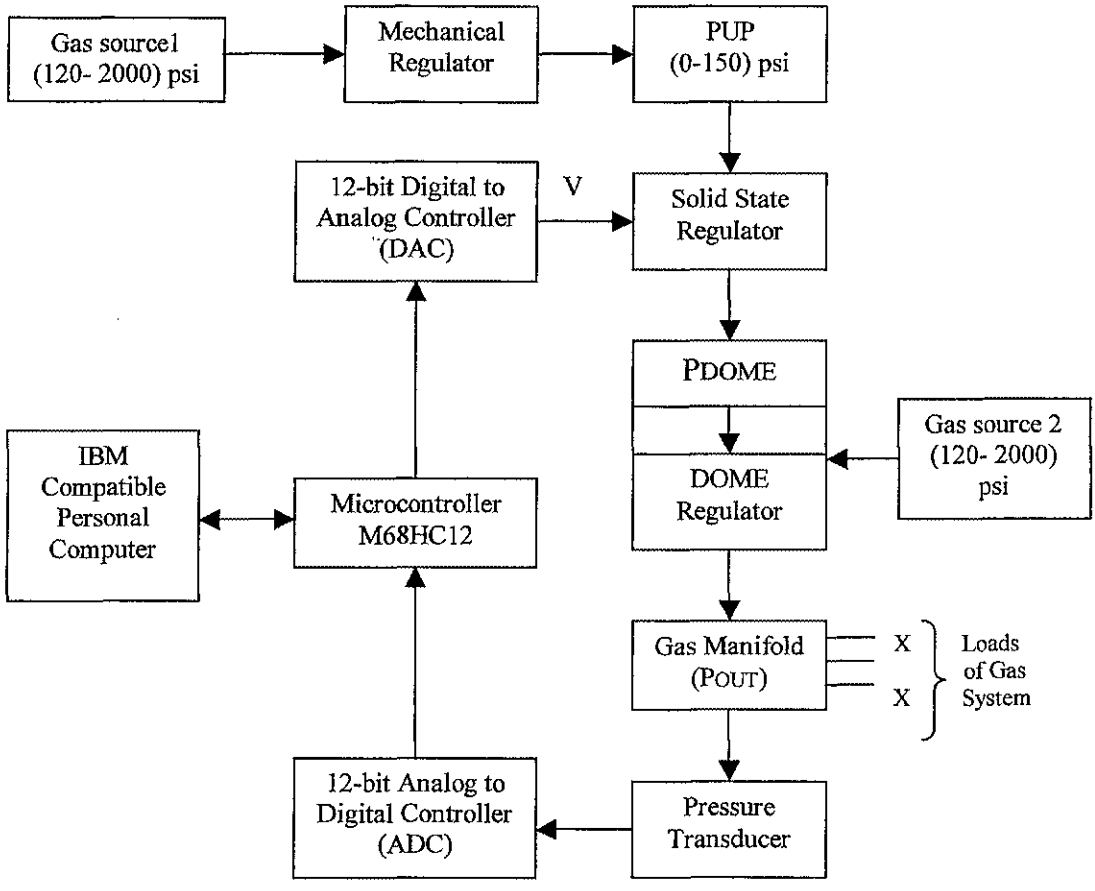


Figure 1.1 CONCOA Electronic Control System

1.3 What was the significance?

This project was of a considerable significance for CONCOA to operate its gas regulator control system electronically and from a central control location. The computerization of this gas regulator control system would increase the accuracy of the gas regulator. This project would help CONCOA to control and monitor the complete operation of their system remotely through a personal computer. The computer control provides them a command-line user interface with a set of commands, such as read, write and debug, which can be entered through a standard IBM-compatible computer keyboard. With these commands, they can now control many different system parameters, electronically, which used to be control manually by turning a knob.

Before this project, the outlet gas pressure at the cylinder banks and gas regulator used to be read by an analog meter with a needle indicating an approximate gas pressure value. Now a digital value with half-percent accuracy can be seen any time on a standard display monitor screen. The computer control also saves time, as one does not need to be physically present where the gas regulator is to modify the gas pressure to a desired value. The electronic control is provided by means of setting control parameters. The control parameters that may be set are KI (I coefficient of PID control coefficients) which provide an integral control, PS which is the set point pressure, TM (loop time) which is the time-interval between the cells to the control program and TR (ramp time) sets the number of loop-time the control program will take to change the set point pressure (soft-start-up).

The software command-set enables a user to set a desired value for a particular parameter. For example, to set a new value of 10 (only decimal number is allowed) for integral control coefficient (ki) we use 'KI (in upper case letters) command as shown below:

KI=10 followed by a carriage return.

In order to verify these changes, we use the same command with a question mark (?) as shown below:

KI? followed by a carriage return.

The expected result should be,

KI=10

This new control and monitor system provides the kind of accuracy, which was impossible with the old manual gas control system. With the help of a monitoring support in the system, now a pressure value can be set for a minimum and/or for a maximum pressure limits. If the outlet pressure crosses either of these upper and lower threshold pressure limits, the control program activates a digital dialer to notify the operator. This really restricts the end user to change the set point pressure with in a certain pressure limits.

Setting any default limits for manifold makes the system configurable and scaleable to any pressure values between the two thresholds limits. Finally, the PID control and the time features, like loop time and ramp time which follow a linear approach to setting the output pressure in steps and with some known time intervals, provide an approximate calculation for a pressure error value and a timing control respectively.

1.4 Background and Review of the Thesis:

Control Corporation of America (CONCOA) manufactures gas regulators for handling high-pressure gasses up to 5000 psi. These gas regulators control the pressure at its outlet and any adjustment to this outlet pressure has to be made mechanically at the cylinder banks and gas regulator. The engineers' team at CONCOA decided to computerize the whole process to adjust the magnitude of this outlet pressure remotely from a central location and to get an efficient electronic control on their gas regulators as shown in the 'Block Diagram of CONCOA Electronic Control System' in figure 1.1.

The pressure (up to 5000 psi) gas-regulator is a mechanically controlled device manufactured by CONCOA; however, the outlet pressure is adjusted by adjusting the

manifold pressure, which is relatively low (up to 100 psi) and thus could be controlled electronically. To regulate the gas pressure at the outlet and to monitor the state and the activity at the outer banks a pressure transducer is attached with CONCOA regulator which closes the feedback system by converting the outlet pressure Pout into an electrical signal. This electrical signal becomes a corresponding digital signal by passing it through an analog-to-digital converter (ADC) and could be easily read and manipulated by a microcontroller for further processing.

Solid-state regulators (SSR) could control 'POUT' but since it is made of silicon, the high-pressure side can only be 120-150 psi. A dome regulator outlet pressure is controlled by pressure applied to its DOME side. So,

$$\text{Pout} = P_{DOME} - \text{Poffset1} \qquad (\text{Poffset} \approx 10\text{-}20 \text{ psi})$$

So use electronic-controlled solid-state regulator to control DOME pressure.
Since,

$$\text{Pout} + \text{Poffset1} + \text{Poffset2} = P_{DOME} + \text{Poffset2} \leq P_{UP} \leq 150 \text{ psi}$$

The system 'UP' pressure side of DOME regulator is 2000-5000 psi, so there is a substantial gas supply for the system.

Pout is controlled by PDOME, which is controlled by the solid-state regulator's feedback input voltage V. This solid-state regulator, which has a built-in pressure transducer, generates a Pressure PDOM proportional to voltage V. The 12-bit ADC converts pressure 'POUT' to a digital-word and this number is transferred to microcontroller. This hex number is then compared to Pset (set-point pressure) and a control voltage number is calculated by the microcontroller using a PID algorithm. This control voltage number is transferred to the DAC-chip, which in turn converts it into the corresponding analog voltage 'V'. This voltage 'V' is applied to the solid-state regulator which adjusts PDOME and hence PO to bring PO $\rightarrow$ Pset.

CHAPTER 2

PROBLEM FORMULATION AND GENERAL SOLUTION

2.1 Problem Discussion:

As discussed earlier, Control Corporation of America (CONCOA) manufactures high-pressure gas regulators, which mechanically control the pressure at their outlets. Since the control system is mechanical, it might induce fluctuation in the outlet pressure causing its level to fall down or up beyond a certain default limit. It might also include a gradual decrease in upstream pressure due to the uniform gas flow through the regulator and/or excursion caused by the addition or subtraction of gas sources or sinks upstream and/or downstream from regulator. So the actual problem was the feedback control of the outlet pressure at the cylinder banks and gas regulator.

To overcome this problem and to adjust and maintain a desired outlet pressure of a high-pressure gas regulator, a project was designed to electronically control the outlet pressure through a central control location using a stand-alone personal computer.

The goal of this project was to design a control, monitor and debug system for CONCOA's high-pressure gas regulator. The key to this design was to electronically control the pressure, which is done by the control program as described in section 1.4 in chapter 1. To provide a computer control over the whole system, the microcontroller is attached to a personal computer through a serial communication link such as RS-232 and screen emulator software such as: ProComm, HyperTerminal, miniide, etc. The object of the software program developed in this thesis, is to develop a monitor program so the operator can change the control parameters; such as the set-point pressure 'PS', etc.

2.2 Hardware Selection:

As stated by CONCOA officials, the following are desirable control accuracies:

- |Pout-Pset| < 0.5 % of full scale

The system that could meet the above design specifications that is the pressure difference at the outlet and the set pressure to be less than 0.5%, forces the digital system to be 10 bits or more. To approach the less than 0.5% accuracy for the overall system, we need a 10 bit analog-to-digital converter (ADC) for reading the output of the transducer as does the digital-to-analog converter (DAC) needed to convert the digital control signal into the analog voltage necessary to operate the solid state regulator. Also the microcontroller architecture should be at least 10 bit, so the digital control signal generated by the controller's program is sufficiently accurate.

Following is the details of the hardware selection and its significance in making the CONCOA system work with all desired functionalities and accuracies.

- Microcontroller- Motorola's 68HC12
- Digital-to-analog converter MAX7645
- Analog-to-digital converter ADS7824
- An IBM-compatible personal computer

Figure 2.1 is a block diagram that explains how these components are connected together to accomplish the goal of providing a centralized electronic control to CONCOA's gas-regulator system by using an IBM-compatible personal computer.

Below is a detailed description of all the components mentioned above.

2.2.1 Microcontroller- Motorola's 68HC12:

68HC12 is Motorola's 16-bit microcontroller. It means it has enough capability to provide a 10-bit accuracy. It has a 16-bit data bus, a revised and better command set than Motorola's 68HC11 microcontroller and a background mode, which enables the communication with a PC, via an RS-232 serial port, without interrupting the main control program operating on the microcontroller.

It also has sufficient parallel I/O interfaces, also known as ports such as: A, B, E, P_e etc. These ports provide a way of I/O communication with the 12-bit ADC and DAC to read the outlet pressure (PO) at the gas regulator and to set a desired pressure (PS) in terms of analog actuating voltage to the solid-state regulator.

This controller is a programmable, 8MHZ, stand-alone 16-bit controller, fully capable of performing the tasks in hand, and is relatively low cost.

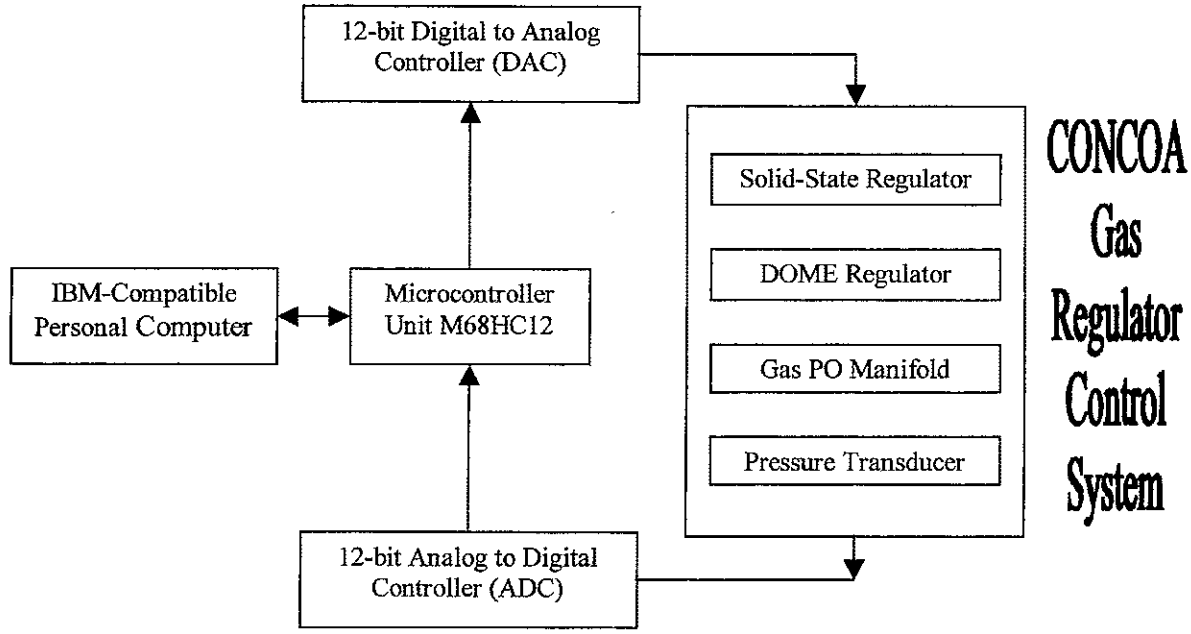


Figure 2.1 Computer control for CONCOA's gas regulator system

2.2.2 Digital-to-Analog Converter:

For this project a 12-bit buffered multiplying DAC, MAX7645 (for more details refer to the web site of Maximum integrated products® at www.maxim-ic.com) is chosen to convert a digital signal into an equivalent analog voltage. It is TTL and CMOS compatible with a +15V supply voltages and available in 20-lead narrow DIP, surface mount small outline and PLCC packages. This device is fully specified for operation over the industrial -40°C to +85°C range. The MAX7645 directly interfaces to 8- and 16-bit microcontrollers and processors and are loaded by a single 12-bit wide word using standard control signals and its 12-bit data latch. At 12-bit it has sufficient accuracy for the problem specifications. Moreover, its analog output voltage range of 0-10 Volts coincides exactly with the input voltage range of the solid-state regulator. Thus, all 12 bits are utilized.

2.2.3 Analog-to-Digital Converter:

The ADS7824 (the electrical characteristics are retrieved from the BURR-BROWN® web site at www.burr-rown.com/databook/ADS7824.htm) is a low-power, 12-bit sampling CMOS A/D converter with a four-channel input multiplexer, S/H, clock, reference and a parallel serial microprocessor interface. It can acquire and convert 12 bits to within ±0.5% LSB in 25μs max while consuming only 50mW of maximum power. It has 12data lines, with which to communicate with MC6812 in an 8 most significant bit (MSB), 4 least significant bits (LSB) format and can be controlled by the MC6812 over 4 additional parallel interface or port lines. This ADC is multiplexed to accept several inputs, permitting the reading of both output pressures, i.e., from the transducer and from a locally adjustable analog voltage. The 12-bit ADC was chosen to provide sufficient accuracy (0.5%). However, the output of the analog pressure transducer circuit had a range of about 0-3.3 Volts, so the actual precision of the ADC was only 675 (2^{12} * 3.3 / 20) only slightly better than 9-bit.

2.2.4 An IBM-Compatible Personal Computer:

To obtain a computer control on CONCOA's gas regulator system, an IBM-compatible personal computer is used with a standard display monitor, a standard keyboard and a RS-232 serial interface. The one end of the main controlling hardware, the microcontroller evaluation board, is connected to the computer via the RS-232 serial interface and the other end to the ADC and DAC via the parallel interface lines also called ports for data input and output as shown in the Block diagram of a Computer control for CONCOA's gas regulator system in Figure 2.1.

The other end (output) of the DAC is connected to the solid-state pressure regulator, which converts the DAC analog voltage to an equivalent DOM pressure. The analog input of the ADC is connected to the pressure transducer amplifier output and to a variable set voltage. The latter is an adjustable voltage divider and used to set the set point pressure at the regulator if required.

2.3 Software Requirements:

To acquire computer control for CONCOA's gas regulator system, a software program was needed to communicate with the company's hardware using a microcontroller to control, monitor and debug the complete system operation. A monitor program was needed with a default prompt so that users can input and verify new parameters' values as command-line arguments while the control program is operating in the background. The software requirements for this project are further described below:

- For an interactive and friendly user-interface, a set of system messages to be displayed in an event an error is occurred (ex: syntax error) or to alarm the user about the current status of the system (ex: control program is enabled).

- Backspace feature to modify a mistake on the command line before the arguments are actually submitted using 'enter' key of the keyboard.
- Command-line help file with a brief description of the available commands and the set of rules for their syntax.
- Safety limits on setting control parameters to prevent the system from getting unstable.
- Memory-modify and memory-dump commands to assist the present and the future developments of the system, while the program is running from the Flash EEPROM on the microcontroller unit.
- Startup of the monitor program in the foreground as the main controlling program for the system and its capability to enable the control program to run in the background.

2.4 System Operation:

Once the embedded hardware is hooked up with the IBM-compatible stand-alone PC, the microcontroller starts executing binary codes from Flash EEPROM and immediately issues CONCOA prompt on the display terminal and then transfers the control to a 'wait-forever-loop' to wait until the user input data through the computer keyboard. This loop is basically the command-line interface, which waits for a user to input the commands for various system parameters. When a user presses carriage return (enter key on standard keyboards), the system takes it as a point of submission for an available command for the monitor or debug program. This action is the same as pressing 'enter key' after writing a command for an operating system like DOS™ or UNIX.

The complete operation of the system can be described in the following three steps:

- Program Execution from Flash EEPROM
- Initialization using default parameters' values
- Initialization and Execution of the Control program

2.4.1 Program Execution From Flash EEPROM:

The software for this project has overwritten the Motorola's SDBUG12 program into the Flash EEPROM. Once the evaluation board is powered up, it starts its execution form the Flash EEPROM where it finds a jump instruction to the main program of CONCOA software. The program then initializes the "Monitor" program and heads towards the next step.

2.4.2 Initialization Using Default Parameters' Values:

After going through the initialization part, the program then loads the default parameters' values and prints them on the monitor screen. Finally, the program displays a message for the operator to change the default parameters' values, if desired, and then displays the CONCOA prompt. Now the CONCOA command-line interface is ready to execute commands and at this point the "Monitor" program is fully initialized and enabled. To enable the "Control" program the end user must need to enter 'DN' (for done - case sensitive) command followed by a carriage return, after changing the default parameters' value to his own. If the default parameters' values are not modified, the end user still needs to enter 'DN' for the initialization and execution of the 'Control' program.

2.4.3 Initialization and Execution of the Control Program:

Finally, when CONCOA command-line interface reads a 'DN' command followed by a carriage return, it initializes the main control program and one of the 68HC12's timers to get into the 'Control' program after every 'TM' ms. Where 'TM' is a set able parameter whose default value is 8, which was found to be optimal during the system testing. This is because the complete control program is written in the timer

interrupt service routine and since the timer interrupt is set to occur after every 'TM' ms, the control program takes control from the monitor program every time the microcontroller services a timer interrupt. The control is always transfers back to the monitor program whenever the system finishes servicing the ISR. The main program then runs with both its Monitor and Control programs enabled.

The program will get back to the first step that is start executing the software codes from the Flash EEPROM every time when the evaluation board is reset.

CHAPTER 3

SOFTWARE APPROACH TO THE PROBLEM

3.1 Software Techniques

The software program for this project is written in assembly language for Motorola's 68HC12 microcontroller. A top-level flow chart is made first (refer to 'Basic programming flow chart, figure 3.1) and then the second-level flow charts and then the other levels as shown in section 3.2. This program is a combination of a top-down and structured programming design style. A modular programming technique is adopted to write the complete software program for this project. Each module is programmed and tested separately before its integration into the main program. On the top level, the complete monitor program can be divided into five sub-programs, which are initialization, display default parameters, display prompt, wait for user input and service user request. These sub-modules are further divided into smaller programs and so forth. Some modules work independently as a subroutine and can be used with other main modules. For example the memory dump and memory modify commands use the same sub-modules to calculate and print the memory addresses as shown in 'Common Debug Command Routines' in figure B4 of appendix B. Most of these modules are generic in nature and can be easily modified to add or remove new parameters. One of the example of such a module is 'syntax check and pass parameter value' as shown in figure 3.7.

This modular programming technique is the same as Gene H. Miller [1] describes in detail in *Microcomputer Engineering*. The main modules are linked together in a top-down design fashion where as the individual modules are coded using the structured programming techniques. As written in the book referenced above, the top-down design technique parts the complete software program into some very basic or parent modules (refer to 'Basic Program Flow Chart' in figure 3.1), which are further divided into other

sub-modules as the child modules. This child-parent tree of the sub-modules grows until the finest detail of the program is achieved. The structured programming technique provides a very strong relationship and integration between the individual software modules. A good example of this kind of programming technique is shown in 'syntax check and pass parameter value' in figure 3.7, which is a nested if-then-else loop to branch to a sub-module when a condition becomes true and in the next level it behaves like a switch-case statement to come out of the loop when a parameter name is matched with an input provided by the user.

Each written module is tested in the RAM on an individual basis as a complete program using the evaluation board's factory-installed 'DBUG12' monitor/debugger program. The final testing of the monitor program is done in the Flash EEPROM because it was too big to fit in the RAM of microcontroller's evaluation board.

3.2 Main Modules and their Flowcharts:

The software model for this project is explained in the 'Basic Program Flow Chart as shown in figure 3.1. It illustrates how the main program and the control program are integrated to pass the information back and forth. As shown in the figure, the system, after the power is turned on, initializes various registers, counters, pointers, and many other parameters. It then displays the default parameters' values and prints a message on the monitor screen asking the system operator to modify the default values, if needed. The system then prints a default prompt (<CONCOA>) for the command-line user interface. The next step is to transfer the control to a 'wait-for-ever' loop for acquiring the user input at the command prompt. At this point only the monitor program is available to service the user requests. In other words, the control of the program only navigates within the 'Main Program Loop' as shown in figure 3.1. To enable 'Control' program, the user has to type in the 'DN' command (use only upper case letters) followed by a carriage return. The 'DN' command basically enables a timer interrupt to occur after every 'TM' ms, and, since the complete 'Control' program is written in the timer

interrupt service routine, the main program transfers the data and the software control between the monitor and the control program whenever it senses the occurrence of a timer interrupt or finishes servicing the interrupt service routine.

The complete software program for CONCOA may be divided into the following five basic modules as described in the previous section:

- Initialization
- Default values
- Display Prompt
- Wait for user input
- Service user request

Figure 3.1 contains a description of each software module and its flow chart.

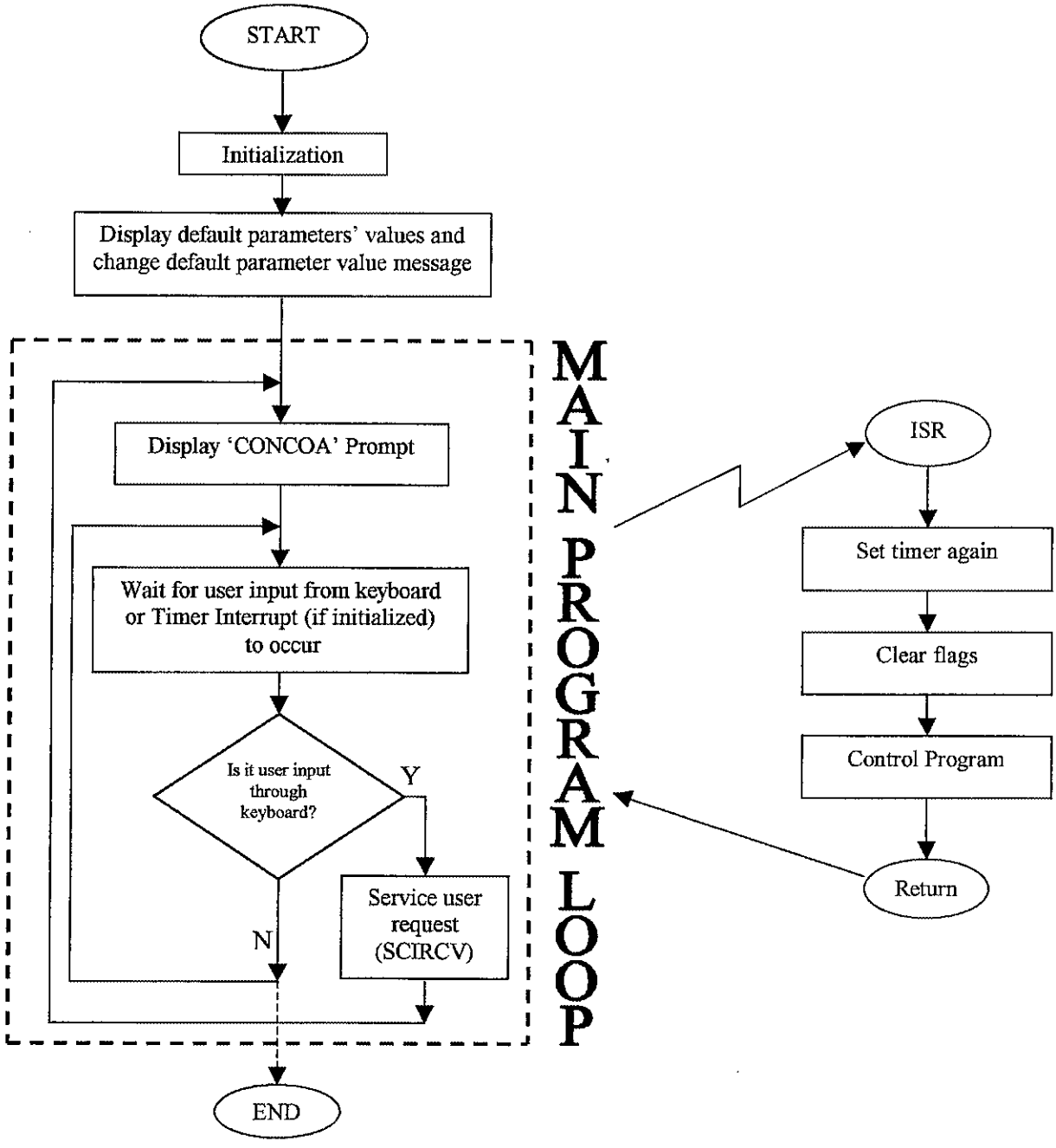


Figure 3.1 Basic Program Flow Chart

3.2.1 Initialization:

The complete program initialization by means of RAM, Flash EEPROM and Main program location is shown below in 'initialization' flow chart in figure 3.2 and can be described as follows:

The RAM location:

The 68HC12 microcontroller has a 1K on-chip RAM, starting from $0800 to $0BFF. The RAM location, set aside for the dynamic data, is called RAM-Data-Monitor segment ($0800-$08CF) and RAM-Data-Control segment ($08D0-$09A0). The following initialization is done in the monitor data segment as shown in figure 3.2.

- Reserve memory location(s) for 68HC12 registers
- Reserve memory location(s) for CONCOA's permanent data registers
- Reserve memory bytes for counters, flags, pointers and other temporary registers.

The Flash EEPROM location:

68HC12 has a 32 KB of on-chip Flash EEPROM, starting from $8000 to $F67F. The Flash memory location, set aside for static data, is called ROM-Data-Monitor ($8000-$8FFF) and ROM-Data-Control ($9000-$9FFF). The following initialization is done in the RAM-Data-Monitor segment as shown in figure 3.2.

- Define byte(s) for parameters to compare with user input, CONCOA prompt, read and write values, carriage return/line feed, default parameters' name and boot up messages.
- Define byte(s) for system error and system status messages.
- Define byte(s) for command line help for CONCOA software.

Main program location:

The following initialization is done in the main program location starting at $A000 in the Flash EEPROM as shown in figure 3.2.

- Setting and clearing 68HC12 registers
- Setting and/or clearing counters, flags, and pointers
- Loading default parameters' values

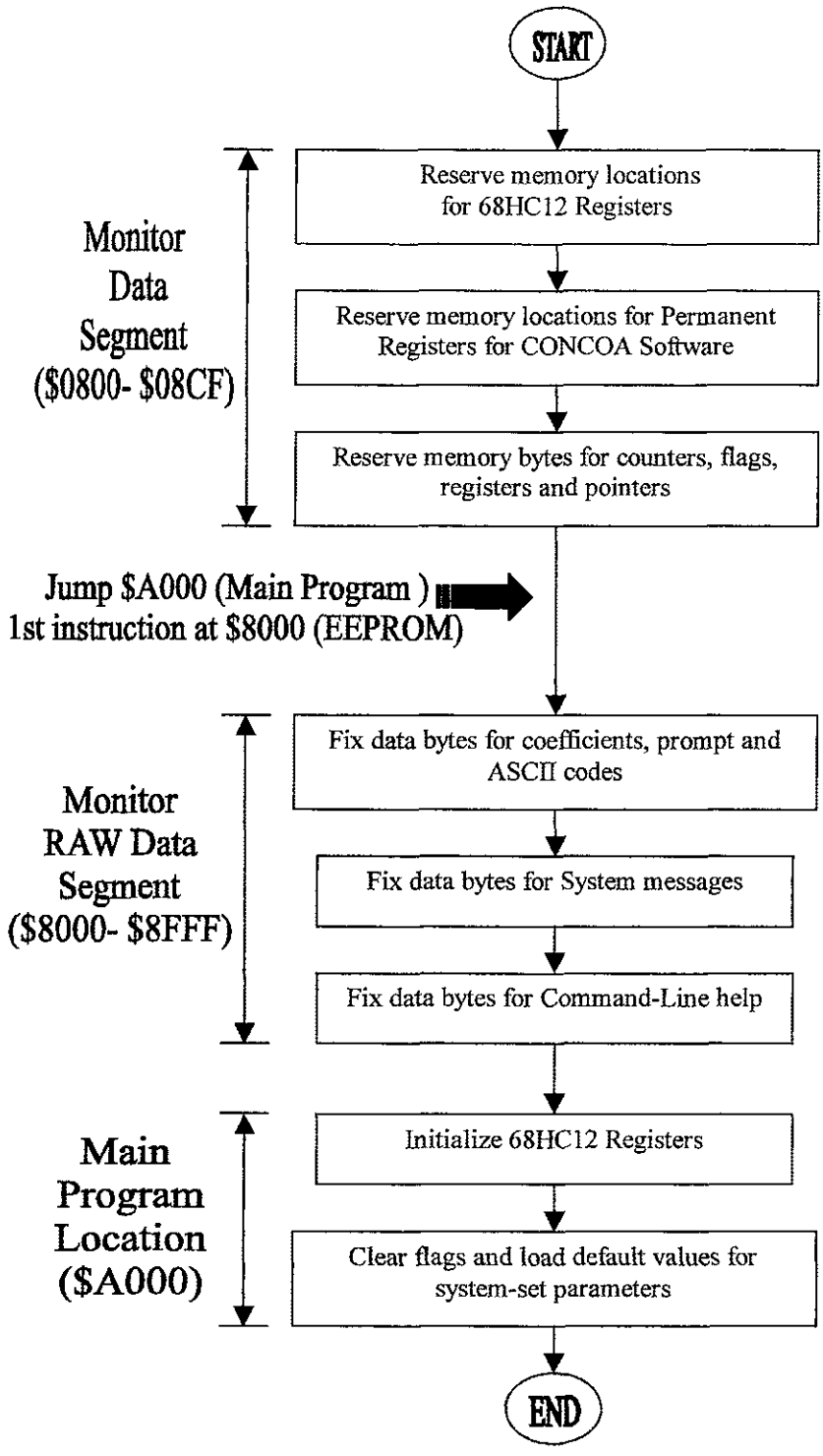


Figure 3.2 Initialization

3.2.2 Default Values:

This program module is used to load the system with the default parameters' values for the following coefficients:

KP, KI, PS, PU, PL, TM, TR, IFB, INT, SNDV

The default parameter values for the above parameters are shown in table 3.1.

Parameter Name	**Default value in Hex**	**Equivalent Decimal Value**
Set point pressure (PS)	03FF	102.3 psi
Pressure Upper Limit (PU)	0999	245.7 psi
Pressure Lower Limit (PL)	00CD	20.5 psi
P coefficient of PID control (KP)	64	100
I coefficient of PID control (KI)	11	17
Ramp Time for soft startup (TR)	1	.1 second =100ms
Loop Time (TM)	8	8 ms
Set pressure from evaluation board (IFB)	0 (Boolean)	
SNDV (DS)	3	3

Table 3.1 Default Parameters' Values

These values can be easily modified using the write-commands for a corresponding parameter. After the default values for the above mentioned parameters are loaded into the RAM, a message is displayed on the terminal screen to change the values of any default parameters as shown in the 'Message Display Module for Default Parameters' flow chart, in figure 3.3. When the system finishes displaying this message

on the monitor screen, it prints a default CONCOA prompt on the screen and transfers the control to the 'wait-for-ever' loop to acquire user input at the command prompt. As discussed earlier, at this point the monitor program becomes ready to service the user input commands.

Now, since at this stage of the program execution the control program is not enabled yet, the user can change the default values, if needed, before pressing 'DN' to enable the control program.

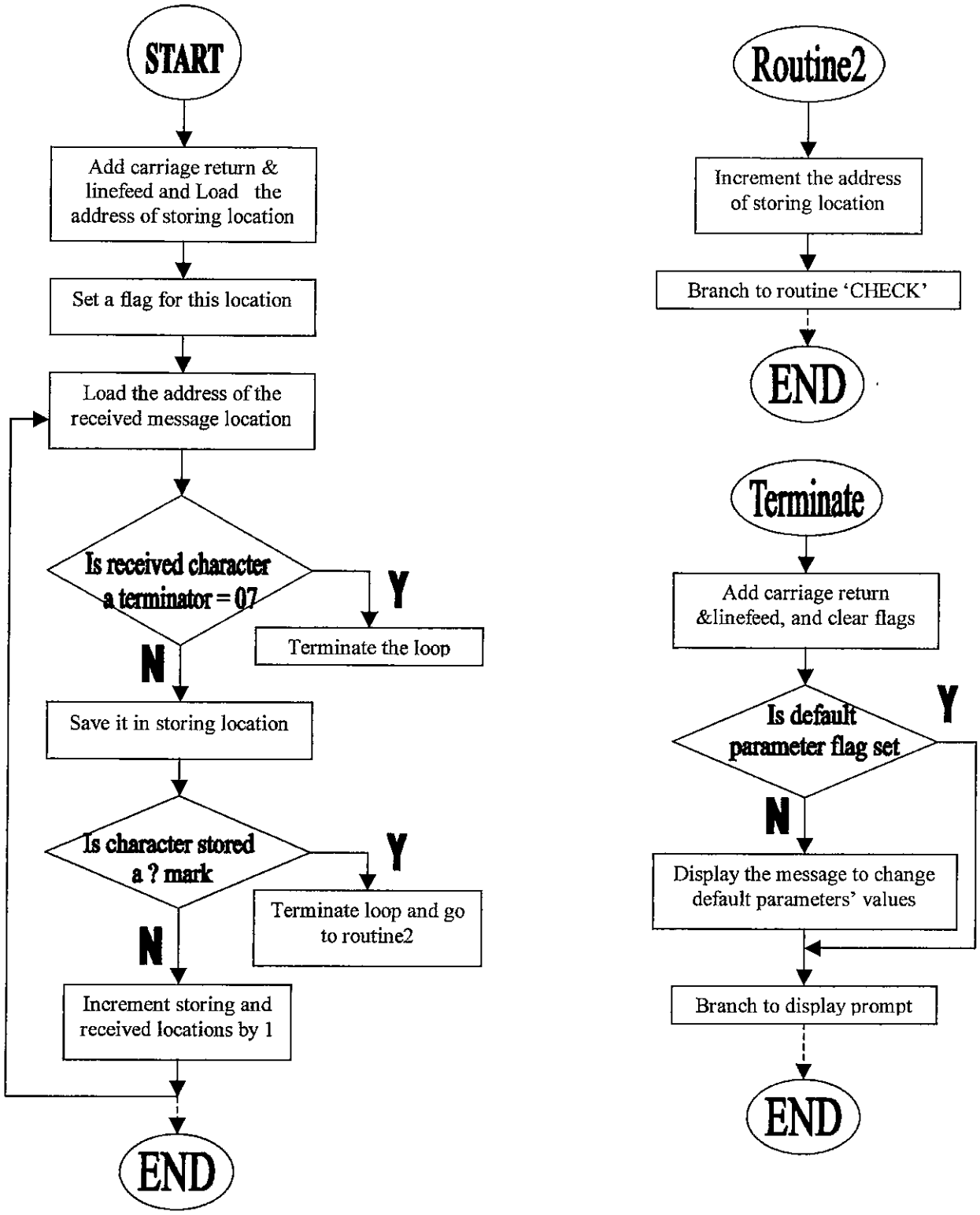


Figure 3.3 Message Display Module for Default Parameters

3.2.3 Display Prompt:

This program module is used to display a default prompt (<CONCOA>) on the terminal screen. This prompt is the command-line interface for entering CONCOA commands to read or write or debug parameters' values. Once the default parameters' are loaded by the system, this prompt is issued immediately on the monitor screen. The presence of CONCOA prompt on display terminal indicates that the system is ready to accept commands for reading and writing parameters' values and debugging any memory locations using memory modify and memory dump commands. The system always prints this prompt after completing a user request. Once a command is serviced, all the counters, pointers and flags are initialized before printing the prompt on the screen again as shown in 'Display CONCOA Prompt' flow chart in figure 3.4.

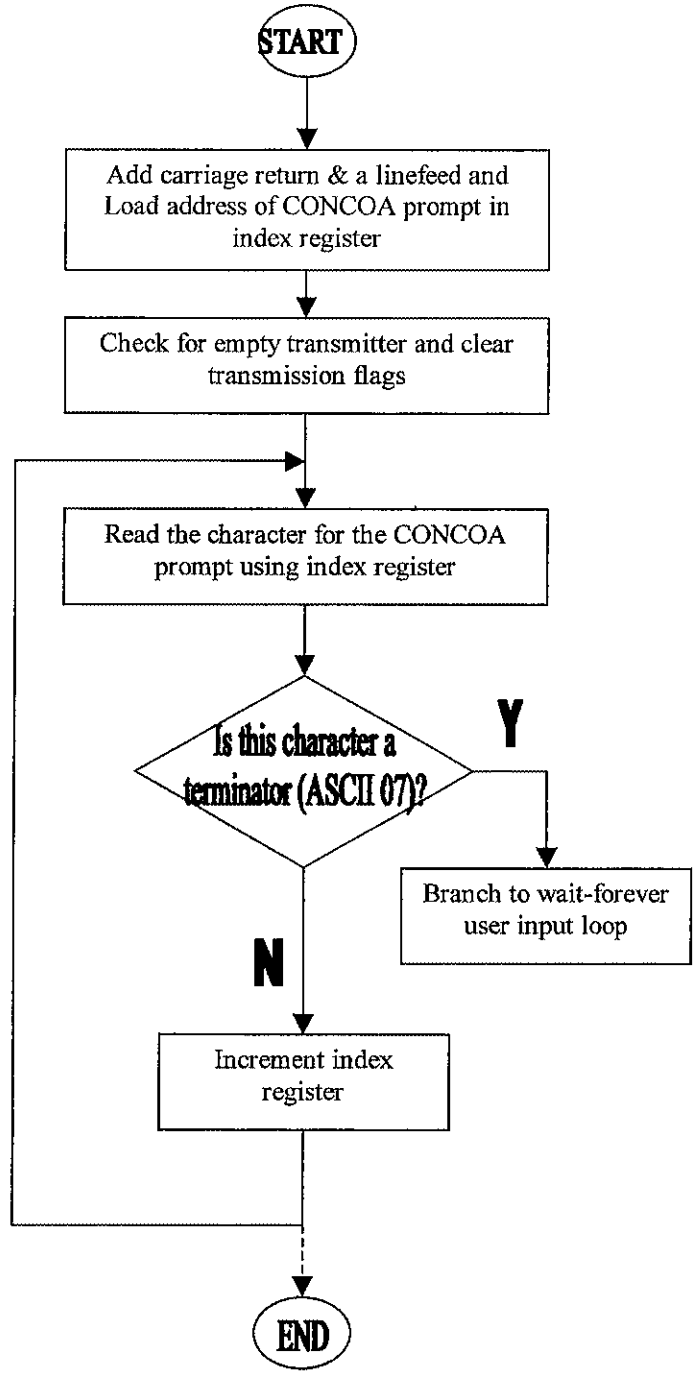


Figure 3.4 CONCOA Prompt

3.2.4 Wait for User Input:

This module is used to acquire user input through keyboard for further processing. After displaying a CONCOA prompt on the screen, the control of the system is actually transferred to this module to wait until a new command is input through the keyboard. This is a wait-for-ever software loop that can be terminated either by pressing a carriage return (Enter key on IBM-compatible keyboard) after entering a user command or by overflowing the input-store buffer for capturing the command line arguments. Once a carriage return is detected by the system, the loop terminates normally and the system gets into the next phase of servicing the user request. If the loop does not terminate normally, e.g., the input buffer exceeds the allowed limit of 16 bytes, the system shows an error message and the control is transferred back to the same loop. In the next phase of servicing a user request, the command line arguments are checked for syntax, value, or any other possible typing errors. For an abnormal termination of this loop, the system reinitializes the counters, pointers and flags, and displays the CONCOA prompt again before going back to the wait-loop as shown in the flow chart in figure 3.5.

This module also has the functionality to allow the user to correct any typing error or mistakes before finally submitting (by pressing 'Enter' key) a command. The backspace key (sometimes symbolized as a left arrow key on an IBM-compatible keyboard) is used for deleting a previously typed character or a false parameter's value. This feature not only deletes a mistyped or unwanted argument but also removes its entry from the input capture register and subtracts the input counter accordingly to get to the right location where the data has to be stored. To make the monitor screen look user friendly, the display terminal is also refreshed to show that the mistyped character is removed from the terminal screen.

This module also distinguishes between a floating point and an integer value for a command parameter and sets or clears a period flag to pass on the status of the input data type to the next phase of servicing a user request. There are also some other flags and counters used as a purpose to keeping track of syntax and other potential errors.

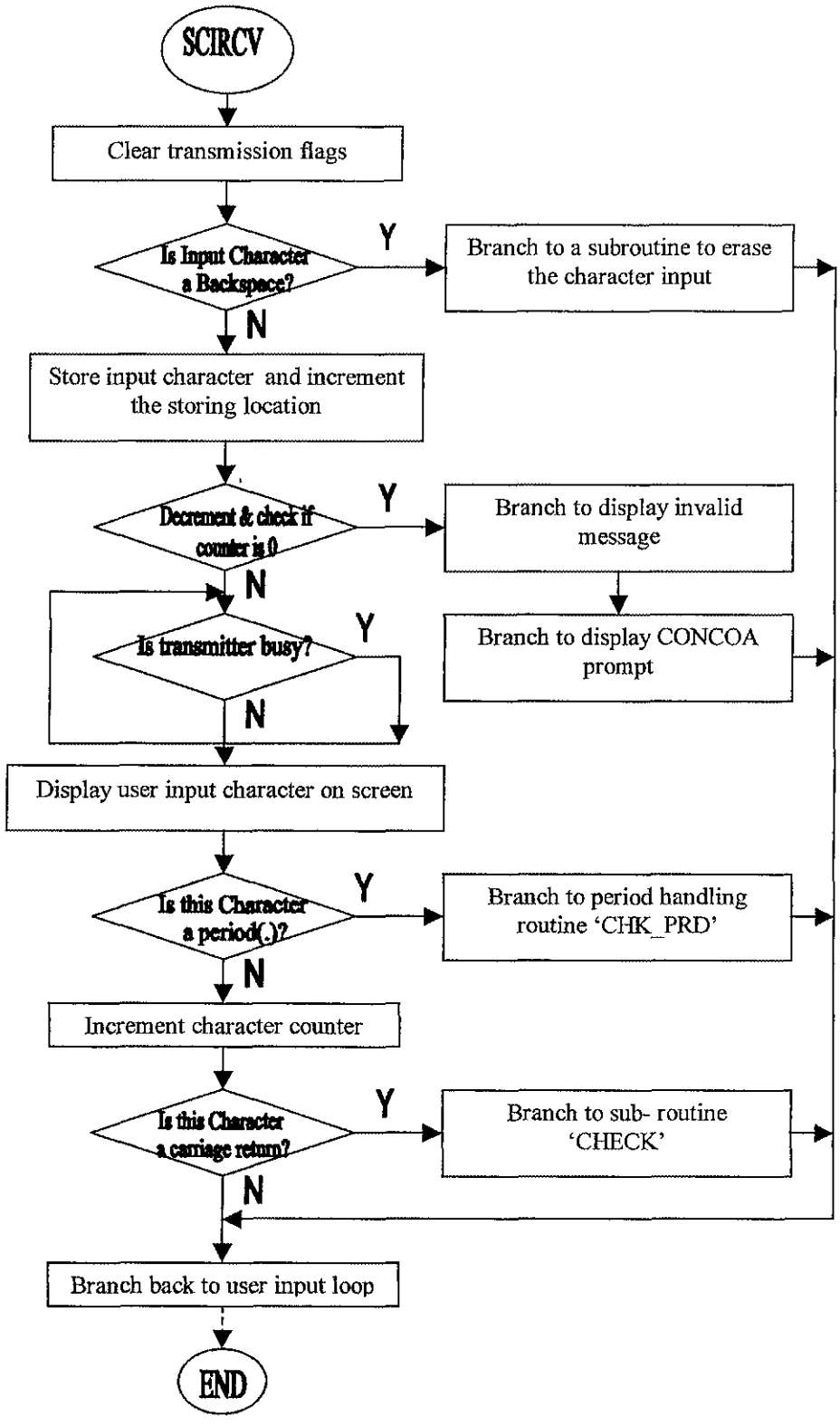


Figure 3.5 Service Routine for User-Input

3.2.5 Service User Request:

This is the most important and the largest program module for CONCOA software. It basically incorporates the most basic programming modules to carry out the tasks, such as: error checking, reading and writing the parameters' values and debugging the system, etc. This software module has the following integrated sub-modules:

- Error Checking
- Parameter Passing
- Read Commands
- Write Commands
- Debug Commands

Error Checking:

This sub-module is further integrated into many small software programs to perform the error checking and correcting for the command line arguments. It typically includes checking the argument's length, its syntax and value before carrying out the user requests. It then checks if the request is a read, write or a debug command as shown in the 'Syntax check and Service User Request' flow chart in figure 3.6. It also checks for a command to enable the control program. If, at any point, an error is found in the command line argument, a corresponding error message is displayed on the terminal screen followed by a CONCOA prompt on the very next line. To check the syntax of commands, the system compares the command line input arguments with a set of arguments already saved at a permanent memory location in controller's Flash EEPROM. It works in a chronological order to compare an input command with the saved commands in the command-database within the Flash EEPROM. Since, all the commands (read, write and debug, etc.) are two-character or two-byte long; they are divided into small sections. For example, for PS, PU and PL, it first checks for the letter P (in uppercase). When a match is found, it then compares the very next letter entered at command prompt with S, U or L (all in uppercase) in the command-set-database for the above-mentioned PS, PU and PL commands respectively. If, for example, a match is

found for letter S with the second input character received at the command prompt, it comes out of the loop assuming a PS command is entered at the CONCOA prompt as shown in 'Syntax error and pass parameter value' flow chart in figure 3.7. If the first command line argument is matched with the letter 'P', but the second command line argument doesn't match with any of the letters shown above, the system displays an error message and goes back to the main 'wait-forever' user input loop.

This programming technique is analogous to the switch-case statement in C or any other high-level programming languages, where it breaks and stops looking for any other case-statement when a match is found. A similar programming technique is used for checking the other parameters as shown below:

KP, KI or TM, TI, TR, etc.

The maximum and minimum arguments' length for a write-parameter value command varies with the type of command. For example for a floating- point write-parameter value command, like PS, it should be between 6-byte (ex: PS=1.1) and 8-byte (ex: PS=999.9) respectively. Where as for an integer write-parameter value command, like KP, it should be between 5-byte (ex: KP=1) and 8-byte (ex: KP=65535). For a read-parameter value command the length is fixed to 4-byte (ex: PS? or KP?). An error message will be displayed when these arguments' length limits are violated. Since all these values should be entered as decimal values, an error message is displayed to alarm the user, incase, a non-decimal value is entered through the keyboard. The way this module checks this error is very simple. Since the ASCII codes for the decimal numbers from 0 to 9 are from 30 to 39 respectively, the system logically AND these codes with a binary mask of 00001111 (0Fh) and then compares the result to see if it is less than 0 or greater than 9.

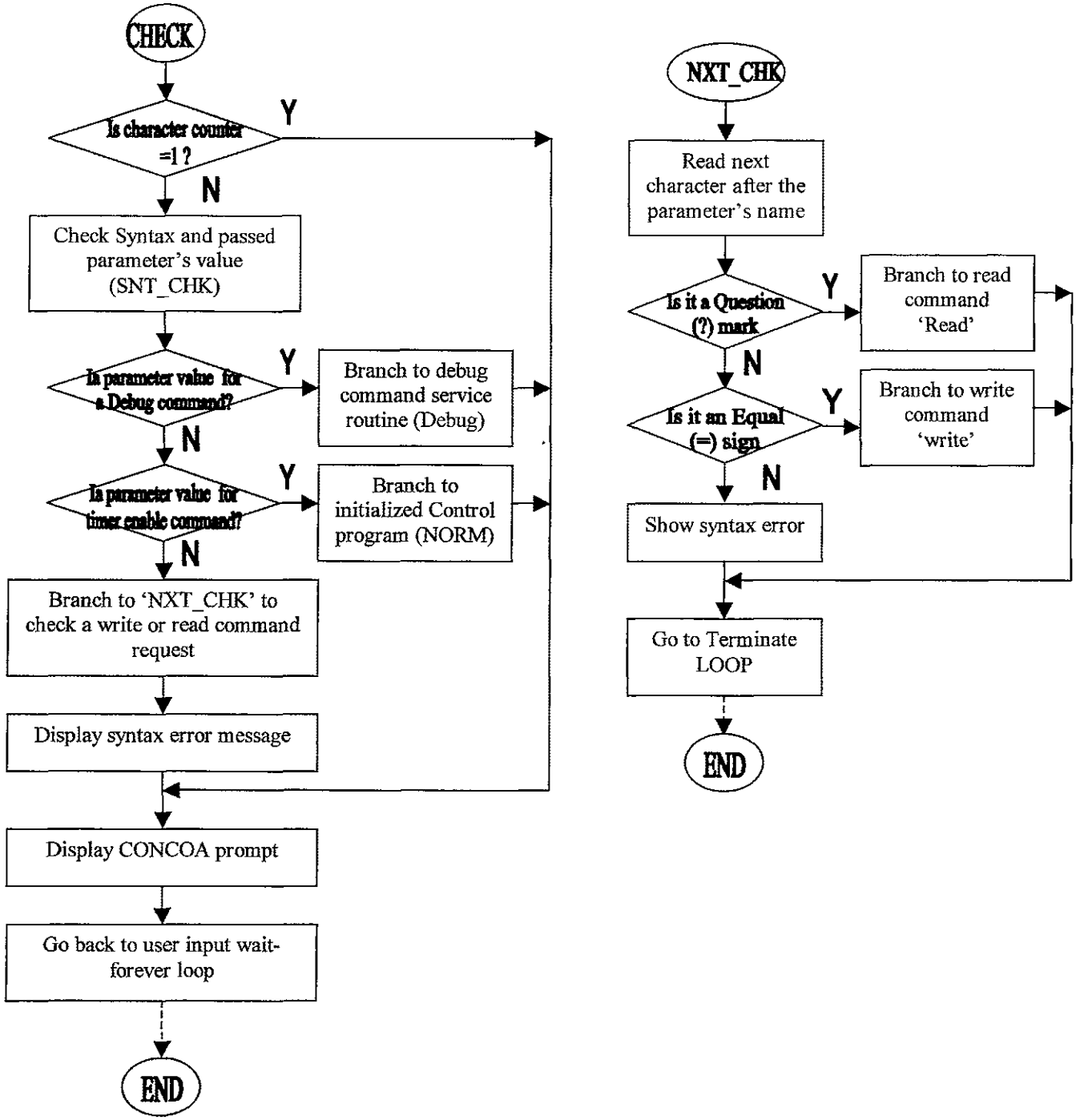


Figure 3.6 Syntax Check and Service User Request

Parameter Passing:

A parameter passing technique is used to recognize which command is entered at the command prompt and what memory location the data has to be stored or read for a particular parameter. When the system successfully identifies a command, it then assigns it a reference value, which is then passed to the next phase of operation. This reference value is two-byte long for every command. All the parameters are assigned a permanent two-byte memory location in the RAM and they have a predefined and constant distance among them. Starting with KP, whose address is taken as the base address, each parameter is two bytes ahead of the others. This known distance is used as a reference and thus added to the base address (address of KP) to get the actual memory location for a particular parameter to carry out a read or a write command request as shown in 'Syntax check and Pass Parameter Value' flow chart in figure 3.7. This parameter passing technique is same as passing a value from one function to the other in C programming language. The next step after passing a parameters' value is to check whether it is a read or a write parameter value request. For a read command the system looks for a question mark (ex: PS?) and for a write command an equal sign (ex: PS=123.4), immediately after the parameters name as shown in 'Pass Parameter Value for other Commands' flow chart in figure B2 (see appendix B).

Since, the debug commands are entered at the command prompt using different set of rules than the read and write command set, the parameter passing module (as shown in "Syntax check and Pass Parameter Value' flow chart in figure 3.7) transfers the control directly to the debug module as shown in 'Pass Parameter Value for other Commands' flow chart in figure B2 (see appendix B).

The same technique of transferring the control directly to the respective module is used for 'CD' and 'DN' commands. The 'CD' (coefficient display) command is used for displaying the most current values of all the parameters at a time. When system finds a 'CD' command it basically uses the same routine that it used to print the default parameters' values at the time of system initializations. The 'DN' (abbreviated for done)

command is used to enable the timer interrupt to occur after every 'TM' ms. Since the control program is written in the timer interrupt service routine, this command actually enables the control program in the background to take control of the system to carry out AD and DA Conversions and the PID control calculations.

For example, if we have the parameters in the following order, considering KP's address as the base address:

Address	**Parameter's Name**	**Distance from the Base Address (Reference)**	**Effective Address**
$0800	KP	$00	$0800 + $00 = $0800
$0802	KI	$02	$0800 + $02 = $0802
$0804	PS	$04	$0800 + $04 = $0804

Table 3.2 Parameter Address Calculations

Where $0800 is the base address and the parameters following it (KI and PS) should always be two-byte apart from each other as shown in table 3.1. If, for example, a parameter, such as: KI is entered at the command prompt, a reference value of 02 is passed on to the next phase which could be a read or a write value operation. This reference value of 02 is then added to the base address (address of KP = $0800) to get the actual address of KP, which is $0802 for this example. For KP and PS this value should be 00 and 04 to obtain a corresponding actual address of $0802 and $0804 respectively. This shows that these parameters form a block of reserved memory locations which can be placed anywhere in the RAM; however, their address locations should always be fixed with respect to each other in order for this technique to work properly. If a parameter's actual memory location needs to be changed, its reference value should also be changed

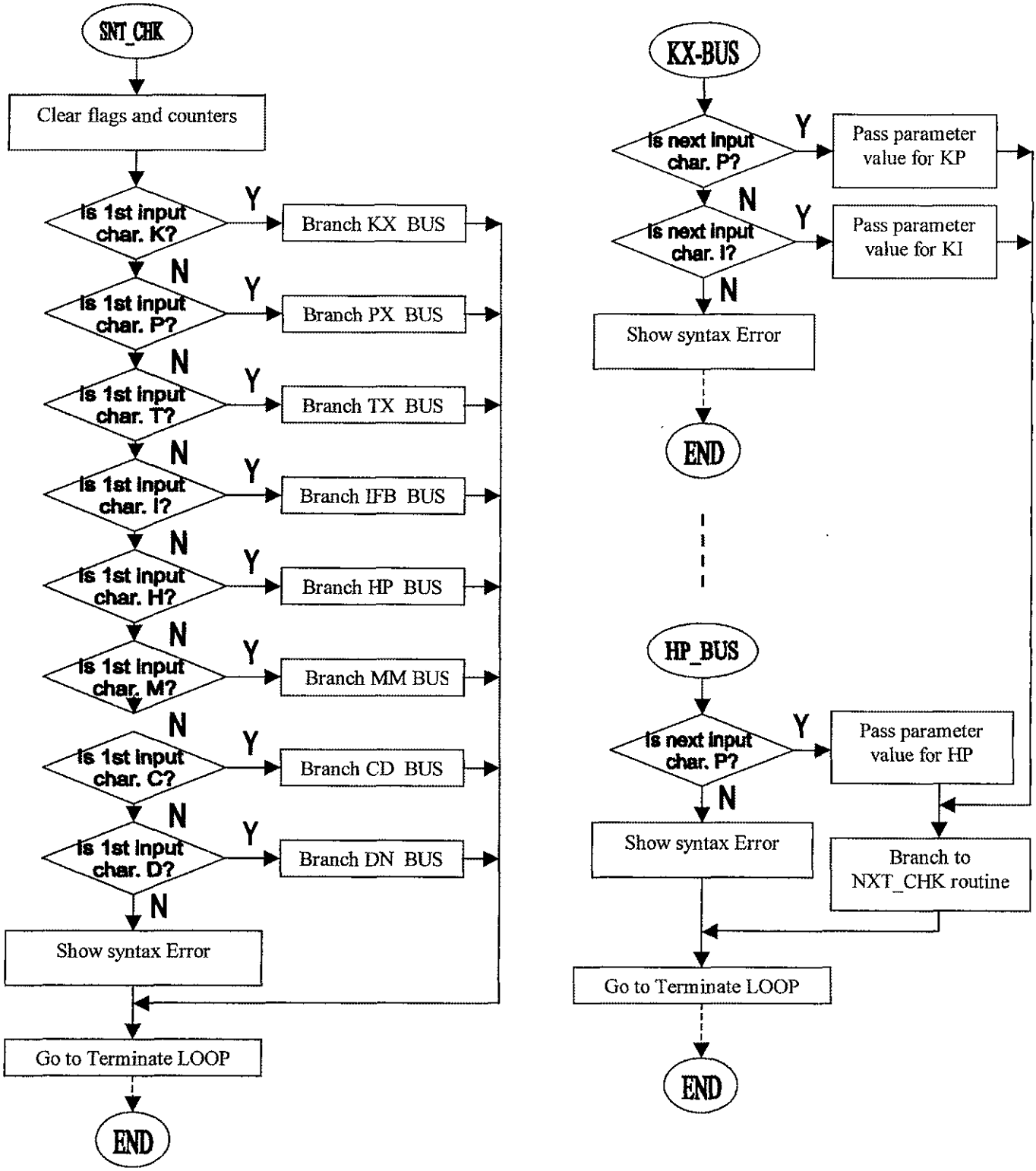


Figure 3.7 Syntax Check and Pass Parameter Value

to obtain the actual permanent location for a read or a write value operation. Now, since the position of different parameters is constant with respect to the KP's address location, changing the memory address of a parameter without its respective reference value may cause some serious software problems.

For read-only parameters, such as: PO, PB or HP, this technique is used to recognize the command type and the reference value is used only to identify which read-only command has to be serviced.

Read Commands:

This sub-module is used to reading parameters' values. There are basically two kinds of parameters, floating point and integers. The floating-point type parameters are those whose values are always displayed with a decimal point and the integer type are those, which must be entered without a decimal point at the command prompt. However, a similar command line input rule is used for reading the corresponding values for both the types (ex: KP?, PS?). Even the command line help is shown on the screen using the same read input command (HP?) value rules (refer to the flow chart in figure 3.8). According to this rule, no read command should exceed three bytes. This means that it should contain the name of the parameter (which is a two-character constant name, such as: PS or HP, etc.) followed by a question mark (?). In order to avoid any syntax error these rules for read commands are to be followed. Unlike the similar rules for entering the read commands, the output of any user request is shown, basically, in two different ways, based on a flag's value. If the flag is set, the read command value is shown as a floating point (values with a decimal point) and if this flag is clear, the input command's value is displayed as an integer as shown in 'Read Value Command Service Routine' flow chart in figure 3.8 and 3.9.

This module works the opposite way than that of the write command module. It first of all grabs the hex value from the permanent memory location of a parameter whose

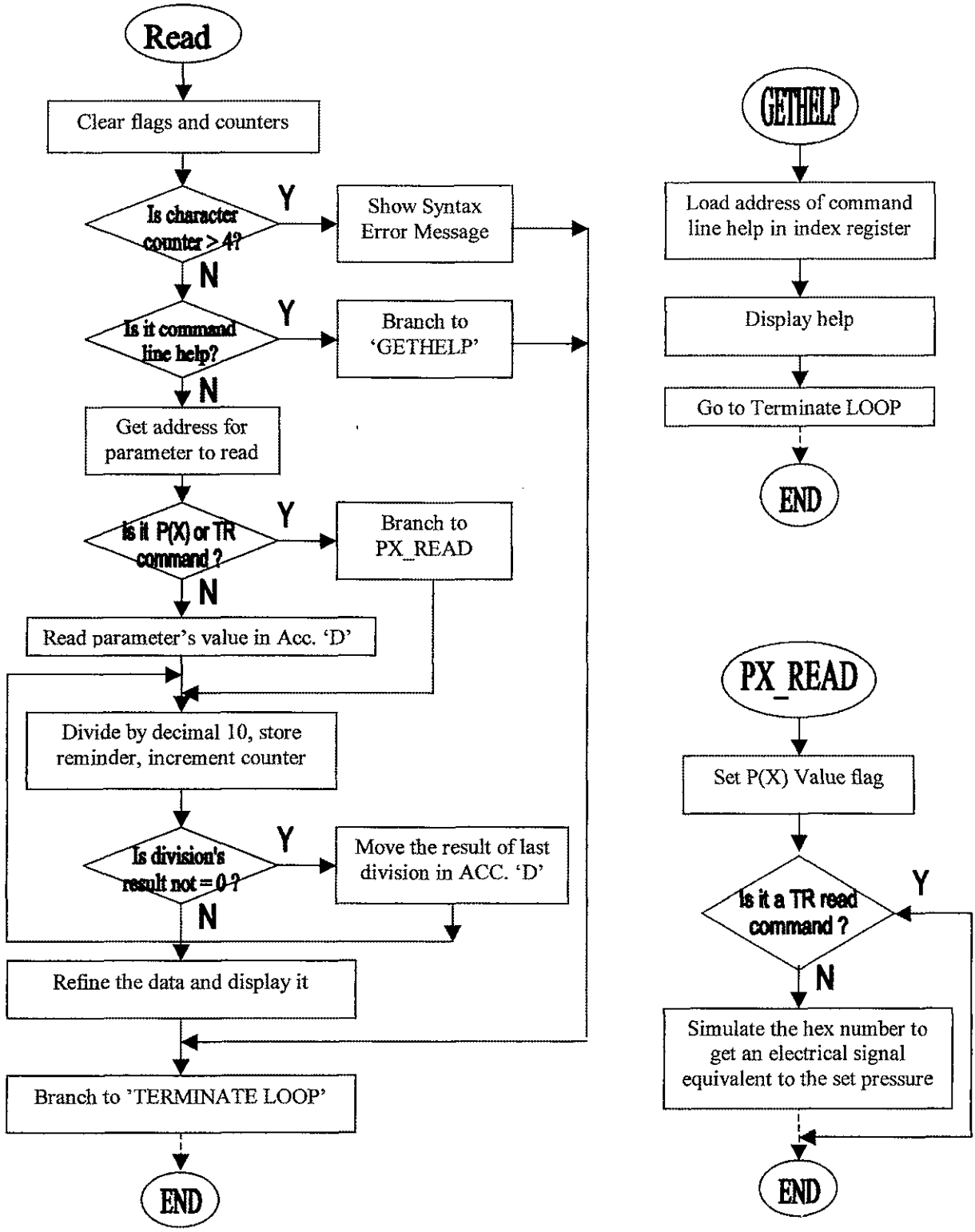


Figure 3.8 Read Value Command Service Routine

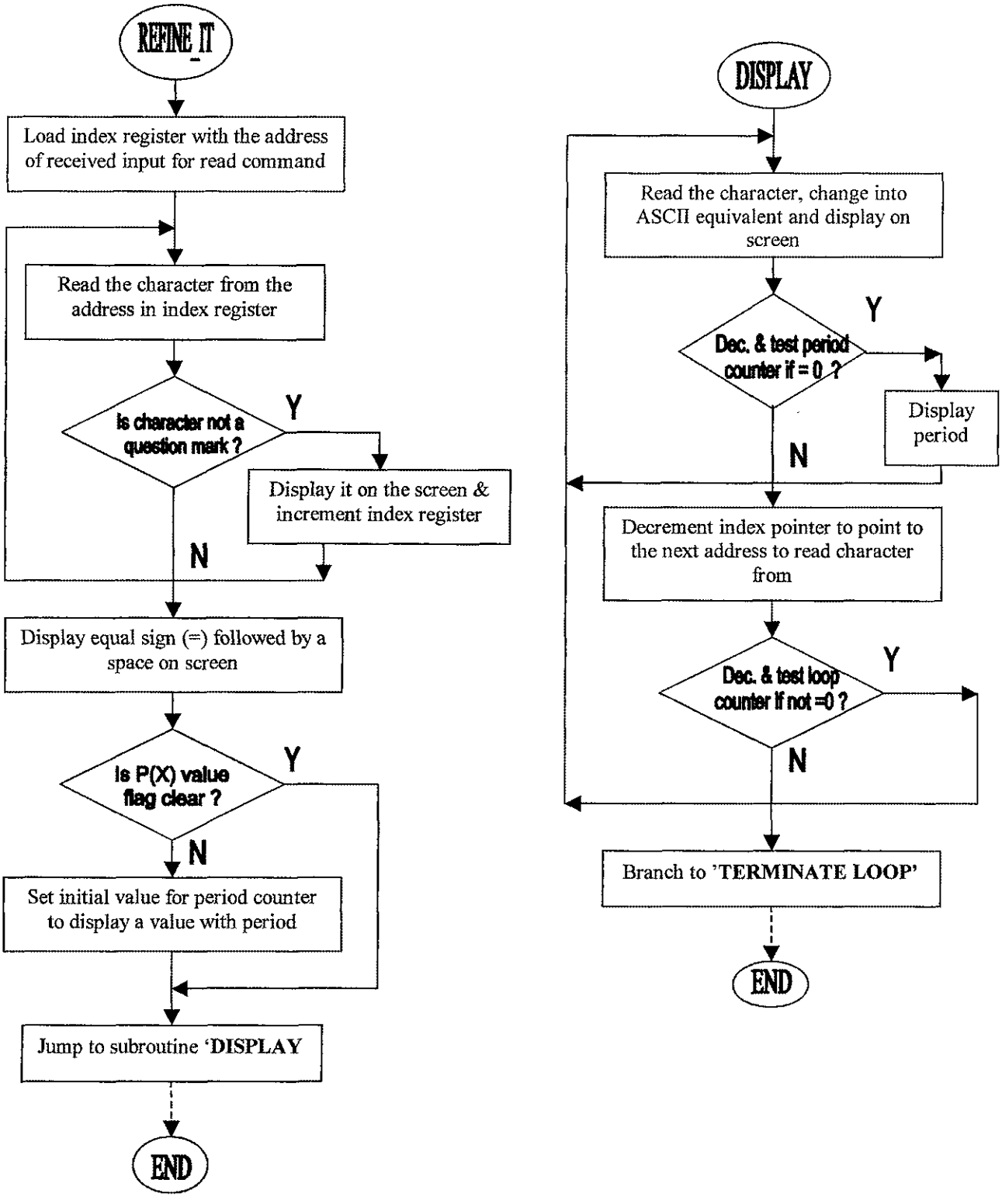


Figure 3.9 Refine and Display Service Routine

value is to be read. The permanent memory location is found by using the same reference -value technique as described in the previous section. This hex value is then translated to its decimal equivalent, and finally, to display this new value on to the terminal screen, it is further translated into their equivalent ASCII codes. Some temporary memory registers are used to hold the immediate equivalent results for the hex-to-decimal and decimal-to-ASCII number translation. To avoid any error manifestation, the system is made immune to any intermittent faults, such as: data error that could be caused due to software or hardware failures.

This functionality is added in the system by using a double error-checking mechanism that is checking the parameter-type and the decimal-point flags before showing the values on the display prompt. For example, if the floating-point-detect flag is set but the parameter is not a floating-point type, the system clears the flag and sends an error message signal on the monitor screen to let the operator know about this erroneous input data. For a command-line help, the system checks the passed on value for the parameter from the previous module. This reference value tells the system that a command-line help is needed to be displayed on the terminal screen. This module is based upon two basic sub modules to carry out most of the tasks. The first sub-module reads the value of a parameter and the other displays this value on the terminal screen. As mentioned above, the second sub-module checks the floating-point-detect flag to display a floating point or an integer value. Finally when this read command module displays a parameter's value on the terminal screen, a prompt is displayed on the very next line and control is transferred to the wait-forever loop (main programming loop).

When a read request is processed by the system, it uses the 'Refine-it' sub module to display it on the screen. For example, when a user wants to read the current value for the set point pressure parameter, he enters 'PS?' This data is always stored into the input capture buffer. When the read module completes translating the hex data from the PS parameter's permanent location in the RAM to an equivalent ASCII code, it echoes the parameter's names, saved in the input capture buffer, to the display monitor until it finds a question mark. As the next step the read module prints an equal sign (=) on the monitor

screen and goes to the location where ASCII codes for PS parameters are saved as shown in 'Read Value Command Service Routine' flow chart in figure 3.9. Now, if the hex number is equal to decimal 6535, the output would be: PS=653.5.

Write Commands:

This module is used to write values for different parameters. Like the read commands module, there are two kinds of parameter's values allowed to be entered on the command prompt. It could either be a floating point or an integer number. The same flag is used, as in the read command module, to distinguish between the integer and the floating- point values. This flag is set (flag value = binary 1) when a decimal point is entered with the value of a parameter and is cleared (flag value = binary 0) when an integer value is entered. This flag value is then passed on to the next module to make decisions for the next phase of servicing a user request. Since only decimal numbers (0 to 9) are allowed to be entered at the command prompt for write value commands, any hex value or a mistyped character causes a syntax error. Also, for both floating point and integer value write commands, the maximum number of typed characters should not be greater than 8-byte or smaller than 4-byte. A violation of this rule can also cause a syntax error as shown in 'Write Value Command Service Routine' flow chart in figure 3.10 and 3.11.

This module, first of all, checks the input data at the command prompt for any syntax errors. If the data using write command is correctly entered, it is then translated from ASCII to a decimal equivalent. A Boolean technique is used to change the data from ASCII to decimal number. For example, if a user types the number 4 at the command prompt, an ASCII value of 34 will be stored in the memory. Masking this ASCII value of 34 with a 0F as an 'AND' Boolean operation will result in a number 04 which is the same number as entered at the command prompt. All the numbers in the input data buffer register are then translated from ASCII to their decimal equivalent using the same technique until a carriage return (ASCII code 0D) is found. At this point the control is transferred to the next sub-program within this module to calculate a hex equivalent of this decimal value. To get the hex equivalent, each place (tenth, hundredth, etc,) of

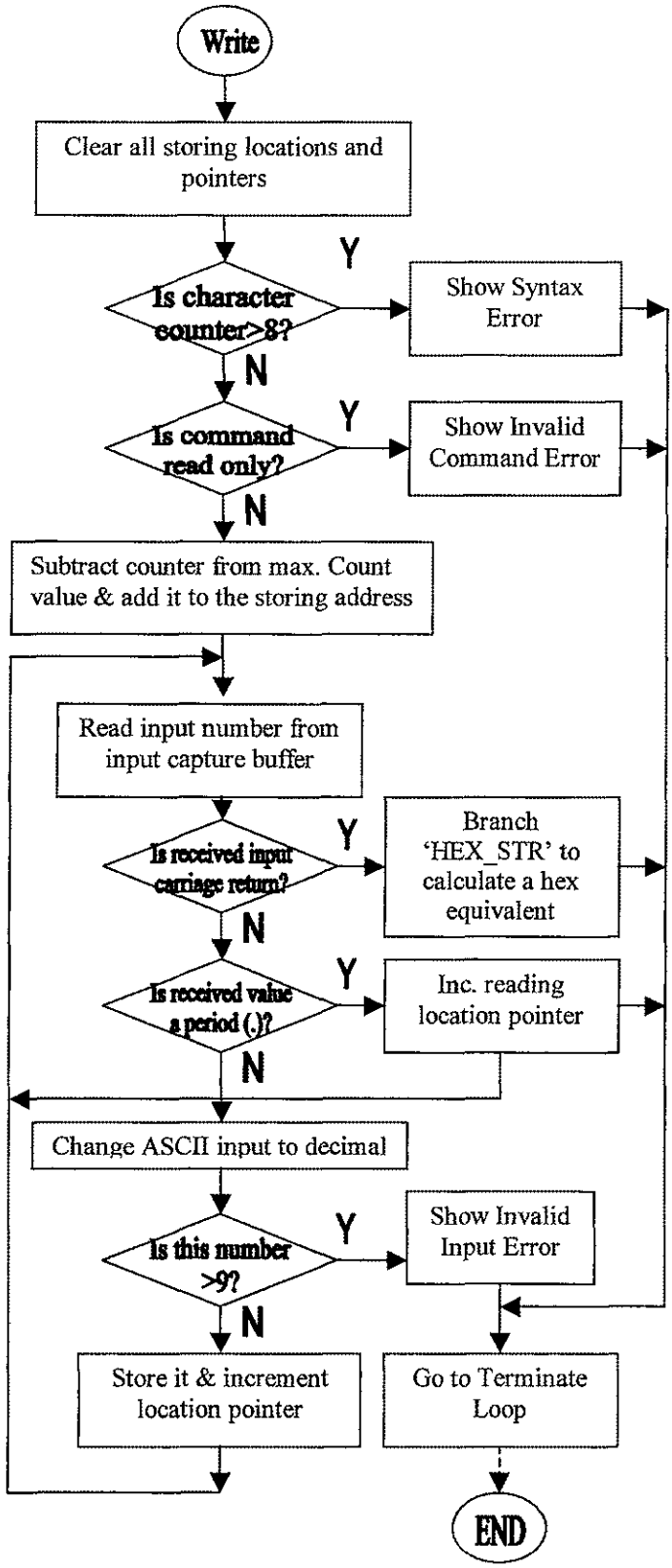


Figure 3.10 Write Value Command Service Routine

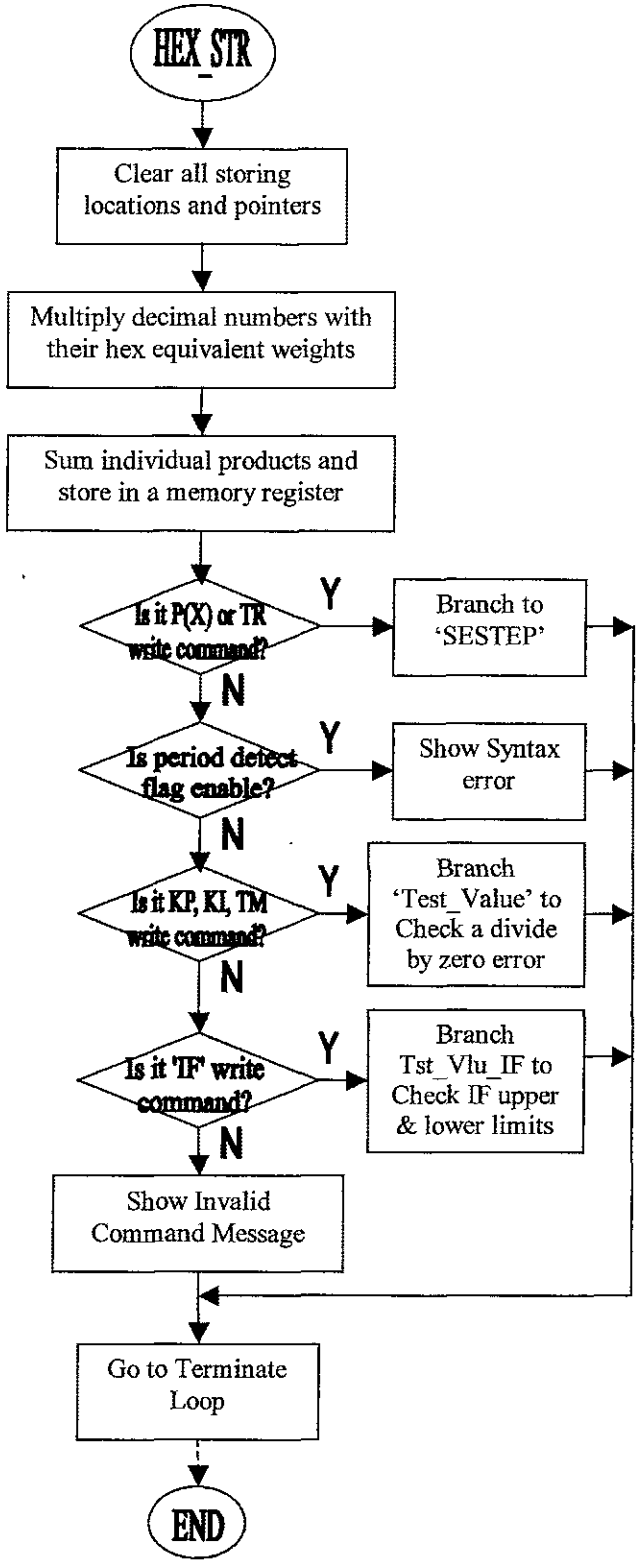


Figure 3.10 Write Value Command Service Routine

decimal number is multiplied to its corresponding hex equivalent number as shown in 'Write Value Command Service Routine' flow chart in figure 3.10 and 3.11.

For example, a 1234 decimal number is equal to 04D2 hex number. To obtain this number, multiply the first decimal number, '1', at the thousandth place by the corresponding hex number 03E8 (1000d = 03E8h). Multiply the second decimal number, '2', at the hundredth place by the corresponding hex number 64 (100d = 64h) and the third decimal number, '3', by 0A (10d = 0Ah). Finally, we can leave the fourth decimal number, '4', as it is, since multiplying it by the unit place hex number (1) will result in the same number. Using sum of the product technique, the equivalent hex number can easily be calculated. In the above example, the individual products are:

(1d) * (03E8h) = 03E8h,
(2d) * (64h) = C8h,
(3d) *(0Ah) = 1Eh and
(4d) * (1h) = 4h

Now, summing up the products will result in:

(03E8h) + (C8h) + (1Eh) + (4h) = 04D2h

This is the hex equivalent of the decimal number '1234'.

This translation of a decimal number into its hex equivalent is controlled by means of a down counter, whose value is known and fixed, regardless the number of data inputs for a decimal value. If, for example, a user wants to change a parameter's value to a decimal 10, then this module stores the actual number 10 as 0010 so that it can use the same techniques of multiplication and addition, as mentioned above, to get a hex equivalent number for the decimal number 10.

The parameters correspond to the pressure values in the system, such as: PS, PU, etc., translate a different simulated hex number to generate an electrical signal corresponding to an equivalent pressure value at the output. For example, if you enter

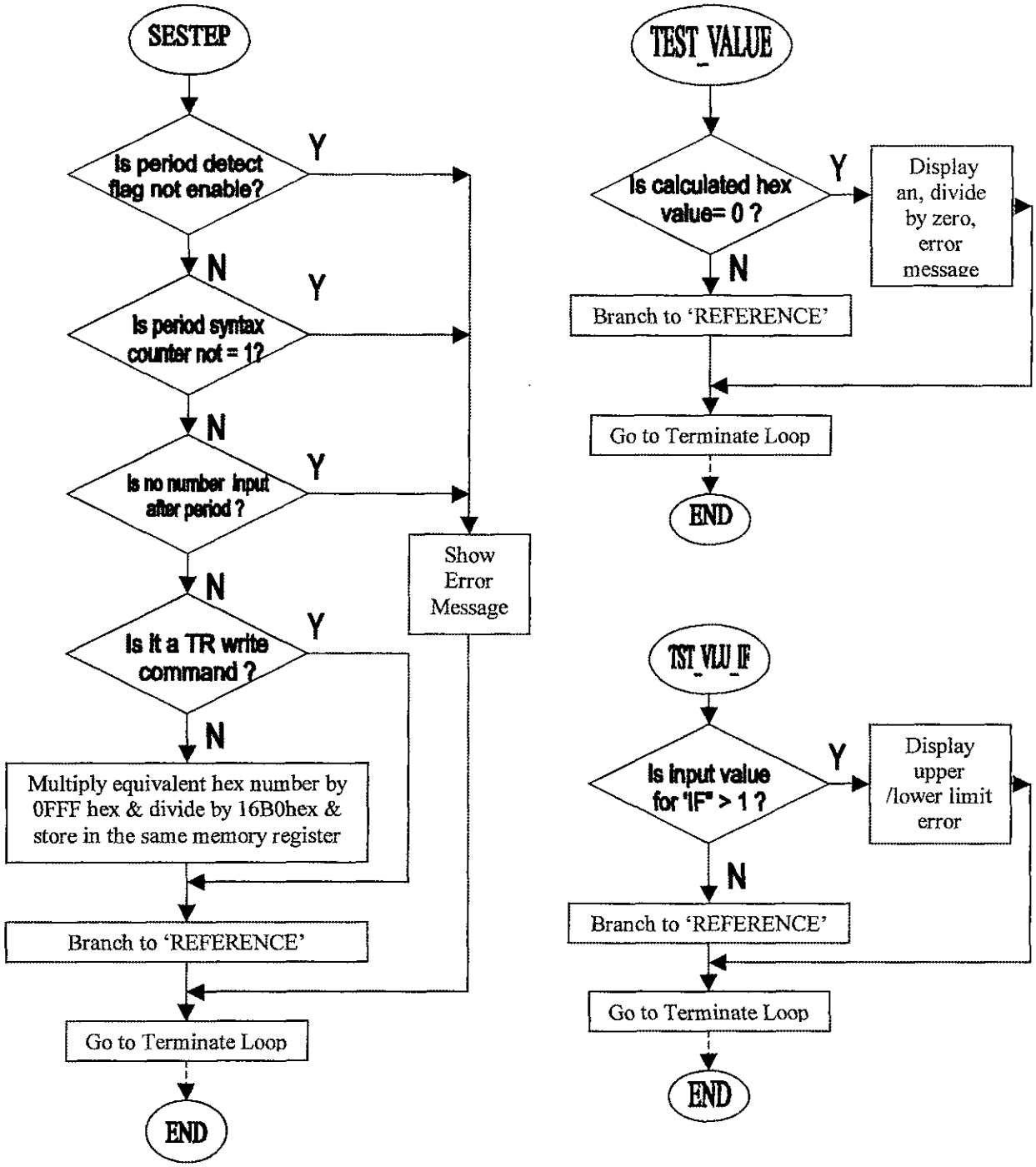


Figure 3.11 Second Step and Test Limits Service Routine

PS=50.0, it translates and store 32 hex, which is equivalent to a decimal 50. But the hex number needed to generate an equivalent 50.0 psi pressure at the output is 23 hex. Now to simulate 32 into 23, we use the formula: (hex number * 0FFFh) / (16B0h) as shown in 'Write Value Command Service Routine' flow chart in figure 3.11.

To keep the control system stable and running without difficulties, some of the parameters are checked for a divide by zero error. Since, some of the parameters must not contain a value equal to zero; the system checks the number first before permanently storing it in a memory location. This software module also checks the input arguments for an upper/lower limit violation.

Finally, when no errors are found in the input data of a parameter, the translated hex value is stored in its corresponding permanent memory location using its unique reference value as shown in 'Write Value Command Service Routine' flow chart in figure B5 (see appendix B for more details). Using "terminate-loop" as shown in the flow chart in Figure B5, the system transfers the control back to the main program loop.

Debug Commands:

This module is used to provide a command line debugging for the complete system. Two commands, Memory Modify and Memory Dump, are used to changing the RAM contents and displaying any memory location including ROM, EPROM and Flash EEPROM respectively. These commands are very useful for testing purposes and can be used any time during the normal execution of CONCOA software program. This feature is included in the system to prevent the excessive writing of the Flash EEPROM. This really helped us testing our program in the Flash EEPROM completely by putting different numbers for different parameters in the RAM. We also checked the outcomes of these parameters after modifying their previous values in the RAM. This software module turned out to be an important source of help to come up with a reasonable set of numbers for many parameters and also to finally write an error-free version for CONCOA software in the Flash EEPROM.

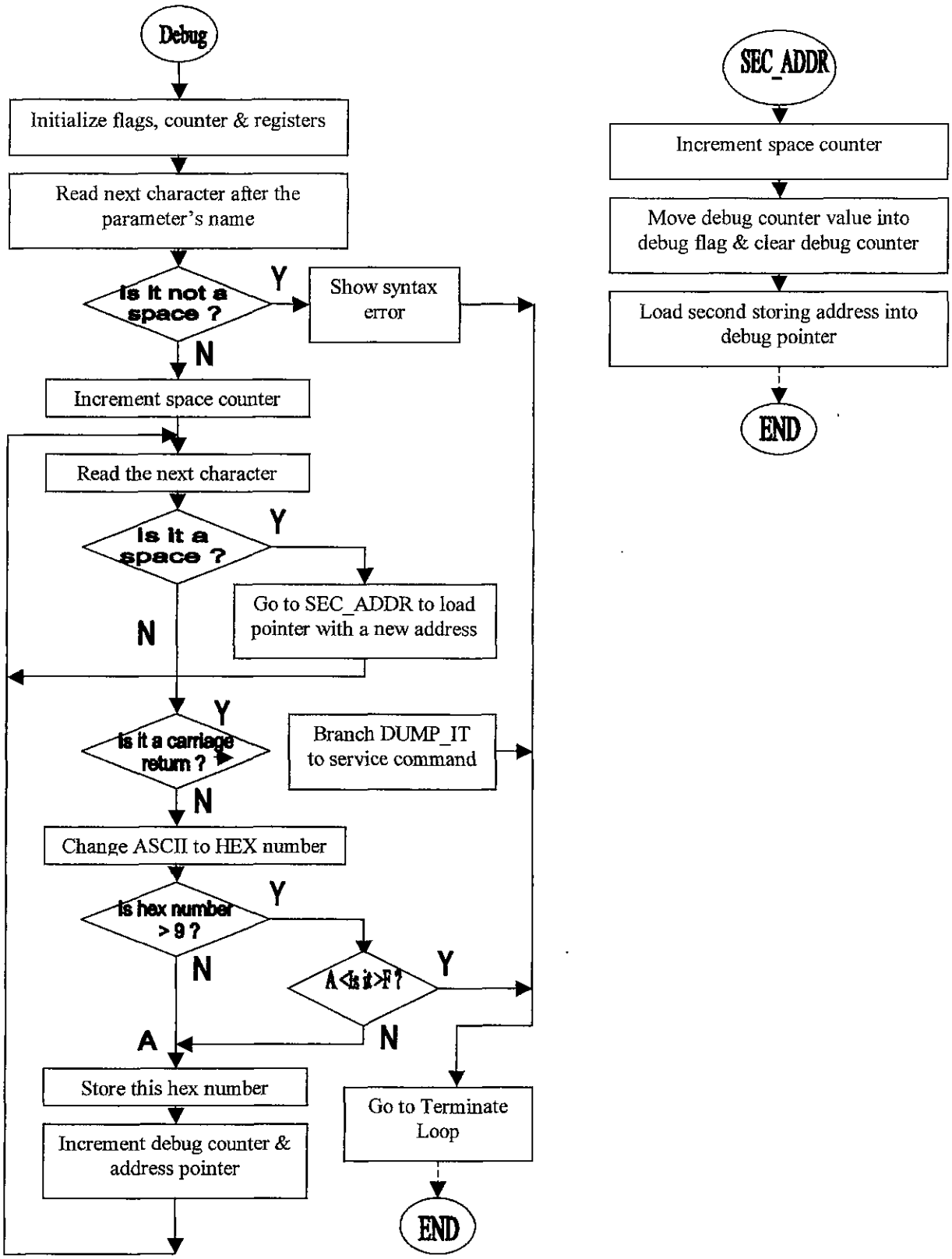


Figure 3.12 Debug Command Main Routine

The basic technique, for transferring control to a particular software module to carry out a user request, is the same as in the read or the write commands. However, these commands follow a different input format at the command prompt. When a debug command is entered at the CONCOA prompt, its reference value is passed to the next module, which recognizes the command type and transfers the control to that particular software module. This module starts by verifying the command line arguments against probable errors as shown in 'Debug Command Main Routine' flow chart, in figure 3.12. It then checks and converts the ASCII value of the memory addresses for the memory dump or memory modify commands to their hex equivalent number. Once a carriage return is found, the system transfers the control to the next phase of operation for a user request to either modify a RAM location or dump a memory block as shown in the 'Debug Command Subroutine' flow chart in figure 3.13.

The following two sub-modules are used to either dump or modify the selected memory locations.

- Memory Modify sub-module
- Memory Dump sub-module

Memory Modify Sub-module:

This sub-module of the main debug module, first of all, checks the address range of a memory location to modify. If the address is outside the 68HC12 microcontroller's RAM or it is some how mistyped, it gives an error message and immediately transfers the control to the 'wait-forever' main program loop. Since this address is stored in the memory as ASCII codes (e.g., for 0800 decimal number it will store 30383030 in the consecutive memory location in the RAM), the program will calculate an equivalent decimal number to compare if the actual number (0800 decimal), that user wanted to change the contents of, is within the RAM. To see how this address is calculated, refer to the software module 'Cal-Address' in 'Common Debug Command Routine' flow chart in figure B3 of appendix B). Now if this address is correct, the system prints this address

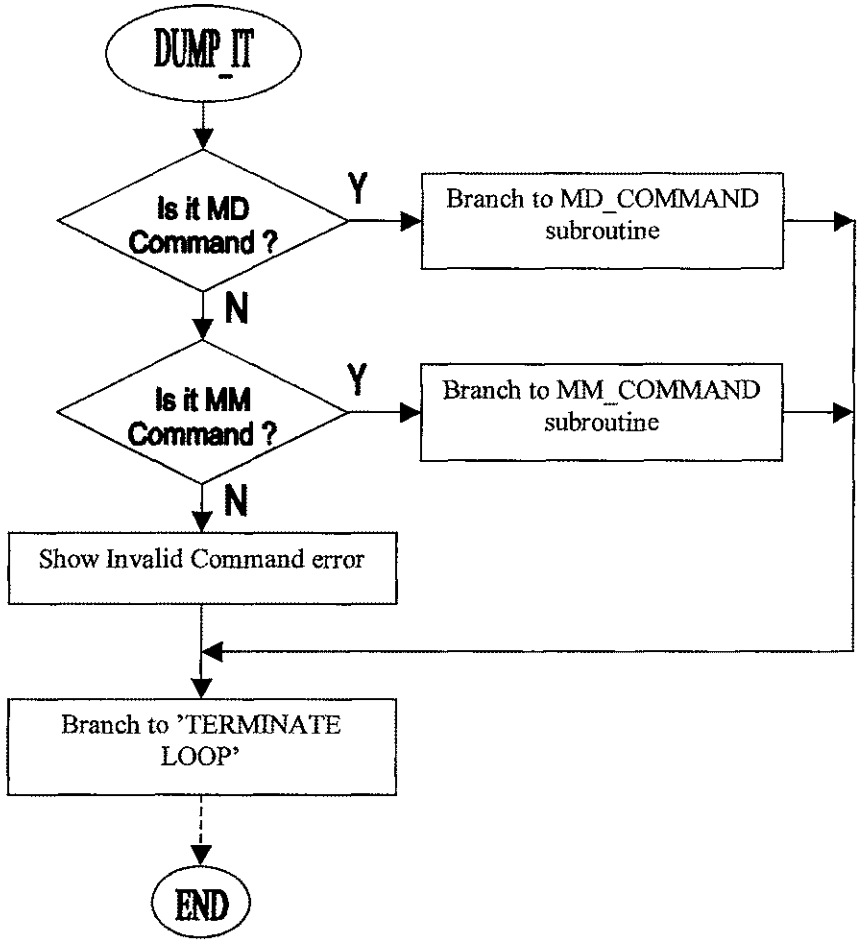


Figure 3.13 Debug Command Subroutine

and its contents (data), separated by spaces, on the very next line of the command line arguments (refer to software module 'First_Prnt' in 'Common Debug Command Routine' flow chart in figure B3 of appendix B). At this point the control undergoes into a user input-wait state and the system waits for a new value or data for this particular memory address. The backspace feature is also available for this command, so any mistyped new value can be easily modified before submitting it finally using the computer's "Enter" key as shown below in 'Memory Modify Command' flow chart, in figure 3.14.

Once a carriage return is detected, the control transfers to the next stage of operation, which is to check the syntax for the new value to modify. If there are no errors, the system converts the ASCII value of the input data to its hex equivalent number and stores it at that specific address which was entered at the command prompt. The system then repeats the same loop of displaying the address and waits for the user to input a new value to modify the next available location. To skip a memory location, one either needs to press carriage return or a plus "+" key and it will print on the next line the very next address, immediately after the skipped address and the system will wait for a new value to modify. In order to go back to a previous memory location a minus "-" key is used. When you finish modifying the memory locations, terminate this loop by entering a period (.) followed by a carriage return as shown in Chk-Option memory module in 'Memory Modify Command' flow chart, in figure 3.15.

In order to verify the memory locations just modified, the memory dump command can be used. Now once a period in used and the memory modify loop is terminated, the control always transfers back to the user input routine.

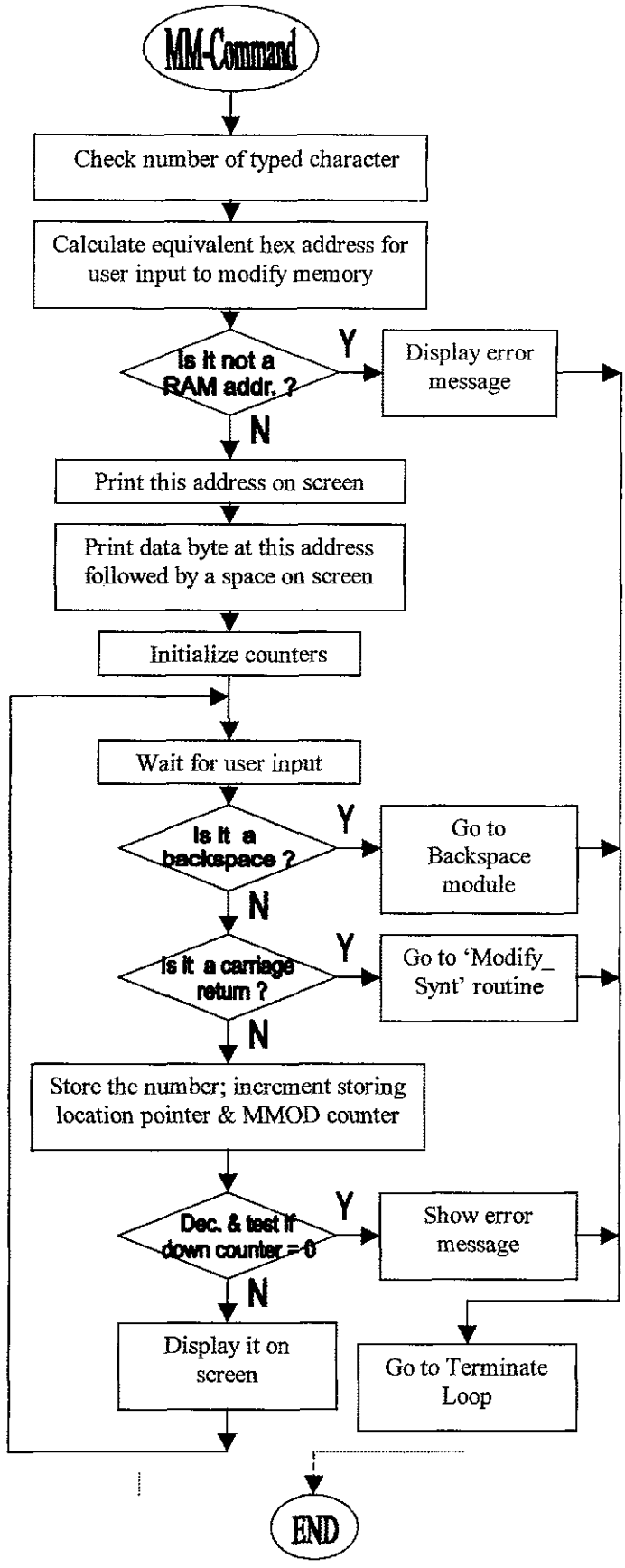


Figure 3.14 Memory Modify Command

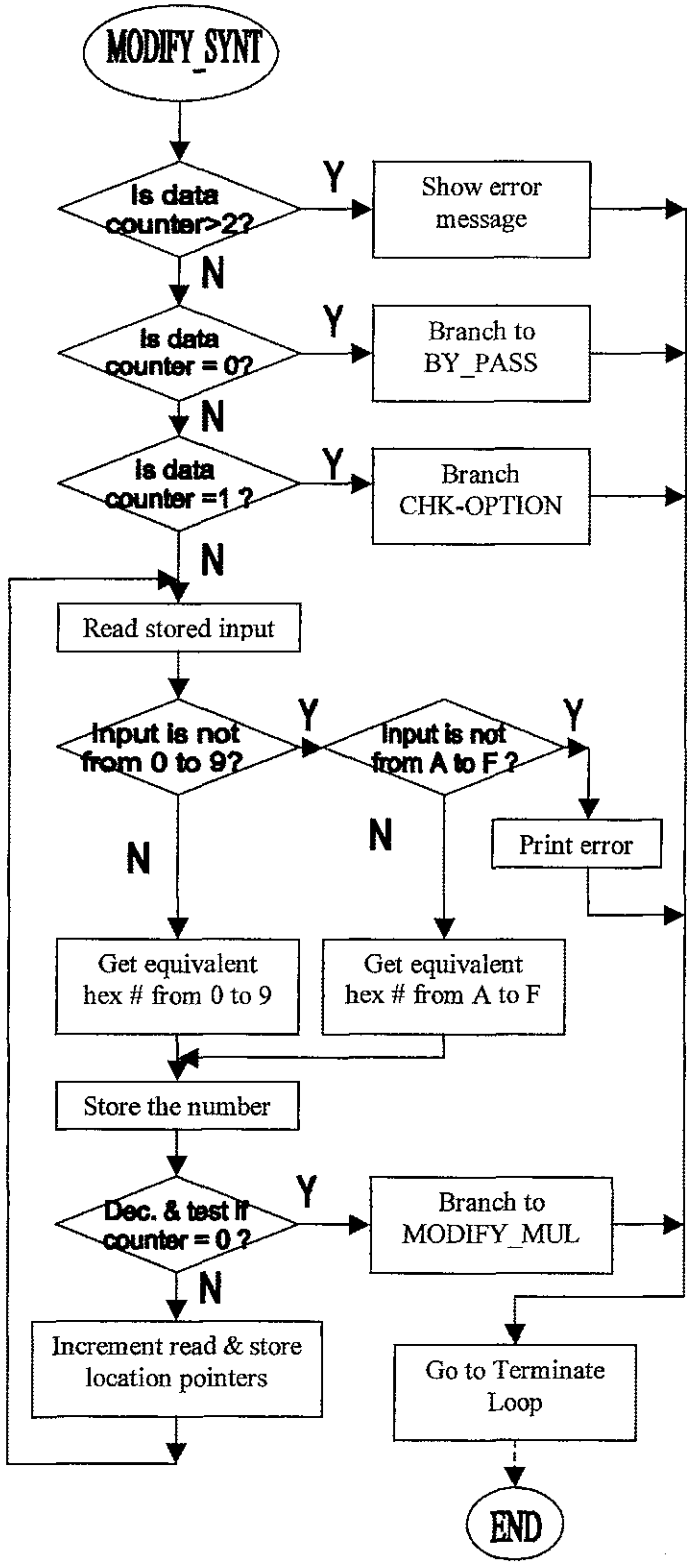
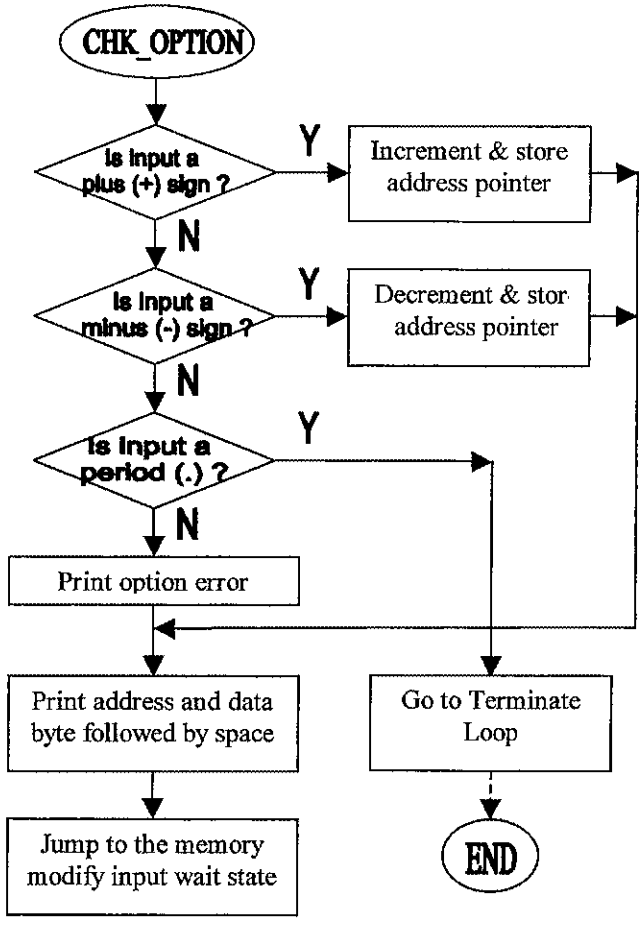


Figure 3.14 Memory Modify Command



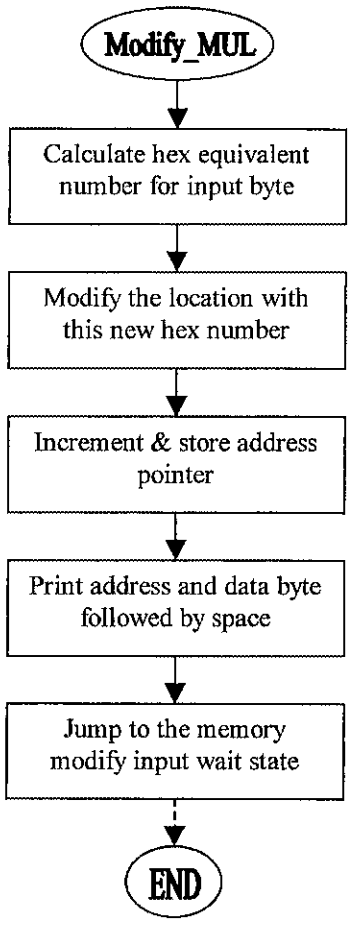


Figure 3.15 Check-Options Service Routine

Memory Dump Sub-module:

This command is used to dump a block of memory locations using a set of rules for entering start and the end addresses of the memory block needed to be dumped at the command prompt. This command dumps a minimum of sixteen bytes. So if it is desired to dump just one memory location, it will automatically dump the very next fifteen bytes after printing the actual (first) one. To see how this address is calculated, refer to the software module 'Cal-Address' in 'Common Debug Command Routine' flow chart in figure B3 of appendix B. The memory dump command prints a constant, sixteen bytes in a row. So if the number of memory bytes between the start and the end addresses is less than sixteen, it prints a complete block of sixteen bytes in that row up till it reaches the very next address which is a multiple of 2 ^ n. If the number of bytes is greater than sixteen, it prints the first sixteen in a row and prints a complete block of the next sixteen bytes on the very next line. Now, if the number of bytes are greater than 16 and less than or equal to 32, it stops printing after two lines (1 line = 16 memory bytes). This shows that it prints at least a sixteen-byte row and so forth depending upon the start and the end address for the memory dump as shown in 'Memory Dump Command' flow chart, in figure 3.16.

Basically, the screen for this command is divided into three columns. The first or the left column shows the address of byte zero of a sixteen-byte block, the second or the middle column contains the contents of these sixteen bytes, starting with byte zero whose address is printed on the left column, the third or the right column indicates the ASCII representation of the equivalent hex number on the middle column. To know how these characters are printed in the 1st 2nd and 3rd columns, refer to the software modules 'Seprt-Nibble' and 'Prnt-Char' in 'Common Debug Command Routine' flow chart in figure B4 of appendix B). For example, if you see a hex number 41 in the middle column, it will show the letter 'A' in the right column, which is nothing but the ASCII code of 41 hex. The ASCII code display in the right column is limited to displaying the alphabets, and some other special characters. For the characters like carriage return (ASCII code 0d) it

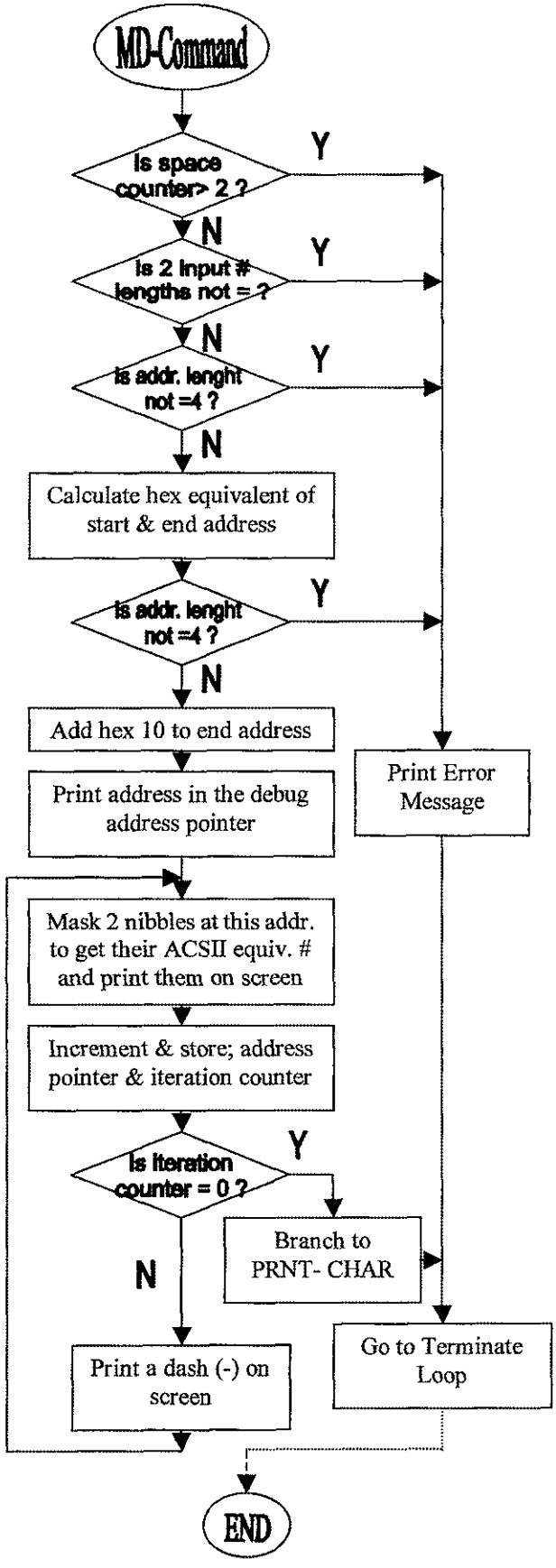


Figure 3.16 Memory Dump Command

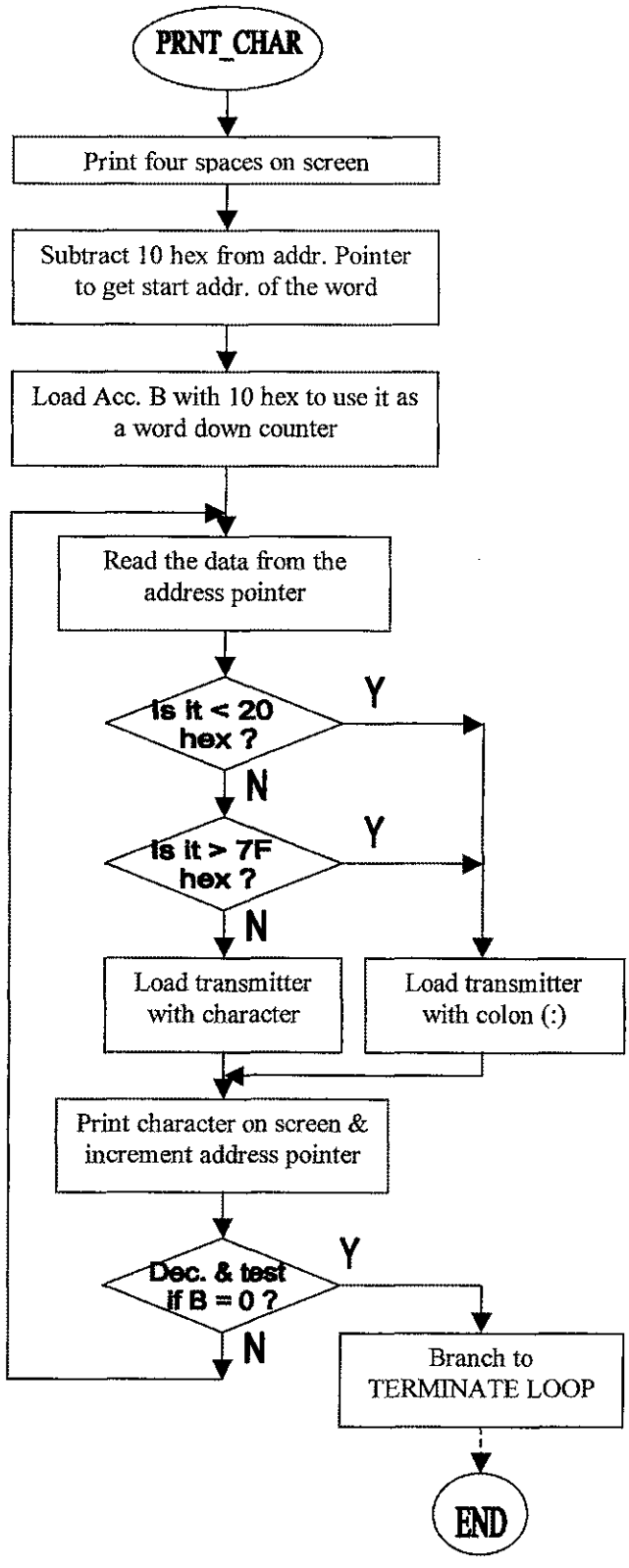


Figure 3.16 Memory Dump Command

just prints a colon (:) to prevent any monitor screen disturbances as shown in the 'Prnt-Char' software module in 'Memory Dump Command' flow chart, in figure 3.16.

Unlike the memory modify command where the request loop can only be terminated by entering a period (.) followed by a carriage return, this module transfers the control back to wait for user input module after displaying the memory contents needed to be dumped. However, like the memory modify module, it also checks the command line arguments for any possible human error.

Once the system finds no errors, it starts by changing the ASCII codes of the addresses to their hex equivalent numbers and stores the starting and the ending memory addresses to two different memory locations. It then prints the start address of the memory byte to be dumped followed by the contents of the next fifteen bytes and the start byte. The next task of this command is to display an ASCII map of the hex equivalent numbers just printed in the middle column. Once it finishes printing the first row, it prints a carriage return and line feed and continues the same operation on the next line and so forth, until the address pointer reaches the end address for the memory dump. Even if the end address is in the middle of a particular row, the system ends up printing until the complete set of 16-byte in that row. Every new row starts with the very next address of the memory after the last address on the previous row.

3.3 Software Integration:

The integration of the modules, mentioned above, can be described as follows:

- Monitor and Debug program integration
- Complete Software integration

3.3.1 Monitor and Debug Program Integration:

All the individual modules of the Monitor and the Debug programs are tested individually before their complete integration. None of the main modules are directly available to carry out a particular user request. Even though the main modules work independently, they all are linked to another module, common to them. This module provides the point of integration for the main modules, such as: write, read and debug, etc. Also, they have another common module to exit the main modules, which provide a common point of integration to exit the individual programs. This exit-link module is used to initialize all the necessary flags, counters and registers, before transferring control back to the main user request loop. The common starting link module, used by all the other modules to take control of the program for a corresponding user request, uses a parameter passing technique to the very next module to identify the command entered at the prompt.

For the memory modify command, a separate user wait-loop is integrated into the debug module, which works independently from the main user-wait loop. The complete monitor and debug programs are nicely integrated and are checked for any possible errors.

3.3.2 Complete Software Integration:

The complete software integration comprises of the Monitor, Debug and the Control programs. On reset the system starts loading monitor program and displays a prompt (<CONCOA>) on the screen. At this point the only program available to the system operator is the monitor and the debug program with all its functionalities, i.e., with all its debugging and monitoring features available. Now, to enable the control program the operator needs to use the DN (for done) command followed by a carriage return. This command actually sets a timer interrupt to occur after every TM milliseconds. This procedure is adopted because the whole control program is integrated

into the interrupt timer service routine. This shows that after every TM ms the control of the system is transferred to the control program to do the analog to digital conversions of PO and PB, perform the PID control calculations to finally calculate a new 16-bit digital word for the solid-state gas-regulator. It transfers this digital voltage word to the solid-state regulator by operating and transferring the 12-bit word to the DAC, whose output is attached to the input of the SSR. The control of the system is transferred from the control program back to the main program every time, when an interrupt service routine is completed. The main program then resumes the task it was doing before the timer interrupt had interrupted the system.

The control program always works in the background and the monitor program in the foreground. Since, the two programs are very nicely integrated and the transfer of control between them is very frequent, it seems for the user as just one program carrying out all the tasks without any internal or external interruptions. Once the control program is running in the background, the debug commands can be used to debug it for any possible errors.

CHAPTER 4

SOFTWARE RESULTS

4.1 Program Testing:

The software development for this project can be divided into the monitor, debug and the control programs, which are well integrated to provide complete software with its controlling, monitoring and debugging features available. To see how the main and the control programs are integrated refer to fig. 3.1 in chapter 3. This thesis only covers the code written for the monitor and the debug programs, which enables the system operators to write and read the system set and system response parameters (for more details refer to section number 4.2 of software command set). The debug program enables a user to modify any RAM locations and print the complete memory map including RAM, ROM, EEPROM and Flash-EEPROM. To use these features effectively a user interface with a software command set is developed using modular programming techniques. These modules are further divided into smaller modules and sub-modules. The testing of the main program, which includes the monitor and the debug programs, can be described as follows:

- Individual software module testing
- Monitor program testing
- Complete main program testing

4.1.1 Individual Software Module Testing:

This kind of testing is done mostly during the code-write up on a regular basis. Since most of the modules were smaller in size, the testing is done in the RAM of

68HC12 evaluation board using factory installed monitor/debugger program 'SDBUG12' in the Flash-EEPROM. The testing in the RAM is done using the SDBUG12 commands. The bigger modules are tested using the same default program, but by dividing them into smaller modules and testing each on an individual basis. Since, a modular programming technique is used to write the software for CONCOA's gas regulator system, individual modules are tested during and after their completion. The three very basic modules of the fifth main module 'Service User Request' that is read, write and debug modules as shown in the 'Basic Program Flow Chart' in figure 3.1, are tested thoroughly on an individual basis before their integration as the main program. There were some other smaller modules and sub-modules, which were also tested with their parent modules to verify the strong integration as a single main module.

The individual testing of the modules can be described as follows:

- Read module testing
- Write module testing
- Debug module testing
- Other modules testing

4.1.2 Read Module Testing:

As mentioned previously that the basic modules are not available directly and can only be accessed through a common module, linked them and the wait for user request modules as shown in 'Syntax check and Service User Request' flowchart in figure 3.6 in chapter 3. This module is tested with some very typical error scenarios for an end user software operator. To provide an interactive command line communication, different warning and alert messages are written for different situations. For example, to read a parameter's value, a question mark must follow the parameter's name. Also the commands should always be entered in uppercase letters at the CONCOA prompt. A syntax error will be displayed if these rules for a read command input are not followed.

The command line help is also a part of the read module and follow the same rules, mentioned above.

The syntax for command line help is the same as that of the other read commands (i.e. 3 bytes- 2-byte for parameters name and 1-byte for question mark). Any violation of these set rules for a read command will cause a syntax error as shown in section 4.3.1. This module was tested during and after its code was written and no errors have been found so far.

4.1.3 Write Module Testing:

This is the most important module as it is used to write the parameters' values, which directly or in directly affect the output voltage for the CONCOA's gas-regulator system. Different system messages are displayed for different kinds of error manifestations. Since, this module is used to modify the most basic PID control coefficients, such as: KP and KI, a new value has to be tested thoroughly before the system actually changes and stores them at a permanent memory location. Otherwise, the controlled gas regulator control system could become unstable and potentially cause damage and/or a dangerous situation.

The value for some parameters has to be checked for a divide by zero error every the time before finally modifying and storing their new values. For example, the P coefficient 'KP', of the PID coefficients, is used as a denominator in the mathematical formulas for calculating a new output pressure value. Thus, a zero for KP can cause a serious problem for the whole system and it can get unstable. Now, if someone intentionally or unintentionally tries to change it to a zero, the system will display an error message leaving the value for KD unchanged.

Since, different parameters are used to carryout different calculations in the control program; certain limits are imposed for their proper modifications. For example it was noticed during the onsite testing of the software program that a TR (Ramp time)

value greater than sixty (60 seconds) caused the system to become unstable, so a maximum value limit of 60 is set for TR. Some parameters, like IF (work as a flag), can only have two values, either a zero (0) or a one (1). Any other value for this parameter is restricted by the system with a corresponding error message as shown in section 4.3.X. The default parameter values for different parameters are shown in table 3.1 in section 3.2.2 in chapter 3.

The write module allows two kinds of parameters' values, such as: floating point and integer. This module has been tested for all pressure write commands, such as: PS, PU, PL, etc, which are floating point type and should always input with a decimal point as shown in section number 4.2.3.

The system displays different messages for different types of errors or typing mistakes at the command prompt. Since, only decimal numbers should be entered at the command prompt, any other value, such as: a hex number, is not allowed and causes an error message to display on the terminal screen. Since floating point and integer number parameters have different input arguments length, the system also tests for any argument length violation and a corresponding error message is displayed for an error (refer to command set description in section 4.2.3. Finally the complete write module is tested for any possible errors and it works fine without any problems.

4.1.4 Debug Module Testing:

The rules for entering command line arguments for this module is different than read and write software modules. Since, the memory modify command is used to modify only the RAM locations, this module is tested for a typical user error of modifying a memory location outside the RAM area. The program responds with an error message of violating the modifying range. All the functions of memory modify command, such as: loop terminator (decimal point), next address pointer (+ sign), and previous address pointer (- sign), are tested and work excellent for all the possible user errors.

As with the memory-modify command, certain rules should always be followed to entering a memory dump command. For example, the start and end addresses must always contain one space between them and they should always be four-byte long. The system is tested to show a corresponding error message for a typical mistake violating any one of the above-mentioned rules for entering command line arguments. The memory-dump command is also tested to display a desired output on the computer monitor screen. The display is parted into three columns for a complete dump (display) of a particular memory block on the monitor screen, such as: the first columns show the address of the first byte for a sixteen-byte row, the second columns show the actual hex contents for the sixteen bytes in a particular row, and the third columns show the ASCII numbers corresponding to the hex values of the sixteen bytes on the second column.

The other smaller modules, for example: modules for separating and storing command line arguments, converting ASCII to hex number and vice versa, user wait loop for memory modify, memory contents display module, etc., are all individually tested for any possible errors during the code write up and after their integration into a single memory dump module.

4.1.5 Other Modules Testing:

The other important individual sub-programs are also tested on an individual basis. These modules includes the backspace, control program enable, wait-forever user input, common-link and exit-link and command-line syntax check modules are discussed below.

Back Space Module:

The backspace feature, which is used to erase the mistyped characters, was also tested and worked without a problem before and after the complete integration. This feature not only deletes mistyped characters but also refreshes the display terminal to make it user friendly. It is also tested to see if it is overwriting the mistyped character (s),

decrementing the character counter by one, and clearing the decimal-detect flag if the mistyped character is a decimal point.

Other Smaller Modules:

The module that enables the control program in the background is also tested for a possible error. In order to avoid any timer interrupt initialization problems, the system checks if a user tries to enable an already enabled control program and displays a warning message discarding the user's request to enable an already enabled control program as shown in 'Control Program Enable Request Routine' flow chart in figureB1 of appendix B.

There is another small but very important sub-module, wait for user input, is also tested with different error scenarios. The system is also tested to see if the program displays an error message and issues a command prompt on the very next line, when a user exceeds the buffers of the input capture register. This module also works perfect without any problems. To see how this routine works, refer to the flow chart in figure 3.5 of chapter number 3.

The two link modules, such as: common-link module (refer to 'Syntax Check & Service User Request' flow chart in figure 3.6 of chapter number 3) and the exit-link module (refer to 'Back' flow chart in figure B5 of appendix B) were tested for any errors during the program integration. These modules work without errors and provide a very strong point of integration for the most important main software modules to carryout user requests.

Some initial stage sub programs, for example, the module to check the command line syntax (refer to 'Syntax Check & pass parameter' flow chart in figure 3.7 of chapter number 3), are also tested with very typical end user error scenarios. All the error messages are also tested for a screen popup for a known user input error and they all performed excellently with their corresponding set of rules.

4.2 Software Parameter Set:

Since the command set and CONCOA prompt form the basis of the monitor program, it is essential to discuss them in this thesis.

The I/O parameter set for CONCOA controller can be divided mainly into three categories, such as:

- Read/Write Parameter
- Read-only Parameter
- Write-only Parameter

Below is a discussion of this command set, their name, correct syntax and the features they offer to accomplish various user requests.

4.2.1 Read / Write Parameters:

These commands are comprised of system-set parameters and system-response parameters. All those commands that can be read and write at any time are in this category. The memory modify (debug command) is also a part of read / write command, since it not only prints the memory contents of a parameter but also let the user to write a new value for it, if needed.

The following is a list of commands that can be used to read the current and write a new parameter's value as shown in table 4.2.

Read / Write Parameter Set			
Descriptions	**Unit**	**Name**	**Features**
Set Pressure	Psi	PS	Read/Write
Maximum Output Pressure	Psi	PU	Read/Write
Minimum Output Pressure	Psi	PL	Read/Write
P Coefficient of PID		KP	Read/Write
I Coefficient of PID		KI	Read/Write
Loop Time	ms	TM	Read/Write
Ramp Time for Soft Startup	s	TR	Read/Write
Flag for Set Pressure from Eva. Board		IF	Read/Write
SUMIN_DIV input		DS	Read/Write
Memory Modify		MM	Debug /Read/Write

Table 4.1 Read / Write Parameter Set

The 'IF' command is used to set the evaluation board mode so that the system can be operated to get a set-point pressure value either from the computer keyboard or from the evaluation board. If the 'IF' value is set to 0 (default setting) the system will read the set-point pressure from the keyboard and if it is 1, the input to the system will be provided by the evaluation board and the system will read from the memory location set aside for the set point pressure from the evaluation board. This pressure value from the evaluation board can be read using the 'PB' command.

The PS command is used to enter a set-point pressure for the output of CONCOA gas- regulator control system. The PU and PL commands are the upper and lower limits for the set point pressure. The KP and KI commands are used for entering a constant coefficient value for the proportional and the integral control of the system respectively.

As discussed in section 1.3 of chapter 1, TM (loop time) is the time-interval for the timer interrupt to occur after every TM ms, and TR (ramp time) command sets the

number of loop-time the control program will take to change the set point pressure (soft-start-up) at the output.

4.2.2Read-only Parameters:

The following commands are read only or system-response commands as they can only response to our request with the most current or desired output on the terminal screen. For example, the output pressure 'PO', which is read only, can only update us about the current value of output pressure. The 'CD' coefficient display command is used to print the current values for all the parameters whose default values are shown at the time of program initialization. The debug command 'MD' is also a read only command, since it displays only the memory block whose contents are needed to be dumped by the user.

The 'IT' command is used to display the status of the timer. If the value of this parameter is binary 1, it indicated that the timer is enabled to interrupt the main program after every 'TM' ms. It also shows that the control program is enabled as the control program is written in the interrupt service routine. A binary value of 0 in the 'IT' register indicates that the control program is not enabled yet. As discussed above, the 'PB' command is used to read the set-point pressure from the evaluation board.

The following is a list of commands that are read only as shown below in table 4.3.

Read-Only Parameter Set			
Descriptions	**Unit**	**Name**	**Features**
Output Pressure	psi	PO	Read only
Set Pressure from Eva. Board	psi	PB	Read only
Command Line Software Help		HP	Read only
Check Timer Status		IT	Read only
Coefficient Display		CD	Read only
Memory Dump		MD	Debug /Read/Write

Table 4.2 Read-Only Parameter Set

4.2.3 Write-only Commands:

There is only one command that falls in this category, as shown below. The DN command (abbreviated for done) is used to enable the control program. As described in appendix B(refer to flow chart in figure B1 to get more information about how to enable the timer interrupt) this command 'DN' (case sensitive= use only upper-case characters)is necessary to enter so that the system start sending and receiving the electrical signals from CONCOA gas regulator system using control program. This command does nothing but set a timer interrupt to occur after every TM ms. Since, the control program is located in the timer service routine, it starts executing after every TM (the default is 8) ms and returns the control back to the main program to resume its job from where (the memory location) it was interrupted.

Write-Only Parameter Set		
Descriptions	**Name**	**Features**
Enable Control Program	DN	Write

Table 4.4 Write-Only Parameter Set

4.2.4 Syntax Rules for Software Parameter Set:

All the command set parameters are *case sensitive* and should be entered *as upper case characters* for a read or a write value operation.

The system-set and system-response parameters are further divided, in terms of their values being written or read, in three parts, as shown below:

i. Commands, which contain a decimal point for both reading and writing values to permanent memory locations.
ii. Commands, which does not contain a decimal point (integer numbers) for both reading and writing values to permanent memory locations.
iii. Control program enable, command line help and debug commands.

i) Parameters with a decimal point (floating-point):

The following parameters' values must always contain a decimal point (•) between the integers for either read or a write value operation:

- PS
- PB
- PO

- PU
- PL
- TR
- CD

NOTE:

The maximum number of integers for a write operation cannot be greater than 4 and the minimum is 2. There should always be only one integer after the period or decimal point.

The 'CD' or coefficient display command, which is read only, displays the current values for all the parameters. That is why it is a part of both numbers (i) and (ii).

Example #1:

To write set pressure (PS) from the PC's keyboard for a value of 50 psi, the following command should be used in the exact order at the CONCOA prompt, as shown below:

<CONCOA> PS=50.0
<CONCOA>

One has to press the '*Enter*' key on the PC keyboard in order to write this value in the permanent storage location for the above parameter (PS). The CONCOA prompt will appear immediately on the next line when the system finishes serving the user's request.

Example #2:

To read set pressure (PS) from its permanent memory location the following command should be entered in the exact order at the CONCOA Prompt:

<CONCOA> PS?
PS= 50.0
<CONCOA>

NOTE:

The read value command is the same for all the other command set parameters that could have a different set of rules for the write value command.

ii) Parameters without a decimal point (integer):

The following parameters' values should only contain integers for a read or a write value operation:

- KP
- KI
- TM
- DS
- IF
- IT
- CD

NOTE:

The maximum number of integers for a write operation cannot be greater than 4 and the minimum is 1. Some of these parameters have a minimum and a maximum value limit that should always be taken into account while writing a value to a parameter.

The 'CD' or coefficient display command, which is read only, displays the current values for all the parameters. That is why it is a part of both numbers (i) and (ii).

Example #1:

To write a value of 100 for coefficient 'P' (KP) using PC keyboard, the following command should be used in the exact order at the CONCOA prompt, as shown below:

<CONCOA> KP=100
<CONCOA>

One has to press the "*Enter*" key on the PC keyboard in order to write this value in the permanent storage location for the above parameter (KP). The CONCOA prompt will appear immediately on the next line when system finishes up serving user request.

Example #2:

To read the same parameter (KP) from its permanent memory location the following command should be entered in the exact order at the CONCOA Prompt:

<CONCOA> KP?
KP= 100
<CONCOA>

NOTE:

The read value command is the same for all the other command set parameters that could have a different set of rules for the write value command.

iii) Control Program enables, Help and Debug command set:

The following commands are different than the rest of the command set.

- HP
- DN
- MD
- MM

For example the command-line help is neither a system-set or response parameter but it is only used to display an already stored help file in the Flash-EEPROM.

The DN command just enables the control program in the timer interrupt service routine by enabling the timer interrupt to occur after every 8 ms.

The debug commands are used to either write a new value for a parameter in the RAM or to dump the memory blocks including RAM, ROM, EPROM and Flash-EEPROM. They follow a different set of rules for the syntax at CONCOA prompt than the read and write commands that would be explained below in the following examples.

NOTE:

The debug commands are recommended only to the advance users of CONCOA software.

Example #1:

To get command line help (HP), the following syntax rules should be obeyed.

<CONCOA> HP? Followed by a carriage return

It will display a one page command line help followed by a CONCOCA prompt, as shown below.

<CONCOA>

Example #2:

To see the examples for memory modify (MM) and memory dump (MD) commands refer to 'Command Set and System Messages Examples', section A1 of appendix A.

4.3 System Messages:

To make an interactive and friendly user interface for this project, system messages are written. Basically, they can be divided into two categories: such as,

- Error Messages
- Alert messages

The error messages, such as: syntax error, length error, number error, limit error, etc. (as shown below in this section), are shown up when the system finds an error due to the invalid command line arguments or an internal error. Different messages are popped up for different kind of error manifestation. These error messages also contain an immediate help to remedy the mistake as shown below in this section.

The alert messages let a user know about the current status of the system. For example, when you use the 'DN' command to enable the Control program, the system prints a message after enabling the timer that the control program is enabled. If for some reason a user has forgotten whether he has enabled the Control program or not and he presses 'DN' again, his request will be discarded and the system will display a message letting the user know that the control program is already enabled. Similarly when the

system initializes after hardware reset, it prints the default value of all parameters and ask the user to either change or go with the default parameters' values.

Following are the system error messages for the CONCOA software:

- Syntax Error
- Value Error
- Divide by Zero Error
- Max / Min Limit Error
- Argument Error
- Length Error
- Other Error Messages

4.3.1 Syntax Error:

Example:

<CONCOA>ps=50.0

Syntax Error

To get command line help type: HP?

Since, the 'P' and 'S' characters are entered as lower case. In order to avoid this error type all characters in upper case.

4.3.2 Value Error:

Example:

<CONCOA>PS=12A.0

Invalid Parameter Value

You can only input the decimal numbers (0 to 9) for this command.

Since, A is not a decimal number. To avoid this error only use integer combinations from 0 to 9 to write a parameter value.

4.3.3 Divide by Zero Error:

Example:

<CONCOA>TM=0

Divide by Zero Error

The input value for this parameter should be greater than zero (0).

Since, TM can not be less than or equal to zero (0)

4.3.4 Max / Min Limit Error:

Example1:

<CONCOA>TR=69.9

Min / Max Limit Error

The upper limit for the Ramp Time (TR) is 60.0 (decimal) seconds.

Since, the TR value should not be greater than 60.0.

Example2:

For more examples of these kinds of error messages refer to 'More Max/Min Limit Error Examples' in section A2 of appendix A.

4.3.5 Argument Error:

Example:

<CONCOA>MD 825A 0800

Invalid Argument

The starting address is greater than the ending address.

Since, 825A (first address) is greater than 0800 (last address).

4.3.6 Length Error:

Example1:

<CONCOA>MD 800 0850

Invalid Argument Length

The start and the end address should be entered as a 4-digit number input.

Since, each address should be entered as a 4-digit number, 800 should be entered as 0800.

Example2:

<CONCOA>KP=123456

Invalid Length

Total number of characters typed is greater than the maximum

Since, you can only enter a maximum of 4 integers for KP command.

4.3.7 Other Error Messages:

Example:

These error messages include; address, number and option errors. To learn about them, refer to 'System Messages Examples, in section A2.2, A2.3 and A2.4 respectively of appendix A.

CHAPTER 5

THESIS RESULTS

Conclusions

The software program for controlling and monitoring a gas-regulator system is presented in this thesis. The program with a command line user interface is made available for CONCOA to centrally control their gas regulator control system using a stand-alone IBM-Compatible personal computer. Basically, two well integrated software (Monitor and Control) programs are written for this project (refer to the flow chart in figure 3.1 of chapter 3). This thesis only covers the monitor program that is the main program of the complete CONCOA software for this project. It works in the foreground as the main controlling authority for the end-user to control and monitor the CONCOA's gas regulator control system by means of a command prompt to enter software commands for system parameters.

This software program provides an access to system-set and system-response parameters for controlling and monitoring various system parameters using a computer's keyboard. The command set for this system provides a comprehensive and thorough control scheme for CONCOA engineers to monitor and control the gas pressure electronically and through a centralized control location. Even though it was not required by CONCOA, but I wrote a debug software module to access the 68HC12's memory locations (RAM and ROMs) when the microcontroller executes codes for CONCOA software from the Flash-EEPROM.

The results show that the system is running without any difficulty, and the successful tests of the software at the company's facility also indicate that the code is organized and well integrated with the main control program to calculate an output

voltage for CONCOA's gas regulator control system. Every software module was tested on an individual basis, and finally the complete program testing, after the individual modules integration, was very successful and was carried out using different kinds of error scenarios.

To make the user interface interactive and friendly, a set of messages is stored in the Flash EEPROM to display in an event to alarm the user about the current status of their command line user request. The software program is well commented and describes the details of almost every single line of code. As a whole, the code for this software is simple to follow and easy to read.

The software modules for this industrial project are made generic to include any new commands or lines of code to add or remove system or control parameters. Also to make it simple, several programmable modules are saved into different files and are linked using the 'include' directive for the assembler used in this project (refer to section 2.3 'Software Requirements' of chapter 2). A complete help module is also available to get command line help at CONCOA prompt. This help module is used for describing different commands with their associated parameters and a correct syntax.

5.2 Recommendations for the future studies:

At this stage we have a software user interface for CONCOA, which provides a command line user access to control the gas pressure electronically. It is also noticed that the CONCOA engineers have a simulation program that keeps track of the system behavior and extract some thousands of data samples while system is running. Those data samples can then be analyzed using a software package like Microsoft® Excel™. Since this system uses the standard emulator software (refer to 'Assembler and Screen Emulator' in section C2 of appendix C), it can only be used in with a stand-alone computer.

One of the best features of the monitor program is the debug software module. But this module can only access the memory locations when the code is executed from the Flash-EEPROM. To provide a better debugging system all the microcontroller registers should also be accessed by the system any time during the normal software execution. But the question is what kind of user interface should be used in the future to come up with all the features discussed above.

On the basis of above discussion, it can be concluded that a graphical user interface is not only more user friendly but it also provide a great deal of interactivity and features that are difficult to acquire using a command-line interface. Using high level languages such as Visual Basic™ or Java™, the coding becomes easier and transportable among different operating (ex: Windows, UNIX) and hardware (ex: IBM, Macintosh) platforms. These high-level languages like Visual Basic™ provide a nice integration with the software packages like Excel™ for analyzing and drawing graphs automatically using some set of commands. Using a high-level language like Visual Basic would help CONCOA to not only control their system electronically but also analyze the useful data for future studies or for a software alarm system using PC speakers. A software dialer can also be embedded with the actual software.

Based on the above discussion, I can suggest the following recommendations for the enhancement of this project of CONCOA:

1. Designing a graphical user interface for the monitor, control and debug system for CONCOA using a high level language such as Visual Basic, C, C++ or Java.
2. A multi-user support using a computer network for this graphical user interface.
3. Enhancement of the graphical user interface to include some of the emulator characteristics, such as: uploading and downloading the software codes into ROM and/or Flash EEPROM of the microcontroller.
4. An event log file with different sorting attributes such as: parameters, time, errors, alarms, system messages, last modified, user names, etc.

5. The graphical user interface should be capable to sort this data from the event log file to come up with some meaningful results and graphs for the future studies and analysis.
6. An automatic event log file should be generated, for software or a hardware failure containing the final values of all the important system and control parameters.
7. Enhancement should be made to the debug program to include the features to show the memory contents of the microcontroller's registers such as, Accumulators, index registers, program counter, stack pointer and the status or the flag register, at any place of the executable software code.
8. A periodic update and simulation of all the parameters' values from the hardware into the graphical user interface as a separate display screen and the introduction of a software switch key to toggle between the user interface and the debug interface.
9. The debug interface should be able to catch the values for the new features, as mentioned in the enhancement model for the debug program, without disrupting the program execution.
10. Support for the hardware alarm system into the graphical user interface for an alarm activation using software codes within the program.

REFERENCES

[1] Gene H. Miller, *Microcomputer Engineering*, Prentice Hall, Inc., 1999.

[2] John B. Peatman, *Design with Microcontrollers*, McGraw –Hill Book Company, 1988.

[3] Frederick F. Driscoll, Robert F. Coughlin, Robert S. Villanucci, *Data Acquisition and Process Control with the M68HC11 Microcontroller*, Macmillan Publishing Company, New York, 1994.

[4] Jonathan W. Valvano, *Embedded Microcomputer Systems*, Brooks/Cole, 2000.

[5] Jean J. Labrosse, *Embedded Systems Building Blocks*, CMP Media, Inc., 2000.

[6] Michael Basr, *Programming Embedded Systems in C and C++*, O'Reilly & Associates, Inc., 1999.

Appendix A

Command Set and System Messages Examples

A1 Debug Commands:

Following are the examples for the memory modify and memory dump command syntax rules.

A1.1 Memory Modify Command

The memory modify (MM) uses the following syntax to modify a memory location in the RAM of 68HC12.

<CONCOA> MM XXXX			Followed by a carriage return
XXXX	**YY**	**ZZ**	Followed by a carriage return
XXXX+1	**YY**	**--**	

Where ***XXXX*** is a 4-byte address of a memory location in the RAM, **YY** are the contents at that location and **ZZ** is the new value you have just entered. Once you press a carriage return after making the changes in the RAM, on the next line it will show the very next address of the memory location and its contents (YY). The two dashes (--) are basically indicating the cursor position.

The above syntax and the screen display can be seen in the following captured screen while the command was running from the Flash EEPROM of the 68HC12 evaluation board.

```
<CONCOA> MM 0820

0820     55 66
0821     66 77
0822     77 88
0823     88 99
0824     99 AA
0825     AA BB
0826     BB CC
0827     CC DD
0828     DD EE
0829     EE FF
082A     FF GG
Invalid Hex Number
Only hex numbers from 00 to FF can be entered

082A     FF 123
Invalid Data Input
Input data should not exceed 8 bits (1 byte).

082A     FF 00
082B     00 FF
082C     02 12
082D     00 _
```

Now if you repeat the same action as in the first line it will show a third line after replacing the cursor with a new value (ZZ). But, you can also use the available options such as:

XXXXYY + Followed by a carriage return

If you use a positive sign (+) or just press the carriage return, it will simply print the same line with the address of the next memory location and its contents and the cursor will wait for your input again. The contents of that memory location will not be changed.

Using a minus sign (-) instead of a plus sign (+) will show the previous address and its contents. If you press the minus sign recursively, it will just scroll the screen with the memory location and contents of the previous address

and will leave the contents unchanged. The above syntax and the screen display can be seen in the following captured screen.

```
<CONCOA> MM 0820

0820     66
0821     77 +
0822     88 +
0823     99 +
0824     AA +
0825     BB +
0826     CC +
0827     DD +
0828     EE +
0829     FF +
082A     00 -
0829     FF -
0828     EE -
0827     DD -
0826     CC -
0825     BB -
0824     AA -
0823     99 -
0822     88 -
0821     77 -
0820     66
0821     77
```

If you use a decimal point (.) as the input, the program will come out of the loop followed by a carriage return. This is the only point of exit for this memory command. For any mistyped option or parameter value the system will display a corresponding error message and print the same address and its contents on the next line for the user input.

The following captured screen for memory-modify command will explain how this option work when the command is running from the Flash EEPROM of the 68HC12 evaluation board.

```
0829    FF +
082A    00 -
0829    FF -
0828    EE -
0827    DD -
0826    CC -
0825    BB -
0824    AA -
0823    99 -
0822    88 -
0821    77 -
0820    66
0821    77 *
Invalid Command Option
The only options to use with this command are: +,-,., and carriage return

0821    77 /
Invalid Command Option
The only options to use with this command are: +,-,., and carriage return

0821    77 +
0822    88 -
0821    77 .
<CONCOA> _
```

A1.2 Memory Dump Command

The following syntax should be used to enter memory dump (MD) command.

<CONCOA> MD XXXX YYYY Followed by a carriage return

Where XXXX is the first or starting address and YYYY is the ending address to dump the memory contents.

It will display dumped memory blocks starting from XXXX and ending at YYYY in the following order, as shown below.

```
XXXX    PPPP PPPP PPPP PPPP    JJJJJJJJJJJJJJJ
XXXX    PPPP PPPP PPPP PPPP    JJJJJJJJJJJJJJJ
   .
   .
   .
   .
YYYY    PPPP PPPP PPPP PPPP    JJJJJJJJJJJJJJJ
```

Where 'P' is the memory contents of the memory-byte starting with the address on the first column followed by fifteen bytes. 'J' is the ASCII code for a corresponding 'P' value in the second column. The next line prints 'XXXX' which is the address of the 17th byte starting from the 'XXXX' on the first line and so on. If the ending address of the memory to be dumped is not a multiple of 16, the system still print the data until it reaches the very next multiple of 16. For example if a user wants to dump 30 memory bytes, the system will at least dump 32 bytes followed by a carriage return.

The following captured screen will explain how we can use the memory dump command to see the memory block we just modified using memory modify command. This screen capture is obtained while the command was running through the Flash EEPROM on the 68HC12 evaluation board.

```
0826    CC -
0825    BB -
0824    AA -
0823    99 -
0822    88 -
0821    77 -
0820    66
0821    77 *
Invalid Command Option
The only options to use with this command are: +,-,., and carriage return

0821    77 /
Invalid Command Option
The only options to use with this command are: +,-,., and carriage return

0821    77 +
0822    88 -
0821    77 .
<CONCOA> MD 0820 0840

0820    66-77-88-99-AA-BB-CC-DD-EE-FF-00-FF-12-00-00-2E    fw:::::::::::::.
0830    00-08-02-00-00-08-04-00-00-7B-00-11-00-4D-20-08    :::::::::{:::M :
0840    00-0E-07-76-00-3E-00-08-00-01-00-01-00-00-08-20    :::v:>:::::::::
<CONCOA> _
```

The following captured screen will show how we can dump the contents of a memory block in the RAM location where we are actually storing the input from the command line arguments. The starting location for the input capture buffer is $0810 in the RAM.

```
<CONCOA>
<CONCOA> MD 0800 0900

0800    02-21-10-01-06-90-BA-04-05-40-10-82-00-0A-00-02    :!:::::::@::::::
0810    4D-44-20-30-38-30-30-20-30-39-30-30-0D-39-39-80    MD 0800 0900:99:
0820    66-77-88-99-AA-BB-CC-DD-EE-FF-00-FF-12-00-00-2E    fw:::::::::::::.
0830    00-08-00-00-00-09-00-00-00-7B-00-11-00-4D-20-08    :::::::::{:::M :
0840    00-0E-07-76-00-3E-00-08-00-01-00-01-00-00-08-00    :::v:>::::::::::
0850    09-10-09-00-2E-2B-33-32-04-11-04-00-0C-04-00-80    ::::.+32::::::::
0860    0D-02-02-FF-00-04-00-07-02-18-00-07-00-00-00-00    ::::::::::::::::
0870    04-00-08-46-00-12-08-10-8A-30-18-00-80-34-08-60    :::F:::::0:::4:p
0880    08-81-38-38-08-80-30-00-00-48-18-00-01-16-20-04    ::04::4::H:::: :
0890    04-00-81-44-48-00-08-40-10-00-00-A0-00-00-08-00    :::DH::@::::::::
08A0    01-62-00-82-00-89-84-00-00-20-80-00-21-00-00-20    :b::::::: ::!::
08B0    18-C0-10-00-00-44-22-12-01-14-08-00-01-20-45-00    :::::D":::::: E:
08C0    04-00-00-80-40-40-00-22-08-04-0E-01-0C-82-00-00    ::::@@:":::::::::
08D0    C9-C0-02-00-DD-12-01-00-20-02-0A-98-1C-00-00-24    :::::::: ::::::$
08E0    00-00-00-10-04-00-00-02-00-00-04-4C-DD-12-76-72    ::::::::::::L::vr
08F0    00-00-00-08-01-00-00-00-40-C0-06-00-C8-00-02-00    ::::::::@:::::::
0900    04-21-C0-62-0A-00-10-02-00-09-21-00-00-00-04-00    :!:b::::::!::::::
<CONCOA>
```

The following captured screen displays a memory block in the Flash EEPROM also known as ROM-Data-Monitor as explained in section 3.2.1 in chapter 3. This memory block of the Flash EEPROM contains the static data for the monitor program such as; command-set-database, system-help, system messages, etc.

```
<CONCOA> MD 8200 8340

8200    64-65-64-20-6F-72-20-50-53-3E-20-50-55-2E-06-54    ded or PS> PU.:T
8210    6F-20-63-68-61-6E-67-65-20-50-53-20-74-6F-20-74    o change PS to t
8220    68-69-73-20-76-61-6C-75-65-20-63-68-61-6E-67-65    his value change
8230    20-74-68-65-20-75-70-70-65-72-20-70-72-65-73-73     the upper press
8240    75-72-65-20-6C-69-6D-69-74-20-28-50-55-29-20-66    ure limit (PU) f
8250    69-72-73-74-2E-06-07-44-69-76-69-64-65-20-62-79    irst.::Divide by
8260    20-5A-65-72-6F-20-45-72-72-6F-72-06-54-68-65-20     Zero Error:The
8270    76-61-6C-75-65-20-6A-75-73-74-20-65-6E-74-65-72    value just enter
8280    65-64-20-66-6F-72-20-74-68-65-20-61-62-6F-76-65    ed for the above
8290    20-70-61-72-61-6D-65-74-65-72-20-73-68-6F-75-6C     parameter shoul
82A0    64-20-61-6C-77-61-79-73-20-62-65-06-67-72-65-61    d always be:grea
82B0    74-65-72-20-74-68-61-6E-20-7A-65-72-6F-28-30-29    ter than zero(0)
82C0    06-07-49-6E-76-61-6C-69-64-20-50-61-72-61-6D-65    ::Invalid Parame
82D0    74-65-72-20-56-61-6C-75-65-06-59-6F-75-20-63-61    ter Value:You ca
82E0    6E-20-6F-6E-6C-79-20-69-6E-70-75-74-20-74-68-65    n only input the
82F0    20-64-65-63-69-6D-61-6C-20-6E-75-6D-62-65-72-73     decimal numbers
8300    20-28-30-20-74-6F-20-39-29-20-66-6F-72-20-74-68     (0 to 9) for th
8310    69-73-20-63-6F-6D-6D-61-6E-64-2E-06-07-53-79-6E    is command.::Syn
8320    74-61-78-20-45-72-72-6F-72-06-54-6F-20-67-65-74    tax Error:To get
8330    20-63-6F-6D-6D-61-6E-64-20-6C-69-6E-65-20-68-65     command line he
8340    6C-70-20-74-79-70-65-3A-20-48-50-3F-20-06-07-49    lp type: HP? ::I
<CONCOA>
```

The following captured screen displays a memory block in the Flash EEPROM also known as ROM-Data-control, which contains the static data for the control program.

```
<CONCOA> MD 9000 9140

9000    04-4C-76-72-00-00-00-08-00-02-18-03-00-01-08-4C    :Lvr:::::::::L
9010    86-06-7A-08-04-CC-90-BA-7C-08-05-FC-08-3C-CD-00    ::z:::::|::::<::
9020    20-18-13-CD-00-19-18-14-B7-C6-83-01-DC-FD-08-3A     ::::::::::::::
9030    18-13-7C-08-D4-18-04-08-D4-08-EC-18-03-00-00-08    ::|::::::::::::
9040    E1-18-03-00-00-08-DD-18-03-00-00-08-F6-86-77-5A    ::::::::::::::wZ
9050    57-86-65-5A-56-18-0B-FF-00-03-FC-90-00-7C-08-EA    W:eZV::::::::|::
9060    FC-90-02-7C-08-EE-FC-90-04-7C-08-F0-FC-90-06-7C    :::|:::::|:::::|
9070    08-F2-FC-90-08-7C-08-E6-18-0B-01-08-D6-79-08-E5    :::::|:::::::y::
9080    79-08-E9-18-0B-90-00-0B-18-0B-90-00-0B-18-0B-90    y::::::::::::::
9090    00-0A-18-0B-09-00-16-18-0B-01-00-80-18-0B-E0-00    ::::::::::::::::
90A0    86-18-0B-01-00-89-18-0B-25-00-8D-FC-08-46-CD-00    ::::::::%::::F::
90B0    FA-13-5C-90-18-0B-01-00-8C-3D-FC-08-46-CD-00-FA    ::\::::::=::F:::
90C0    13-D3-90-5C-90-4C-8E-01-79-08-E8-CE-08-E8-0E-01    :::\:L::y:::::::
90D0    04-3F-0E-01-02-18-0E-01-01-0A-86-70-5A-56-86-74    :?:::::::::pZV:t
90E0    18-20-00-1E-86-71-5A-56-86-77-18-20-00-14-0E-01    : :::qZV:w: ::::
90F0    01-0A-86-72-5A-56-86-76-18-20-00-06-86-73-5A-56    :::rZV:v: :::sZV
9100    86-75-5A-56-4F-08-04-02-20-FA-16-93-94-18-20-FF    :uZVO::: ::::: :
9110    B7-FC-08-D0-CE-00-10-18-15-7E-08-40-BE-08-42-18    ::::::::::~:@::B:
9120    2E-00-32-BE-08-44-18-2D-00-32-4D-08-F8-B6-08-E5    .:2::D:-:2M:::::
9130    81-01-18-27-00-A2-81-02-18-27-00-CA-FC-08-4A-8C    :::':::::':::::J:
9140    00-00-18-27-00-1D-FC-08-D2-CE-00-10-18-15-7E-08    :::':::::::::~:
<CONCOA> _
```

The following captured screen displays a memory block in the Flash EEPROM also known as Main program location as explained in section 3.2.1 in chapter 3. This memory block of the Flash EEPROM contains the executable codes for the main CONCOA software. Actually the microcontroller evaluation board starts executing the program codes from the Flash EEPROM location $8000 (by default) where the first instruction is a jump to the main program location, which is $A000 as shown below.

```
<CONCOA> MD A000 A140

A000    CF-0A-00-18-0B-00-00-C2-18-0B-34-00-C1-18-0B-0C    ::::::::::4:::::
A010    00-C3-86-F8-5A-09-4D-08-F8-79-08-60-79-08-71-14    ::::Z:M::y:`y:q:
A020    10-18-0B-00-08-87-18-03-00-64-08-38-18-03-00-11    :::::::::d:8::::
A030    08-3A-18-03-02-C1-08-3C-18-03-06-9C-08-42-18-03    :::::::<:::::B::
A040    00-8D-08-44-18-03-00-08-08-46-18-03-00-01-08-48    :::D:::::F:::::H
A050    18-03-00-00-08-4A-18-03-00-00-08-4C-18-03-00-02    :::::J:::::L::::
A060    08-56-16-AA-D9-CE-80-1C-7E-08-7C-18-0B-FF-08-6C    :V::::::~:|::::l
A070    CD-08-10-7D-08-76-FD-08-76-FE-08-7C-A6-00-81-07    :::}:v::v::|::::
A080    18-27-00-18-6A-40-81-3F-27-0A-02-7D-08-76-08-7E    :'::j@:?'::}:v:~
A090    08-7C-20-E2-08-7E-08-7C-18-20-01-50-16-AA-D9-18    :| ::~:|: :P::::
A0A0    0B-00-08-6C-79-08-6A-C6-01-F1-08-71-18-27-00-3E    :::ly:j::::q:':>
A0B0    CE-89-89-7E-08-78-79-08-6B-16-A0-CB-C6-07-F1-08    :::~:xy:k:::::::
A0C0    6B-18-27-00-29-08-7E-08-78-20-EE-3B-34-35-FE-08    k:':):~:x :;45::
A0D0    78-A6-00-16-AA-D4-81-07-27-0A-81-06-27-0B-5A-C7    x:::::::':::':Z:
A0E0    31-30-3A-3D-7A-08-6B-20-F7-16-AA-D9-20-F2-16-AA    10:=z:k :::: :::
A0F0    D9-CE-86-3D-16-AA-D4-96-C4-A6-00-5A-C7-81-07-27    :::=:::::::Z:::'
A100    03-08-20-F0-79-08-60-79-08-69-79-08-71-79-08-6E    :: :y:`y:iy:qy:n
A110    79-08-6D-79-08-64-79-08-6F-18-0B-0F-08-61-CD-08    y:my:dy:o::::a::
A120    10-7D-08-76-4E-C4-20-02-20-FA-96-C4-96-C7-81-08    :}:vN: : :::::::
A130    18-27-00-40-FD-08-76-6A-40-02-7D-08-76-73-08-61    :':@::vj@:}:vs:a
A140    18-27-08-F7-4E-C4-80-02-20-FA-16-AA-D4-5A-C7-81    :'::N::: ::::Z::
<CONCOA>
```

A2 Help Command:

The following captured screen shows how a one page command-line help will be displayed while the system is executing the software codes from the 68HC12 microcontroller.

```
<CONCOA> HP?

This is the HELP for CONCOA Software.
The following parameters should be entered as integer only:
KP, KI, TM, and IF.
 1.) To write value...say for KP... KP=XXXX +Enter Key.
       Where XXXX could be any integer from 0 to 9.
       The max. # of integers could be 4 and the min. is 1.
 2.) To read value... say for KP... KP? +Enter Key.

The following parameters should always be entered with a
decimal point:
PS, PU, PL and TR.
 1.) To write value...say for PS... PS=XXX.X +Enter Key.
       Where XXXX could be any integer from 0 to 9.
       The max. # of integers could be 4 and the min. is 2.
       A period (.) is a must to enter these parameters and there
       should always be one integer after the period.
 2.) To read value... say for PS... PS? +Enter Key.

The following parameters are read only:
PB, PO and IT.

<CONCOA>
```

A3. System Messages Examples:

A3.1 More Max/Min Limit Error Examples

Example1:

<CONCOA>IF=2

Invalid Value

IF values can either be zero (0) or a one (1)...

IF= 0 ... Set Pressure should be entered trough the PC Keyboard.

IF= 1 ... Set Pressure should be entered trough the Evaluation Board.

Since, IF is only used as a flag value for entering the set pressure either from the keyboard or from evaluation board.

Example2:

<CONCOA>PS= 99.9

Invalid PS Value

The upper PS limit is exceeded or PS> PU

To change PS to this value change the upper pressure limit (PU) first.

<CONCOA>PS= 8.9

Invalid PS Value

The lower PS limit is exceeded or PS< PL
To change PS to this value change the lower pressure limit (PL) first.

NOTE:

For the above example we have assumed the upper pressure value (PU) = 100.0 psi and the lower pressure value (PL) = 10.0 psi

A3.2. Address Error:

<CONCOA>MM 8000

Invalid Input Address

Only RAM (random access memory) locations from 0800(hex) to 0A00 (hex) can be modified.

Since, 8000 is a Flash-EEPROM location and memory modify (MM) command is used to modify RAM locations for 68HC12

A3.3 Number Error Example:

<CONCOA>MM 0800

0800 00 GG

Invalid Hex Number

Only hex numbers from 00 to FF can be entered

0800 00 --

This error shows up when a user try to modify the memory location and mistypes a two-byte hex number (GG) as shown above.

A3.4 Option Error Example:

<CONCOA>MM 0800

0800 00 =

Invalid Command Option

The only options to use with this command are: +,-,., and carriage return'

0800 00 --

This error shows up when a user tries to use an option in memory modify command and mistypes a character which is not available as an option.

Appendix B

More Flow Charts

The 'Control Program Enable Request Routine' flow chart in figure B1 describes how the control program is enabled using the 'DN' command. Once a user presses DN followed by a carriage return, the program control is immediately transferred to 'Norm' sub-module (see figure B1). This module increments and tests a flag whose value is initially set to zero. If this value is a binary one (1), which indicates that it is the first try to enable control program, the system branches to another sub-module 'Init_Cntrl' to enable the control program. If the binary value is greater than one this indicates that the user has tried to enable an already enabled program. To prevent this situation and to alarm the user, an error message is popped up telling user that the control program is already enabled and setting the value to binary two to avoid any other action of enabling an already enabled control program.

The 'Init-Cntrl' module sets the input /output port registers, initializes the AD and DA converter bits connected to port P and then finally initializes the timer as shown below in 'Output-Ports' and Init_Timer sub-modules respectively in figure B1. The clock frequency of timer is set to 1MHZ and a hex number is entered for timer interrupt to occur after every 8ms.

As mentioned earlier in the introductory chapters that the complete control program is written in timer interrupt service routine, so every time an interrupt occurs it causes the monitor program to transfer the flow of program instruction to the control program. The control is always transferred back from control program to the monitor program, when the program finds a return from the interrupt instruction (RTI).

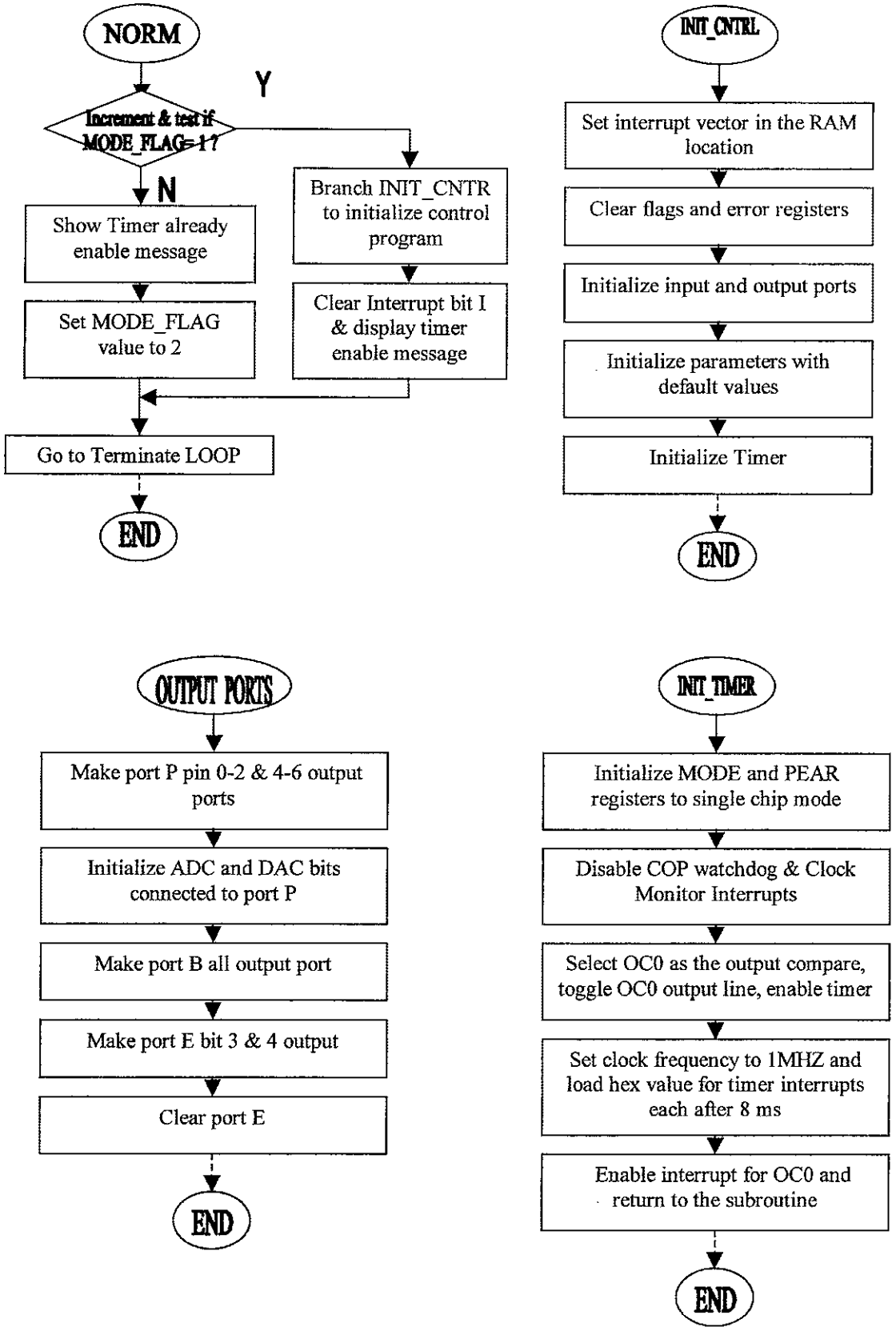


Figure B1 Control Program Enable Request Routine

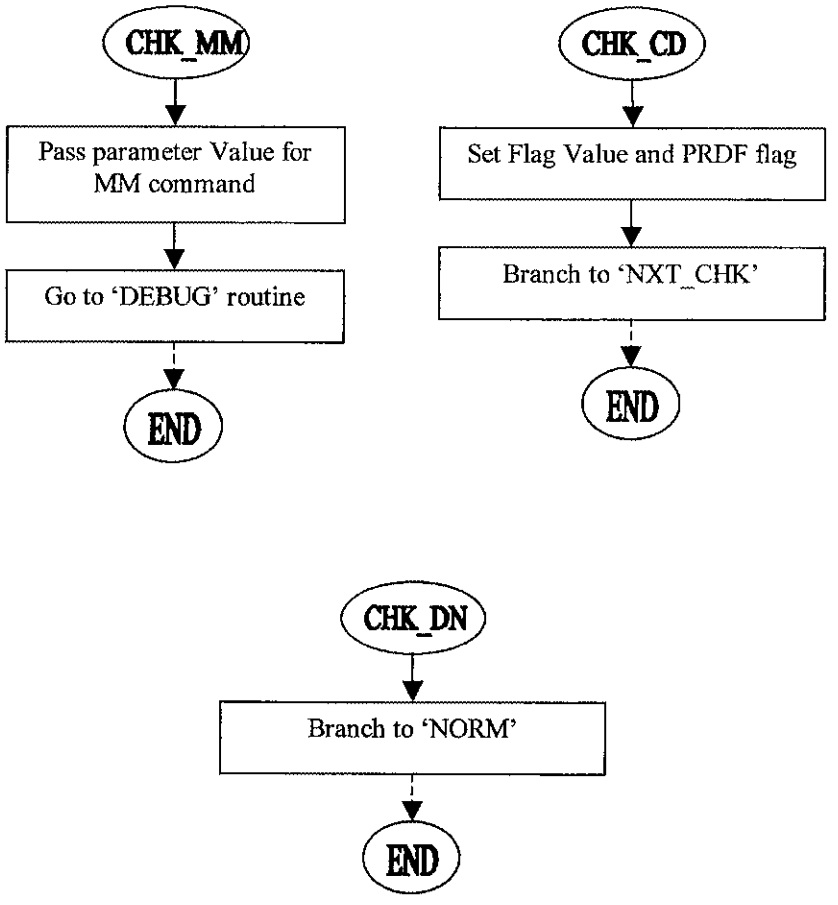


Figure B2 Pass Parameter Value for Other Commands

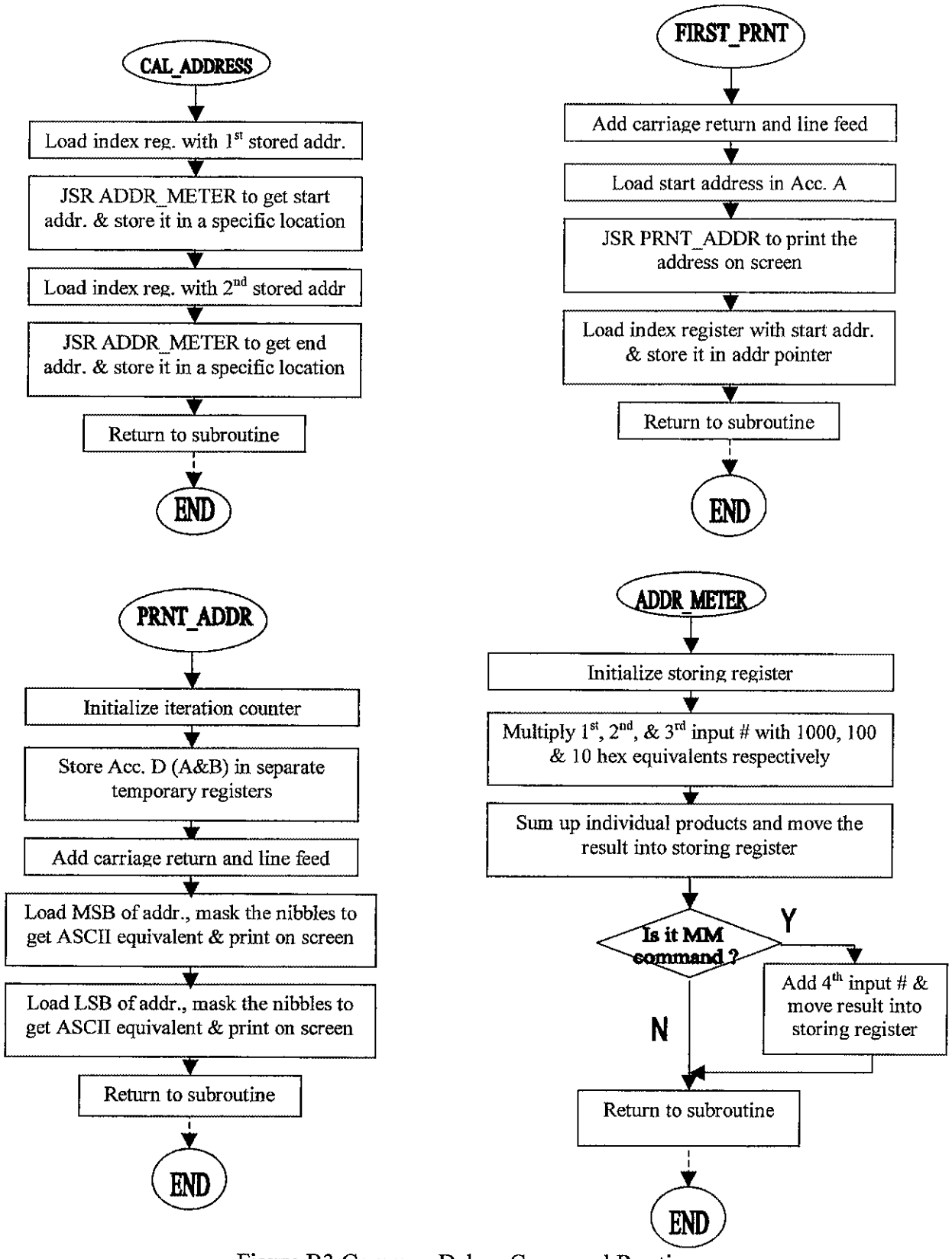


Figure B3 Common Debug Command Routine

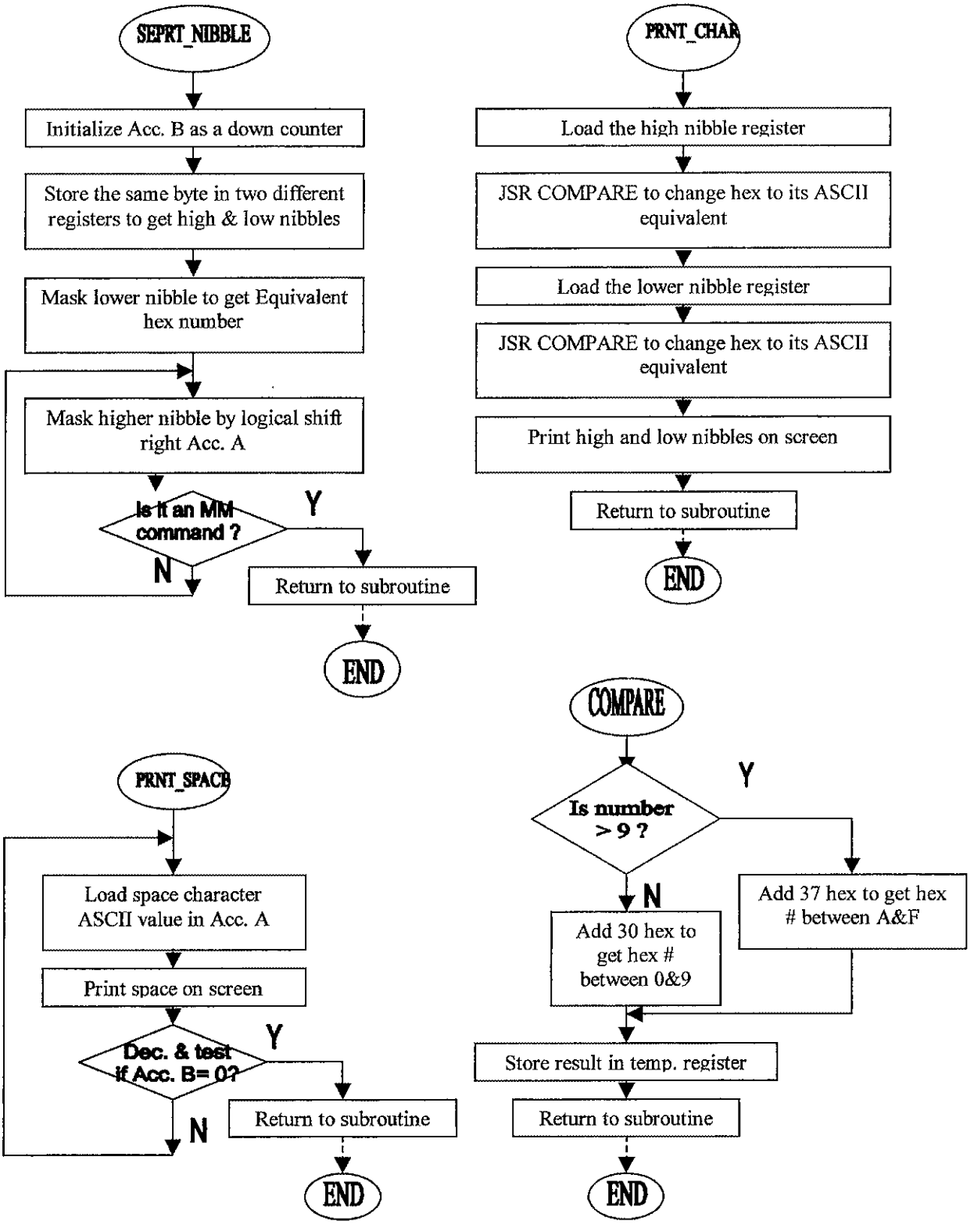


Figure B4 Common Debug Command Routine contd.

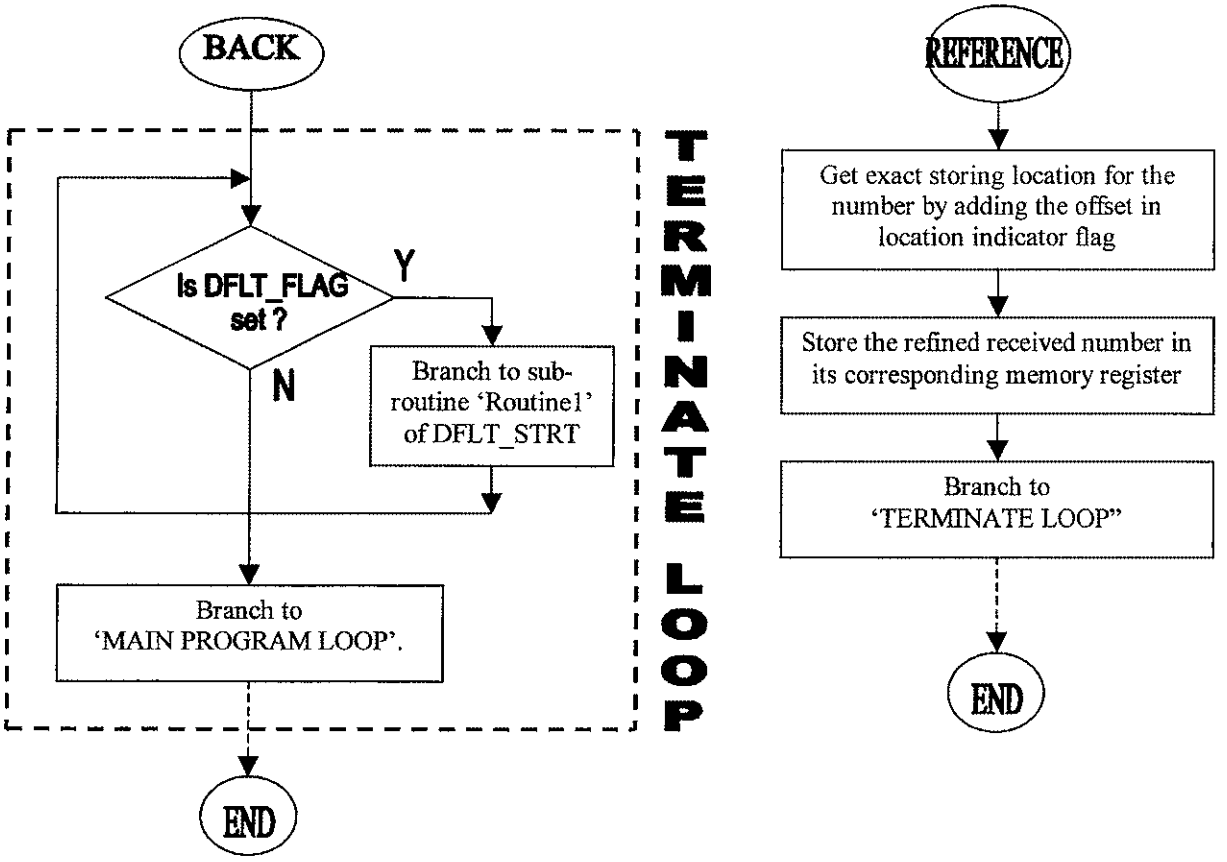


Figure B5 Terminate and Reference Modules

Appendix C

Flash EEPROM Programming

The software program for CONCOA is 3000 bytes long and thus needed to be written into the 32K-byte long on board Flash-EEPROM. After assembling the codes successfully, we took the following steps to transfer the whole program into the Flash-EEPROM of the microcontroller.

- Change the binary (user-assembled) file of the software (file type S extension S19) to include the interrupt vectors for the timer and/or serial interrupts
- For the M68EVB912B32 microcontroller evaluation board, change the jumper W3 and W4 to position 1 which is called the BOOTLOAD mode to reprogram the Flash EEPROM with the user codes
- Change the jumper setting for jumper W7 to VPP
- Connect 12V DC to jumper W8 which is VPP – needed to program Flash EEPROM
- Now power up the evaluations board, you will see the following three options available with this mode of operating the evaluation board.

  (E) rase, (P) rogramm, (L) oad:

- Hit E to erase the factory installed microcontroller evaluation board D-Bug12 monitor/debugger program.
- Once we are done with the erasure of the D-Bug12 program, hit P to reprogram the Flash EEPROM with our code in the Flash EEPROM.
- We used ProComm screen emulator software to download the S19 binary file for the codes of CONCOA software; other emulators such as Microsoft® windows™ HyperTerminal can also be used.

- Once the system is done downloading the codes into the Flash EEPROM, it will shoot a message 'programmed' on the terminal screen and then on the next line it will display the three options to erase, program or load the Flash EEPROM again
- Finally to see the Software for CONCOA running from the Flash EEPROM, we need to change the board setting back to the evaluation board mode and hit a reset if the CONCOA prompt does not pop up on the Terminal display

Appendix D

Assembler and Screen Emulators

MS DOS™ and P&E® editors were used to write the assembly codes for this project. The software codes are assembled using IASM12, version 3.14 from P&E Microcomputer Systems (for more information on this product visit the web site at http://www.pemicro.com/). This software program came with the purchase of 68HC12 microcontroller evaluation board.

A DOS™ or Windows™ based software emulator, such as: ProComm™, HyperTerminal, and MiniIDE™can be used to emulate the computer screen with 68HC12 microcontroller default prompt. For this project, the ProComm™ software is mostly used to emulate the microcontroller's command-line interface on a standard display monitor. The same software emulator is used to download the assembled binary codes (S record) for CONCOA software to the Flash EEPROM to replace the default 'DBUG12' monitor/debugger program, which is the default program (comes with the purchase of Motorola's microcontroller evaluation board) to communicate with the microcontroller evaluation board. The same binary codes may also be downloaded using Windows™ based emulators, such as: HyperTerminal and MiniIDE™. Since the Flash EEPROM is replaced with the software for this project, the monitor displays CONCOA prompt (<CONCOA>), which is the command line interface for this software.

SYED N. HYDER

Home: (757) 440-5734
E-mail: s.n.rizvi@usa.net

DEGREES:

Master of Science (Computer Engineering), Old Dominion University, Norfolk, VA, May 2002
Bachelor of Science (Electronic Engineering), N.E.D University, Karachi, Pakistan November 1997
One year Diploma (Computer Maintenance), Sindh Board, Karachi, Pakistan May 1996

PROFESSIONAL CHRONOLOGY:

CONCOA Corporation
Virginia Beach, Virginia
Graduate Researcher, January 1999 – December 2001

Technology Applications Center
Norfolk, Virginia
Network Administrator, May 1999 – August 2000

CONSULTING/PART TIME EMPLOYMENT:

Diner's Club International
Karachi, Pakistan
Intern, September 1997 – December 1997

Faith Computers
Karachi, Pakistan
Hardware Engineer, June 1995 – January 1998

TEACHING EXPERIENCE:

Old Dominion University
Norfolk, Virginia
Lab Assistant, September 2001- May 2002

PROJECTS:

- Designed Microprocessor's ALU, FIR/IIR filters, and advanced digital circuits using VHDL
- Socket programming using TCP/IP and UDP/IP protocols to make client/server, multi-group/multi-domain chatting software on a SUN Solaris platform using C language

- UNIX system and shell programming on a SCO UNIX platform
- Designed an automatic multi-mode traffic light control system using Intel's 8075 microcontroller
- Designed an I/O card for an IBM-compatible Personal Computer and wrote its device driver in C and Assembly language

COMPUTER SKILLS:

- Languages: C++, C, Assembly, VHDL, LabView, ORCAD, HTML, JavaScript
- Simulation tools: Funsim, Simnet
- Operating systems: Linux, UNIX (SCO, Solaris, HP), Novell, Windows 98/NT/2000 & Macintosh
- Packages: Adobe Photo shop, MS photo editor, MS Visio, MS FrontPage, Office2000/XP suite

OTHER SKILLS:

- ROM, EEPROM and Flash EEPROM programming experience
- Wrote machine codes on Motorola's 6800 trainers
- Worked with variety of microcontrollers, such as: Intel's 8031/51,8075; Motorola's 68HC11/12
- Worked with Motorola's microcontroller evaluation boards, such as: XC68HC912D0, XC68HC12B32 and M68HC11E9BCFN2
- Worked with analog/digital oscilloscopes, spectrum analyzer, frequency counter, function generator, digital/analog multimeter, etc.

LANGUAGES:

Reading and speaking competence in English and French

MEMBERSHIPS:

YMCA
French Cultural Center