DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/88D-63

8755 EMULATOR DESIGN

THESIS

John L Woods

Captain, USAF

AFIT/GE/ENG/88D-63

AFIT/GE/ENG/88D-63

8755 EMULATOR DESIGN

Thesis

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirement for the Degree of

Master of Science in Computer Engineering

John L Woods

Captain, U.S. Air Force

December 1988

Preface

The purpose of this design project was to design and build a device which could replace the 8755 microchip in experimental circuit designs. The 8755 emulator provides the standard input/output ports and 2048 bytes of memory. It also provides the ability to load this memory from the Z-100, to single-step a target circuit, break on a specified address, and control the I/O ports from the Z-100.

In designing and building this device I received support from several sources. I would like to acknowledge the members of the AFIT Electrical Engineering office for their support in helping me to acquire the parts and test equipment which were imperative to my thesis effort. I would like to thank LtC Charles Bisbee for his patience and guidance, as well as LtC Bert Garcia and Major Joe DeGroat for their support. Most of all I would like to thank my wife and children for their patience and support.

Captain John L Woods

Table of Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	1
Background	1
Problem Statement	2
Thesis Objective	2
Current Knowledge	2
Assumptions	4
Scope	5
Standards	5
Approach	6
User Module	7
Emulator Module	8
Target Module	9
II. Literature Review	11
Introduction	11
Emulation Process	12
Hardware Design Techniques	14
Software Design Techniques	16
Conclusion	19
III. Theory of Operation: Hardware	21
Introduction	21
Block Diagram	22
Central Processor Section	23
Emulator Memory	26
Target Memory	27
Serial Port	29
Parallel Port	31
Break Address Section	33
Target 40 pin Section	37
Special Function Section	39

IV. Theory of Operation: Software 42

Introduction 42
Bootup Software 44
Main Emulator Software 47
User Memory Control 48
Emulator Mode Control 50
Single-step control 51
Break Control 51
8755 I/O Control 54
Z-100 Control Software 55
Pass User Memory 57
Break Control Module 60
Parallel Port Control 62
Main Menu Control 64

V. Recommendations 67

Appendix A: Hardware Diagrams 69

Appendix B: Net Wiring List 84

Appendix C: Software Flowcharts 115

Appendix D: Source Code 131

Appendix E: User Guide 193

Bibliograhy 195

Vita 196

List of Figures

Figure		Page
1.	Emulator Block Diagram	69
2.	Emulator Schematic Diagram	70
3.	Bootup Flowchart	115
4.	Emulator Main Flowchart	116
5.	Emulator User Memory Flowchart	117
6.	Emulator Single-step Flowchart	118
7.	Emulator Break Control Flowchart	119
8.	Emulator Break Interrupt Flowchart	120
9.	Emulator Parallel I/O Flowchart	121
10.	Z-100 Main Flowchart	122
11.	Z-100 User Memory Flowchart	124
12.	Z-100 Break Control Flowchart	126
13.	Z-100 Parallel Port Flowchart	127
14.	Z-100 Single-step Flowchart	128
15.	Z-100 Target Control Flowchart	129

List of Tables

Table		Page
1.	Parts List	82
2.	Emulator Address Map	130
3.	Register/Comparator Truth Table	35

Abstract

This paper discusses the requirements to develop and build an electronic device to emulate the 8755 microchip. The design had five basic objectives: (1) Allow the user to download 8755 emulation memory; (2) Allow control of the target program from the Z-100 microcomputer; (3) Provide a single step capability; (4) Provide breaking at a specified address. (5) Allow the user to set or change the emulated 8755 input/output ports.

It describes the standard memory and input/output capabilities of the 8755. It describes in detail the emulator enhancement features to the standard 8755. The hardware circuits used to implement the emulator are discussed at the block diagram, component, and signal levels. It concludes with a detail description of the emulator software used to control the hardware.

8755 EMULATOR DESIGN

I. Introduction

Background

The Air Force Institute of Technology (AFIT) offers Advanced Microcomputer Engineering (EENG 687) as a primary course in its digital engineering curriculum. As a course requirement students must design and build a coprocessor board capable of interfacing to a Zenith Z-100 microcomputer. The coprocessor must be able to take control of the Z-100 upon command from a software routine. During the design and fabrication of the coprocessor, the software routine must be programmed into an Intel 8755 microchip. The 8755 is needed because it provides 16 input/output lines as well as the memory for the software routine. To program the 8755, the student must perform the following sequence of operations:

1. Load the software routine into a specified Z-100.
2. Erase the 8755 memory with an ultraviolet light.
3. Set-up the 8755 programming device.
4. Connect and adjust a power supply for the programming device.
5. Program the 8755.
6. Verify that the 8755 is properly programmed.

If the student finds errors in the 8755 software routine, the entire process must be repeated. The sequence takes a minimum of an hour to complete. Over the last several

quarters, the instructors have received numerous complaints regarding the time wasted as a result of this programming sequence (1). Eliminating this problem will allow EENG 687 students to concentrate on the specific course objectives.

Problem Statement

The current hardware and software available in EENG 687 are inefficient and hinder the students ability to successfully design and build the required Z-100 coprocessor within the course time constraints.

Thesis Objective

The purpose of this thesis is to design and build an 8755 emulator. The emulator will be a microprocessor based system that connects to a host computer via an RS232 interface. The user must be able to download 8755 emulation memory, cause the target processor to execute, single step, or break on access of specified memory locations, and set or change the emulated 8755 input/output ports.

Current Knowledge

Currently, the 8755 is available only from the Intel corporation. The 8755 provides 16, bit defineable, input/output lines and is directly compatible with the requirements defined for the Z-100 coprocessor board. In as the 8755 provides 2048 bytes (2 kbytes) of eraseable programmable read only memory (EPROM) for program storage. EPROM allows the user to store a program in the device and remove the device from the power source without a loss of

data. In contrast, random access memory (RAM) allows the user to load a program but will lose its data if the power source is removed. The coprocessor board will need to be installed and removed from the Z-100 on a frequent basis to allow all students equal time on the Z-100 systems. The requirement for bit definable input/output lines and non-volatile memory can only be satisfied by the 8755 or an 8755 emulator.

To eliminate the problems associated with the 8755 programming process, the emulator must replace the EPROM section with RAM. The emulator must appear to be a standard 8755 to the circuit into which it is installed. As a result the emulator must provide additional control circuitry to load and modify the target system's memory. Additionally, circuitry will also be required to provide additional features, which will further enhance the student's design and troubleshooting ability. Enhancements to the standard 8755 are transparent to the target system but will be very useful to the user. All enhancement features will be initialized and invoked from a host computer, but carried out by the emulator board. Currently designed enhancement features of the emulator board are:

1. A means of stopping the target system after the execution of each instruction. The user is able to disable this mode via the host computer keyboard.
2. Special registers monitor all address bus accesses from the target system. When a previously defined address

is detected, the emulator will halt the target system and advise the user of the halt condition and halt address.

3. The user may edit a software program on the host computer, while the target system is simultaneously executing the program in target memory. When the user is ready to load the new program into the target memory space, the emulator will halt the target system and perform the memory download.

4. The ability to change the 8755 port configuration without modifying the target memory.

Currently, literature from the 8755 manufacturer indicates that none of the enhanced features are available (2). The exact means of implementing these enhancements is discussed in chapters three and four.

Assumptions

The following assumptions have been made with respect to the design of the 8755 emulator project.

1. Users of the 8755 emulator are thoroughly familiar with the proper operations of the standard 8755 microchip. This is essential since the emulator will functionally appear to be an 8755.

2. Users are familiar with the operations of the host microcomputer and the MS-DOS operating system. This is required because the emulator will be controlled via the host computer serial port.

3. The user possesses the ability to write software routines in assembly language code and convert this code to machine readable format.

Scope

This project will investigate applicable hardware and software resources, then design and build an 8755 emulator. The specific goals of this project are to:

1. Provide a device which will meet all timing constraints of the 8755.
2. Provide two 8 bit input/output ports, with individual bit direction control.
3. Provide 2048 bytes of random access memory.
4. Provide the ability to perform single-step operations, with control via the host computer.
5. Provide the ability for the user to download control software from the host computer to the emulator board.
6. Allow the user to set break addresses to stop program execution when accessed.
7. Allow the user to reconfigure or read the 16 input/output ports from the host computer.
8. Provide documentation of all hardware and software associated with the emulator.

Standards

The emulator device and its control software must meet the following criteria:

1. The external electrical functions of the emulator must:

   a. Respond to all 8755 control signals.

   b. Meet or exceed the timing parameters of the 8755.

   c. Provide the same output power drive capability as an 8755.

2. The software used to control the emulator must be properly documented. This will allow future students to modify or update the device as required.

3. The device must perform each function listed in the scope section of this paper.

Approach

A practical solution to this problem is to develop an emulation device using commercially available hardware and in-house software to mimic all standard 8755 operations.

The emulator board, design is separated into three major functional sections. Each section is then designed and implemented with its unique set of hardware and software resources. The first section is the user module and refers to the host microcomputer, its operator input keyboard, and its serial input/output port. The operator issues all commands to the rest of the system via the computer keyboard and software routines.

The second section is the emulator module and consists of the hardware and software used by the 8755 emulator to communicate with the user as well as with an attached circuit. The last section is the target module and refers to

the hardware which is attached to the emulator via a 40 pin plug. The 40 pin plug represents a standard 8755 and plugs directly into the target circuit. Some portions of the emulator and target module are controlled, mutually exclusively, by both modules. A three module approach was selected in order to establish small, functionally cohesive, manageable modules.

Each module will be self-contained, with limited input and output paths. Controlling access to each module will ease the design, troubleshooting, and maintenance of the system. A brief description of each module and the interrelationship between modules follows.

User Module:

This section of the emulator consists of software which resides on a host computer. The hardware used is the host computer and will not require any modifications. The user software will provide instructions and menus to control the emulator module. The host computer will download the emulator control program to the emulator board automatically. This software is the emulator system monitor and controls all aspects of the emulator with the exception of the boot-up routine. The boot-up routine will reside on the emulator board at all times. After bootup is finished, a setup menu will be displayed. Each item on this menu should be invoked in sequence. The setup menu consists of the following list of options:

a. Pass target memory.

b. Set break addresses.

c. Configure ports.

d. Return to main menu.

Option (a) of the setup menu downloads the 2048 byte target software to the emulator. This software is used exclusively by the target system, but may be modified by the user module. Option (b), the set break address function, downloads a file containing 1 to 5 memory addresses to the appropriate registers on the emulator board. Option (c), the configure port function, defines the direction of data flow through each bit of the emulated 8755 ports. The last option, return to main menu, simply displays menu 2.

The main menu will be the starting point of all actions atfer system startup. The main menu consists of the following options:

a. Start or stop target system.

b. Enable/disable break detector.

c. Change break addresses

d. Enable/disable single-step

e. Edit/view/download user memory

f. Edit/view parallel ports

g. Exit to DOS

Emulator Module:

This module of the emulator contains all emulator hardware components. It interfaces with the host computer via an RS232 serial port and the target system via an extended 40

pin ribbon cable. To implement this module, a separate microcomputer was designed and constructed. This microcomputer consists of an 8088 central processing unit (CPU) and its associated memory devices. A read only memory module contains the software necessary to initialize the hardware upon power on. To communicate with the Z-100, a universal asnychronous serial receiver/transmitter (UART) is used. The UART receives parallel data from the CPU and converts it to a serial bit stream. This serial data is transmitted to the host computer via a serial port. This process is reversed when data is transmitted from the host computer to the emulator board. The emulator and user software, as well as all system commands, are received via this path. The emulator exercises control over the target system and the input/output ports. Control of the target system is implemented by connecting the emulator to the control lines of the ports and target system. When the emulator's CPU is controlling the system, the target system is inhibited. When a requested function is completed, the target system is restarted. The emulator may communicate with the host computer while the target system is in its operational mode without conflict.

Target Module:

This section of the emulator is located on the emulator board. The break on address function is invoked by the user, implemented by the emulator, and receives all necessary data

from the target. The target module functions in a passive mode and does not invoke any commands relative to the user and emulator modules. All control, address, and data lines of the target are monitored by the emulator. Depending on the data received, the emulator will invoke a number of responses and prompt the user for additional input when required.

II. Literature Review

Introduction

This literature search reviews an emulation system design for the Intel 8755 microchip. This device provides erasable programmable read only memory (EPROM) and input/output ports to peripheral devices (1). The 8755 is used in microcomputers and in industrial applications. The compatibility of this device with the 8088 CPU has prompted its use in educating students at the Air Force Institue of Technology (AFIT). In the school environment at AFIT, the 8755 must be programmed and reprogrammed during the course of laboratory exercises. Frequent changes to the software programs waste time because the 8755 must be erased, reprogrammed, and verified with each change. The entire process can take a significant amount of time and if one piece of code is incorrect, the entire process must be repeated. To increase the effective use of class time and resources, an 8755 emulator is needed (1). The emulator must provide program memory, the ability to single-step through a program, input/output ports, and reduce the time required to program or reprogram the device. Sound hardware and software development techniques must be used to design, fabricate, and test the 8755 emulator. This literature search addresses digital emulation techniques currently used.

Emulation Process

The emulation process starts with performing an in-depth requirement analysis, to ensure that what the user wants is clearly understood and documented (5:35-36). Pressman states that the system designer begins with customer-defined goals and constraints, and devises a representation of function performance, interfaces, design constraints, and information structures. Booch believes the systems engineer must bound the system by determining the scope of what is to be accomplished (3:76). Wilcox states that the requirement analysis process is an interactive one which is completed when the customer and system designer agree on exactly what is to be done (9:27).

Following the requirement analysis, the requirements are organized into tasks which are then alloted to teams with specified due dates (6:50). Each task is approached as a separate module with a set of input and output parameters. Each module may be accomplished independent of other modules as long as the boundary specifications are adhered to. If, during the course of developing the modules, the boundary parameters must change, then the change must be approved and coordinated by the system engineer (3:121). Modules may perform strictly hardware or software functions, but most perform a combination. Regardless of how the external processing is accomplished, the system engineer is only concerned with the system module at the interface level.

An alternate process for producing an emulation device is to perform a preliminary requirement analysis, which intentionally leaves many questions unanswered. (10:400-402). The goal is to start working as quickly as possible on a prototype of the desired system. The customer is then given a demonstration illustrating the strengths and weaknesses of the proposed design. Wilcox states that in many cases the user will make suggestions to add or delete features as a result of having a real model to base his decision on (9:63-64). The primary benefit gained from prototyping is the reduced likelihood of changes in requirements after the design process has begun. A disadvantage of prototyping is the cost in time and material to develop, build, and rebuild the demonstration model. The decision as to which emulation approach to take is made by the system engineer. The user only defines the end product performance specifications and the format of data entering the system from an external source (10:405). The implementation of the modules is determined by the hardware and software engineers. The emulation process suggested by Pressman lends itself to the 8755 emulator design. The time restraints imposed by AFIT prevent the use of rapid prototyping. A thorough requirement analysis will prevent late changes to the emulator performance goals and also save the time and material required to build a prototype. The requirement analysis is only the first step in an emulator design project. The requirement will eventually have to be implemented with

hardware and software . When the customer and system engineer have a firm set of requirements, the design engineer and software engineer may start system the development process (8:87-88). Initially, hardware and software engineers must work together to determine which modules pertain to hardware and which pertain to software. Both areas have a variety of design techniques which may be used to develop the modules.

Hardware Design Techniques

Engineering design is a creative process of devising a product to fill requirements specified by the customer (7:379). The problem-solving approach is one creative design technique and involves defining the problem, selecting a possible solution, evaluating the solution, generating other possible solutions, and selecting the best solution. Central to successfully using the problem-solving approach is the ability to plan a project. Objectives must be clearly stated, and time schedules developed and adhered to. Strategies to achieve objectives must be developed and a specific plan of action developed (4:185-186).

Top down design is another technique often used in hardware, as well as software, design (3:76-85). Top down design decomposes the problem at hand into seperate but functionally similar modules. Each module is decomposed further until the original complex task has been reduced to several small manageable tasks. Top-down design incorporates

heuristic design rules when the lowest level modules are implemented.

Regardless of the design technique used, the actual hardware components must be selected. This selection process will again include software engineers since their software will have to run on the hardware selected (6:100-110). Off-the-shelf components should be used whenever possible to save the cost of creating these components. Review and modifications are recommended at this stage since any change beyond this point will be costly .

The top-down design technique provides an excellent approach to the 8755 emulator design. At its highest level of abstraction, the emulator must appear as an 8755 microchip with enhanced features. This level of abstraction may then be broken down to a user module, emulator control module, and target system module. Each module may be broken into still lower level modules until each subfunction is represented as a separate module. When this level of abstraction is reached, the heuristic design rules may be applied to implement each subfunction. This approach to the problem provides a direct path to design and implementation.

The hardware design process is an important segment of the overall design effort, but equally important is the software needed to control the hardware. Hardware and software design must take place in parallel and in reaction to each other. Software design techniques exist which allow

the interactive development of software and hardware, but still addresses the special needs of the software designer.

Software Design Techniques

Any digital emulator project requires the extensive use of software design techniques and methods. Modern software design is an outgrowth of hardware and system engineering. It encompasses methods, tools, and procedures that enable the designer to control the process of software development (4:188). Software design methods encompass a broad array of tasks that include: software requirements analysis, design of data structures, program architecture and algorithm procedures, coding, testing and maintenance. Software design tools provide automated support for these methods. Software design procedures provide the sequence in which methods will be applied, the deliverables that are required, the controls that help assure quality, and the milestones to assess progress. The steps necessary for software design are referred to as software design paradigms (6:110-118). The paradigm chosen is based on the nature of the project and application, the methods and tools to be used, and the controls that are required.

The classic life cycle paradigm is the oldest and most widely used . Software design using the classic life cycle demands a systematic sequential approach that begins at the system level and progresses through analysis, design, coding, testing, and maintenance. Software analysis starts with the

system requirements analysis to allow the software designer to understand the nature of the programs to be written. The software designer must also be familiar with hardware to be sure the system components are compatible. Software design translates requirements into a representation of the software that can be assessed for quality before coding begins. During coding, the software design is translated into machine readable code. Software testing focuses on the internal logic of the software and the external functions of the system to ensure that a defined input will produce the desired output. Software maintenance applies each of the preceding life cycle steps to an existing program rather than to a new one.

The classic life cycle paradigm is not without its criticism in some situations (3:87). Booch points out that real projects rarely follow the sequential flow proposed by the classic life cycle, and often it is difficult in the beginning for the customer to state all requirements explicitly. Further, Booch suggests using the protyping paradigm in situations where these criticisms are valid. The prototype paradigm is best suited for situations where the customer does not have a complete understanding of system requirements.

Prototyping is a process that enables the developer to create a model of the software to be built. The model can take one of three forms: a paper prototype that depicts human-machine interaction in a form that enables the user to understand how such interactions will occur, a working

prototype that implements some subset of the functions required of the software, or an existing program that performs part or all of the functions desired but has other features to be improved upon in the new development effort. All prototyping approaches begin with the customer and designer defining overall objectives for the software, identifying whatever requirements are known, and outlining areas where further definition will be required.

A quick design is then initiated and focuses on a representation of those aspects of the software visible to the user. The quick design leads to the construction of a prototype. The prototype is evaluated by the customer and is used to refine requirements for the software to be developed. A process of interaction occurs as the prototype is refined to satisfy the needs of the customer, while simultaneously enabling the designer to better understand what needs to be done. The prototype is usually a first try, throw away model and may cause some problems.

A problem associated with the protype paradigm is that when the customer is informed that the device must still be built from scratch, he usually objects. The appearance of needless waste is a problem with this approach. If the customer demands a few quick fixes to the prototype instead of a complete design and fabricaton, problems with quality and reliability may result (9:122-125).

The 8755 emulator is well suited to aspects of the classic life cycle and the prototyping paradigm. A thorough analysis and design will ensure minimal redesign problems as the project progresses, and prototyping will ensure that no important design considerations were overlooked. The combined application of these techniques provides a direct and reliable approach to designing the emulator software. When the software is completed, it must be tested on the hardware it was designd for, and receive final approval from the system designer.

Conclusion

A clear need exists for the development of an enhanced 8755 microchip. Currently, there is not a commercially available digital device to satisfy the needs of AFIT (1). The emulator will provide memory to match that available on the 8755 and a complete set of I/O ports. Enhancement features include the ability to reconfiqure ports from an attached microcompter, to stop the emulater at any given time, and to allow easy reprogramming of the 8755 memory. The process of designing the emulator begins with a thorough requirements analysis involving the customer and system designer. The requirements will be divided into modules which will be small and manageable. The modules will be implemented using current hardware and software design techniques.

The engineering design process is a creative process which involves making all the decisions necessary to

implement the design requirements successfully. The designer could use the problem-solving approach, heuristic design rules, or top-down design. For the 8755 emulator, the top-down design is most compatible since this approach requires fewer initial designs and matches the requirement analysis process closely. By aligning each requirement with a module to be implemented and then decomposing the modules into subfunctions, an accurate design is probable. The subfunctions are implemented at the hardware interface level and require close coordination between software and hardware designers.

The software design must be conducted simultaneously and interactively with the hardware design. A combination of the classical life cycle and protyping paradigms provide the most effective approach to implement the requirements defined in the requirement analysis. This approach also provides the interaction needed between the hardware and software development efforts. Each paradigm individually has problems and limitations which can be avoided by selective application at the appropriate time. Together these paradigms provide rapid but accurate software development and implementation.

III. THEORY OF OPERATION: HARDWARE

Introduction

This chapter provides detailed information on the hardware devices used in the 8755 emulator. The software to control the system is presented in chapter 4.

The 8755 EPROM/I-0 device is produced by the Intel corporation. Intel produces a family of micro devices which are all designed to work together with minimum support logic and driver chips. To ensure maximum compatibility between the emulator and the actual timing and power parameters of the 8755, Intel chips are used when possible.

The overall purpose of the emulator system is to provide all the features available on a standard 8755, and the additional capability to single step, load the user memory directly, break on specified memory addresses, and control this process via an RS232 serial link to a host computer microcomputer. The user circuit must also be able to use the standard 8755 features when desired, while the emulator circuit remains transparent in terms of timing constraints.

In order to accomplish the requirements listed above the 8755 I/O and memory hardware were broken into two separate sections of hardware. The additonal special features required were accomplish through the use of several subsections which are described in the remainder of this chapter. Many of the devices used in the emulator circuits require software inputs

to initialize and control them. When a device requiring software is discussed, the discussion will be brief and the full explaination reserved until chapter 4.

Each major functional section of the emulator will be discussed separately so that all details of that subsection are presented as a package to the reader of this paper. The block diagram (figure 1) and schematic daigram (figure 2) referred to in this section may be found in Appendix A.

Block Diagram

The block diagram for the 8755 emulator is shown in figure 1. This diagram breaks the emulator circuit into eight functional areas. The central processor section processor exercises control over all over parts of the system. The emulator memory section contains all system memory which is used exclusively by the central processor. The target memory section contains the user 2 kilobyte static ram which is used primarily by the target, but is also accessible by the central processor. The serial port section is built around an Intel 8251A Universal Asynchronous Synchronous Transceiver (UART) and contains all the hardware necessary to support serial communications. The parallel port section contains an 8755-2 I/O memory chip and the necessary support devices. The parallel section is accessible by either the target or central processor. The break section monitors the target address bus for memory address loaded from the host computer via the central processor. Upon detection of a memory access

by the target which matches any of the stored addresses in the break registers the target is halted and the central processor is sent an interrupt request. The break section may be disabled through software control from the host computer. If the break section is in its enable mode the interrupt request will advise the host computer of which address has been detected. The target system remains halted until restarted by a command from the host computer. The target module contains a 40 pin socket plug which replaces the 8755 in the application circuit. All pins are identical and all 8755 features may been accessed by the target through the plug. The target section also contains additional hardware to support the enhanced features of the emulator listed in the introduction. The special features section includes the harware necessary to provide the emulator single-step, target single-step, and mode control functions. The mode control function is used to control whether the emulator or target system has access to user memory and parallel I/O ports at any given time. Access to these two section is mutual exclusive under all circumstances. The remainder of this chapter provides a detail discussion of each section of the block diagram.

Central Processor Section

System timing is provided by U9, an 8284A clock generator and driver. The 8284A is driven by a 14.3 MHZ crystal oscillator. The CPU operating frequency is 4.77 MHZ at the

output of U9 pin 8. The PCLK output is used as the internal CLK signal to the UART(U3) and is 2.39 MHZ. The 8284 provides synchronization of the ready input signal and the system clock. The RDY output is a synchronized signal generated by the into at pin 4. This input is used to halt the CPU during single-step operations, by driving U8 (8088) pin 22 low. The system reset is input to the 8284 as a low on the RES* pin when switch S1 is pressed. This signal is synchronized and inverted then output at U9 pin 10 as a high. The CPU, UART, and 8755 are reset by this signal. For a system reset the signal at U8-21 must be high for a minimum of 50 microseconds. Capacitor C1 and resistor R5 ensure that the minimum timing requirement is satisfied. When this happens U8 will clear its status flags, DS, SS, and IP registers. The CS register is set to FFFFh, thus FFFFO is the address at which the CPU restarts when reset returns to a low state.

The heart of the emulator is the 8088-2 (U8) central processor unit (CPU). The 8088 provides the address, data, status, and control signals to direct the emulator operations. The address lines AD0 through AD7 and A16 through A19 are multiplexed to also function as data and status pins, respectively. These signals must be latched in order to be available for an entire machine cycle. Address latches U6 and U7 are used to latch the address bus during T1 of each cycle. The signal used as a latch strobe is the address latch enable (ALE). During each bus cycle from the 8088-2, AD0

through AD7 and A16 through A19 change to valid data and status during T2. This data remains valid until the middle of T4. Address lines A8 - A15 are not multiplexed and are valid for the entire machine cycle. This timing arrangement of the ALE is important in the implementation of the target and emulator single step features. The ouput of U6 and U7, plus A8 through A15 form the address bus used to drive all emulator devices (figure 2a). The latched address remains valid until the falling edge of the next ALE, at which time the new address is strobed into U6 and U7.

The 8088-2 control lines used by the emulator include the RD*, WR*, ALE, IO/M*, Reset, MN/MX*, DT/R*, DEN*, and Ready. The RD* and WR* controls lines are buffered through U60 (7407), because the fanout capability of both lines is exceeded, the other control lines of U8 did not require buffering. The RD* signal is used to turn on the output drivers of a selected device. When the RD* line is low the data bus is strobed into U8 (8088). When WR* is low the data present on AD0 through AD7 are strobed into an addressed device. The ALE is also used to strobe U36 (8755) control signals and to reset the emulator single step flipflop. If the current machine cycle is accessing memory the IO/M* pin will be low. If the machine cycle is an IO access then this pin will be high. For the emulator the 8088-2 is configured to operate in the minimum mode by tying the MN/MX* to VCC. The DT/R pin controls the direction of data flow through U14 (74F245) and U35 (74F245) during each machine cycle. When

DT/R* is high data is transferred from the CPU and when the pin is low data is moves toward the CPU. The DEN* output is used to control the outputs of the data transceivers when the user RAM and 8755 are accessed. Finally, the INTR pin is used to detect when the break section has detected a break address. This pin may be disable with a software instruction, but if it is enabled the CPU outputs a interrupt acknowledge on INTA. Following each break detection the INTR input is disabled and must be reset by the interrupt handler routine.

Emulator Memory

The emulator memory space is addressable only by U8, and therefore does not require any multiplexed address and controls lines (figure 2c). All addresses discussed in this paper are in hexidecimal format. The emulator memory is controlled by U5, an Intel 8205 3-to-8 decoder. Address lines A16, A17, and A18 are used to drive the decoder input. Output U5-15 is ORed with A15 to control U15 (MSM62256) and ORed with A15* to control U16 (MSM62256) chip enable inputs. These two static RAM chips form page 0 of the 8088 memory space. Addresses (00000-07FFF) are located on U16 and addresses (08000-0FFFF) are located on U15. The emulator memory map is shown in table 2. Output U5-7 provides the emulator clip select to U17 (2716), which is page 7 of the memory space. A 2716 EPROM was used because it was readily available in the engineering laboratory. The system ROM is located on U17 with addresses 70000 - 707FF. Since A16-A18 are all ones

after a reset, and the lower 11 address bits are 7F0, the emulator begins its operation from the address 707F0. Decoder output U5-13 is used to control the system mode flipflop U39-B (7474). Decoder output U5-12 and U5-13 control the target system single-step circuits U65 (7474). Both of these control lines will be discussed in greater detail later. Finally, output U5-9 controls U18 (UM6116), which is page 6 of the emulator memory space. Static ram U18 may be addressed by the target system also and will be discussed in detail in the next section.

The emulator I/0 address space is controlled by U38 (74154) a 4-to-16 decoder. Each output of U38 will be discuss in the functional section where it is used. The control inputs to U38 consist of address lines A4 - A7 and an inverted IO/M* control signal. The IO/M* signal ensures that U38 and U5 (8205) are never active at the same time. This separation of memory and IO space is necessary since the A4 through A7 are used throughout the emulator.

Target Memory

Another user requirement is for the emulator to eliminate the 8755 ROM, and to provide the target with equivalent user RAM. Additionally, access to the user by the emulator system and the user's CPU ram must be mutually exclusive. To accomplish this requirement, the address lines A0 through A10 are multiplexed using 74F157A 2-to-1 multiplexers. Each multiplexer has two sets of four inputs and one set of four

outputs. Address lines A0 through A3, A4 through A7, and A8 through A10, and CS* are applied to U10, U11, and U12, respectively (figure 2c). One set of these signals come from the emulator and an identical set comes from the user circuit. The output of U10, U11, and U12 are applied to U18 (UM6116). The select line pin 1 of U10 through U12 controls which set of input signals are connected to the output. When select is high the emulator is in control of the SRAM. Conversely, when select is low the target system has access to the SRAM. The select lines of all multiplexers in the emulator are control by the output of the mode flipflop U39-A (7474). The write enable input (U18-21) is enabled only by the emulator. This limited access to the write enable is necessary because the a real 8755 does not have a write capability.

The output of U18 is connected to U14 and U13. These devices are 74F245 transceivers and are required to isolate the SRAM from the system data bus. The output enable input on U14 is controlled by the ORing of chip select of U18 and data enable of U8. The output of U41-A (7432) is low only when DEN* and CS* are both active. The direction of data flow through U14 is controlled by the CPU RD* signal. Transceiver U13 performs a similar function for the target circuit's access to U18 (UM6116), except data flow is only allowed from U18 into port B and out of port A. This prevents the user from inadvertently writing to U18. Output enable of U13

(74f245) is controlled by ORing IO/M* (MISC1-7), CE1* (MISC1-1), and the target RD* (MISC1-9) signals. When all three inputs are low U40-D will produce a low when U18 is accessed by the target. The 74F157A and 74F245 devices were selected to limit the amount of delay introduced, while allowing a flexible means of gaining access to U18. The multiplexer-transceiver combination adds only 14 ns delay, and is transparent to the user.

Serial Port

A primary requirement of the emulator system is the ability to communicate serially with the host computer microcomputer. This is accomplished via the serial port J2 on the rear of the host computer. The emulator uses U3 an 8251A universal asynchronous receiver transmitter (UART) as its serial port. The 8251A is capable of being programmed for asynchronous or synchronous communications. This project uses only the asynchronous mode, and operates at 9600 baud, with odd parity, and one stop bit. The specific software required to configure the UART is discussed in the next chapter. Data transferred between the CPU and UART is in an eight bit parallel format. An 8286 transceiver (U4) is used to isolate the UART from the CPU. The outputs of U4 are tri-stated unless the UART's chip select is low. The direction of data flow through U4 (8286) is controlled by an inverted RD* from the CPU. The UART converts the parallel data to a serial string and inserts start, stop, and parity bits. This string

is transmitted serially on the TXD pin U3-19 (8251A). The UART output is standard TTL output, which must be converted to RS232C signal specifications. The 1488 line driver (U1) converts the TTL levels 0V and 5V to -12 and +12, respectively. The output of U1 is connected to the RS232C cable and received by the host computer. Data terminal ready (DTR*) and request to send (RTS*) signals are also provided by the UART and passed through the 1488 driver. To receive data from the host computer the RS232C signals must be converted to TTL prior to being applied to the UART. Line receiver U2 (1489) converts the input data string, clear to send (CTS*), and data set ready (DSR*) signals. The use of the DTR*, RTS*, CTS*, DSR* handshking signals will be discussed in the next chapter. The RS232 cable also provides a ground return path between the host computer and emulator.

The receive and transmit clock is provided by an external function generator via connector J2. At 9600 baud the required clock frequency is 153.6 Khz when the baud rate factor is programmed for 16 (see chapter 4). For proper operation U3 (8215A) requires a minimum clock period of 310 ns. The system clock of the cpu has a period of 210ns, but the PCLK (U9-2) satisfies this requirement with a period of 420ns.

The control signals of the 8251A are the CS*, RD*, WR*, and C/D* inputs. The CS* and RD* must be low to transfer data from the UART to the CPU. The CS* and WR* must be low to transter data from the CPU to the UART. The C/D* input is

connected to A0 on the address bus and determines which internal register is being accessed. A detailed description of register control is covered in the next chapter.

Parallel Port

An 8755 has 2 kbytes of EPROM and two 8 bit data ports (figure 2j). The data ports may by used as inputs or outputs and each bit is individually defineable through software. Software requirements for the 8755 are covered in the next chapter. The literature review found that the 8755 is the only commmercially available device with bit-defineable ports. To implement this feature the emulator is using an 8755-2 (U36), the the memory section of this device was not used in this application. Since the emulator and the user must have access to the 8755 ports three more 74F157A multiplexers, U31, U32, and U33 are used to control access to this chip. Multiplxer U31 directs ALE, RESET, RD*, and WR*; U32 directs CLK, VDD, A0, and A1; U33 directs CE2, IO/M*, CE1,and IOR* from the emulator or user CPU control lines. The active set of inputs at any given time is determined by the mode flipflop U39-A (7474). All outputs of the multiplexers go to U36 to allow control of this device by either source.

When CE1* and CE2 are active the 8755 will strobe the address and IO/M* lines on the falling edge of ALE. To prevent extraneous data during an access to user ram, the CE2 input is ANDed with the target IO/M* control line before being applied to U33 (74F157). The output of this gate will

drive CE2 low during each memory access by the target system. This will inhibit the 8755-2, thus preventing an I/O operation. The 8755 does not strickly require the use of the IO/M* line to perform an I/O operation. The I/O may be accessed by using only the IOR* and IOW* lines. This option is still available to the target user, but the emulator was not designed using these lines. If this approach were taken the possibility of bus contention would exist when the user tried to perform a normal memory access. This problem only exists from the target system, because to it the 8755 uses the same chip enables for its memory and I/O operations. The emulator directly accesses the user memory and the 8755, because from its perspective theses devices have separate chip select addresses. The clock input to U36 was routed through the multiplexer to allow the user the option of inserting wait states, but the 8755 typically does not require wait states with Intel microprocessors. The clock is only used to drop the ready line U36-6 during each bus cycle. The emulator system does not use the ready line with its operations with the 8755. The 8755 I/O ports are controllable by the emulator or the target, but the port outputs are connected only to the target system. Thus when the user performs an I/O operation through the ports the data is receive and transmitted directly to his circuit application. The user may also drive these ports from the emulator, but the output will still be applied to the target circuit. If

the user desires to read the I/O ports via the emulator the data may be displayed at the host computer terminal. The emulator may also be used to reconfigure the data direction registers while the target is in its halt state. This feature allows the user to trouble shoot a circuit without having to change the user program which could also be used to do the reconfiguration.

The key hardware which provides the user with these flexible features are U34 (74F245) and U35 (74F245). These are 5 ns maximum delay data transceivers used to provide isolation between the emulator and target data busses. Transceiver U34 is connected to the target and is only enabled when the mode flipflop is set to the target mode and the target is performing an I/O operation. These two signals are combined with AND gate U56-C (7400), with this output applied to the OE* input of U34. The emulator transceiver U35 is controlled by ORing U36 chip select with the DEN* output of the emulator CPU.

Break Address Section

Another key requirement of the emulator is to provide the capability for the target system to break (halt) on specified memory addresses. In order to provide this feature the emulator must store the break address, monitor the target system address bus, detect a valid match, halt the target at before the address changes, and interrupt the emulator so it can handle the break condition. Additionally, the emulator

must allow the user to disable the break section if desired, and clear the break condition on command.

The break circuit consists of two primary hardware sections. First, the ability to store and monitor the address bus is provided by a set of ten 74F524 reqister/comparator chips (figure 2e). These devices are eight bits wide and can perform serial or parallel reads and writes. The parallel mode is used in this application, in order to allow direct a connection between the break circuits and the emulator CPU data bus. Since the address bus of the target is 11 bits wide two 74F524s are cascaded to form a 16 bit combination. The highest five bits are tied to ground and not used. The 74F524 retains the desired break address in an internal register and compares this address with the target address during each bus cycle. When a match is detected the equal (EQ) output goes low. This output is open collector and both cascaded EQ lines are tied together and connect to VCC through a 1k resistor. The clock input (CP) to the 74F524 is used for internal synchronization and is applied at pin 11. This signal must be provided by the CPU that is reading or writing to the chip. The data inputs which receive the desired break address from the emulator are also used to monitor the target address bus. Therefore, the address and clock lines are multiplexed using U42, U43, and U44 which are 74F157A chips (figure 2d). The control lines for this type of multiplexer was discussed earlier. The break addresses are label break_1 through break_5, but the actual chips are broken out as break_1A and

break_1B through break_5A and break_5B. This was necessary because the data bus is only eight bits wide and each break address requires two write addresses. During the address monitoring process all address bits are applied simutaneously to the comparators from the output of U29 (8282) and U30 (8282). In order for each comparator to be individually addressable, each chip required its own chip enable address. All chip selects are located in the emulator I/O address space and controlled by U38 (74154). Each comparator also requires combinational logic to drive its S0 (Pin 1) and S1 (Pin 19) inputs. Figures (2f-2h) shows specific connections for each control block, but each block performs according to the following truth table.

Table 3. Register/Comparator Truth Table

S0	S1	Operation
L	L	Hold - Retains data in shift register.
L	H	Read - Read contents in register onto data. bus.
H	L	Shift - Allows serial shifting on next clock
H	H	Load - Load data on bus into register

The second hardware section is the break detect interrupt circuits (figure 2k). This section is built around an 8259A (U58) interrupt controller chip. This device is produced by Intel and is compatible with the emulator CPU. The 8259A is software programmable and must be intialized by the CPU. The

specific software requirements are discussed in chapter 4. The data bus of the 8259A is connected directly the CPU. It uses the RD*, WR*, A0 lines to determine which internal register is being accessed. This chip handles a maximum of eight interrupts based on its programmed priority scheme. Each cascaded break comparator is connected to the interrupt controller through an invertor. When a break is detected the open collector output of the comparator is pulled low, inverted, and applied to U58 (8251A). The low to high transition causes the INT pin to go high. If the CPU's interrupt flag is set the high on its INTR line will initiate an interrupt sequence. The CPU responses by sending two interrupt acknowledge pulses on its INTA line. The 8259A will return a byte after the first interrupt acknowledge specifying which interrupt type has occurred. The second interrrupt acknowledge received is used by the 8259A for internal housekeeping. The CPU uses the interrupt type to determine the address of the break handling routine.

When any one of the break detectors outputs go low the target system must be halted. To accomplish this the output of the break detectors are ANDed together through U62A (7411) and U62B. The output of these gates is ORed with the break control flipflop output and fed through an open collector buffer (U61-B) to the target ready line. If the output of the break control flipflop is low when a break is detected the target system is halted by forcing its ready low. The break control flipflop is controlled by the emulator CPU by setting

and resetting U66-A (7474). To enable the break section an I/O operation is performed to address 00h to clear the break flipflop. To disable the flipflop and I/O operation is perform to address 10h, which sets the flipflop. When the flipflop is set its output is high, which makes U63-B (7432) go high. This prevents the open collector buffer from going low regardless of what the rest of the break detection section is doing. After a break operation has occurred the break detectors automatically clear themselves upon the arrival of the next target address. The CPU interrupt clearing process is handled with software and will be discuss in chapter 4.

Target 40 Pin Section

The address lines from the target plug MISC1 (12-19 and 21-23) are the output of the target CPU. The 8755 address lines are strobed by the ALE signal and normally do not require latching. Since the 8755 memory section is not being used, two 8282 latches are placed between the MISC1 lines and U10, U11, and U12 address inputs . Target address lines A0 through A7 flow through U29 (8282) and A8 through A10 flow through U30 (8282) (figure 2i). Both latches are strobed by the target CPU ALE (MISC1-11) at the beginning of each cycle. Memory access by the target system is similar to I/O circuit access. When the target system reads memory the control signals are not directed to the 8755, but instead are fed to the user memory chip U18 (UM6116). To allow access by the

target or the emulator multiplexers U10 - U12 (74F157) are used as described in the user memory section. The output enable line is multiplexed through U31, using the read line that also drives the 8755 read input. This provides an accurate control arrangement because the mode flipflop ensures that only one set of control lines are applied to the user memory and 8755 control lines. To ensure that the target system cannot write to the user static ram, the chip select CE1 is ORed with the IO/M* signal before being applied to the multiplexer input. This controlled chip select is only active when the target is performing a memory read cycle. Since the IOW* is high during a read cycle this ensures that the ram write enable is at its required inactive state during each read cycle. The write enable signal must be low when the emulator performs a memory write cycle, therefore the WR* control from U8 (8088) must have access to the ram device. When the mode flipflop is set to allow emulator access, the emulator may read and write the user ram or the parallel ports of the 8755. To prevent bus contention when the target system accesses the ram while the emulator is performing other operations, separate tranceivers were required at the data output of the ram device. The data direction control input to U13 (74F245) is tied to ground so that data may not be moved from the target system to the ram chip. This prevents anyone from maliciously performing a memory write from the target system. The output enable of U13 (74F245) is

controlled by the target read line, which is qualified by the memory chip select described earlier. The normal memory access time of the 8755 is 450 ns and 300 ns for a 8755-2. The 8755-2 is used in the emulator design to ensure that the user may use any currently available version of the 8755 in a circuit design. The control signals and data from the target CPU must flow through a number of support devices for either a memory or I/O operation. For a memory operation U29 and U30 (8282) introduce a 30 ns delay, U10 - U12 (74F157) introduce a 6.5 ns, and U13 (74F245) introduces a 7.5 ns delay. Since U18 is a 150 ns access time static ram, the total memory access time is a maximum of 194 ns. Therefore, the additional hardware is transparent to the target system. During an I/O operation to U36 (8755-2) the additional hardware includes U31 - U33 (74F157) with 6.5 ns delay, and U34 (74F245) with 7.5 ns delay. The 8282 U29 and U30 delay is not encountered during I/O operations, because U36 only requires the A0 address line and this is taken directly off of the target plug.

Special Function Section

The final requirement of the emulator is to provide the user with the ability to single-step the target from the host computer. This process required both hardware and software support. The single-step ability of the emulator is limited to either a step mode or a free-run mode. If the user desires to single-step from a specific location in memory this must

be done by first breaking at that address and then entering the single-step mode.

The target single-step mode is entered when the CPU performs a memory operation to address 40000, thus clearing the single-step mode flipflop U65-B (figure 21). By placing a low on the Q output of U65-B and ORing this with the output of the stepper flipflop U65-A the target is halted after each bus cycle. The stepper flipflop is stepped by an I/O operation at address 20h by the emulator CPU. This signal is used to clock U66-A (7474), which then pulses U63-A (7432). If the single-step mode flipflop output is low, this signal will force the target ready line high. The ready line will remain high until the next target ALE signal goes high. The ALE signal is inverted and applied to the clear input of U65A, which drives the stepper output low. When the stepper output goes low this forces the target ready line low and halts the target CPU. To put the target system in the free-run mode, the emulator CPU performs an memory operation to location 30000. This sets flipflop U65-B and prevents the stepper flopflip from driving the target ready line low.

An additional single-step circuit was built to allow the emulator CPU to be single-stepped (figure 21). This circuit is not used in the normal operation of the emulator, but is provided for the convenience of maintenance to the system. The operation of this stepper is similar to that described for the target. However, the clock pulse for this stepper is generated by switch S2. When S2 is depressed the pulse is

debounced and used to trigger a one-shot multivibrator U45 (74121). Each pulse from S2 produces a positive going squarewave at the output of U45, which then clocks the stepper flipflop U39-A. The stepper flipflop is cleared each bus cycle by the ALE signal. The output of U39-A is Ored with the output of switch S3. If S3 is set to ground the CPU is controlled by the stepper flipflop and halts after each cycle. If S3 is set to VCC the stepper flipflop has no effect since the output of U41-D (7432) is held high. The output of U41-D is fed to the clock generator (U9), synchronized, and applied to the CPU.

The emulator is built on two perforated project boards as which are connected by a 40 pin ribbon cable. The pin connections at both ends of the cable, MISC2 and MISC3, are listed in appendix A. Additionally, table 1 of appendix A lists the parts used to build the emulator. The net wiring list is shown in appendix B.

IV. THEORY OF OPERATION: SOFTWARE

INTRODUCTION

One of the primary enhancement features of the emulator when compared to a standard 8755 is its capability to be controlled from a host computer microcomputer. The hardware to provide this ability was discussed in the last chapter. In this chapter the software to control the emulator system is discussed in detail.

The software necessary to control the emulator may be broken into three separate sections. The first section to be discussed is the emulator bootup routine. The software is situated in the emulator address space such that it is the first code encountered upon system startup. The bootup routine has the responsibility to initialize the emulator. This is accomplished through a series of read and write operations to the proper pieces of hardware. The bootup routine also performs a self-test on its static ram prior to making contact with the host computer. Likewise, the user static ram is tested and any problems reported to the host computer. Finally, the bootup routine must download the main emulator control software and verify to the host computer that the program was received accurately.

The second section of software is the main emulator control software. This program is initially located on the host computer, but is downloaded to the emulator. This

section is responsible for all emulator activity after the initial bootup routine has terminated. The emulator code contains all the flag and message smybols used between the emulator and the host computer. It also contains the software necessary to change any of the hardware originally configured by the bootup routine. The emulator does not actually alter any subsystems without receiving instructions from the host computer. Once the emulator is downloaded it is passed control by the bootup routine. The emulator then awaits further instruction from the user via the host computer.

The third section of software is the host computer emulator control software. This code resides on the host computer and is only usable by the host computer. This program contains all the flags used by the emulator and uses them to pass desired action commands to emulator. Each transaction between the emulator and host computer is preceded by an attention signal, contains a positive or negative acknowledge, and ends with the EOT signal. If the acknowledge message is negative the appropriate error message is display to the user by the host computer.

Each software program has a corresponding flowchart in appendix C and these should be used by the reader during the discussion which follows. The source code listing for each section is in appendix D. The source code contains explanatory comments to aid the reader in interpreting the assembly language code.

Bootup Software

Upon startup the CPU outputs address FFFF0 (Table 2) as described in the hardware discussion. This address equates to 707F0 in the emulator address space. At this location the emulator branches to address 70000 and begins its initialization process (figure 3).

The first task accomplished is to halt the target system to ensure that no interference is created between the two CPUs. This is accomplished by performing a memory write cycle to address 40000, which puts the target in its single-step mode. Here the single-step mode is equivalent to a halt, because the target system cannot advance without a step pulse, which must come from the emulator. The second task to be accomplished is to put the break circuits into their off state. This is accomplished by performing an I/O write cycle to address 0030, which masks out the state of the break detector circuits. This is done to ensure that an erroneous break detect does not occur before the break registers are properly configured. The break section remains disabled until changed by the main emulator program. At this point the target circuits are isolated from the emulator.

The bootup routine now begans to initialize itself for operations with the host computer. The serial port is programmed for asynchronous communication by writing three consecutive zero bytes to put the UART in its worst case

configuration (2). This causes the UART to be set for synchronous operation with two zero bytes for SYNC characters. At this point the UART is reset by writing 40h to the control register. Next, the mode register is loaded with 5Eh, which configures the UART for asynchronous transmission, one stop bit, odd parity, eight data bits, and a baud rate factor of 16. The baud rate factor is used to determine the required clock frequency as:

Clock frequency = baud rate factor X baud rate (1)

For the emulator the baud rate is 9600 baud, therefore the UART clock frequency is 153.6 Khz. This clock is only used to clock the serial shift registers in the UART. The next byte written to UART goes into the control register. The receiver and transmitter are enabled, error flags cleared, and the data terminal ready pin activated by outputting 37h. At this point the UART may be used to communicate with the host computer.

Before sending or receiving data through the UART its status register must be read and analyzed. Before each transmit operation the status register is read and compared with 01h to see if the transmit register is empty. Prior to each read operation the status register is compared with 02h two see if a new data byte has been received. Bits three four, and five indicate whether a framing, overrun, or parity error has occurred. These flags are used with any transmit or

receive operation to ensure accurate data transfer. For file transfers, proper transmission and receiption of data is also verified by using a checksum counter. The checksum value follows the end-of-file (EOT) during each communication between the emulator and host computer.

Following the initialization of the UART, the static ram on page zero is tested. To test the ram alternate 00h and FFh bytes are written to all 64 kbytes. At the conclusion of the write cycles each location is read and tested to see if the proper data is there. If an error is found an error counter is incremented and the NACK signal sent to the host computer at the conclusion of the test. If all is well this is sent to the host computer and an acknowledge reply is returned. Once the emulator memory has been tested the user static ram is scanned in the same manner and the results reported to the host computer. If problems are reported the host computer is responsible for advising the user. The bootup routine must have three failures of a given memory section before an error message is sent. At this point the emulator should be reset and the bootup routine run again.

When the all clear is sent to the host computer this is used as a signal to download the emulator program. The starting address for storage is set at 00000 and the emulator will loop until the host computer responds with the download signal. The next byte to be received by the emulator is stored at the first memory location. All subsequent bytes are

stored sequentially until the EOT signal is detected. Each byte received must be tested and also added to the checksum register. When the EOT is found the next byte is compared with the checksum value an if a match is found an acknowledge signal is sent to the host computer. If the checksums do not match the host computer will retransmit the emulator, in response to a NACK signal, and the process is repeated. After three attempts to pass the emulator memory the host computer will display an error message and the emulator resumes testing its memory. If the acknowledge signal was returned the emulator passes control to address 00400 and the bootup routine is terminated.

Main Emulator Software

The emulator routine starts by waiting in a loop for the host computer to communicate the desires of the user. The host computer provides a series of menus and prompts to communicate with the user. The emulator operates in a passive mode, except that it will signal the host computer if a break condition occurs.

The emulator software provides four basic functions which may be requested by the host computer. Figure 4 shows the basic flow chart arrangement of the emulator code. The emulator waits for the attention signal (++) followed by an ascii character. If a 'U', ascii 55h, is received the user memory procedure is activated and controls the interaction between the host computer and the user memory. If a 'S',

ascii 53h, is received the single-step control procedure is activated. If a 'B', ascii 42h, is received the break control procedure is activated. Finally, if a 'P', ascii 50h, is received the 8755 control procedure is activated. Each procedure returns control to the wait loop upon termination.

User Memory Control

The user memory procedure allows the user to download the target program to U18, write to a particular location, or read a particular location. All user memory functions must be initiated by and coordinated with the host computer.

When the emulator loop receives a (++U) string the user memory procedure is called (figure 5). Upon activation the procedure returns a ready signal to the host computer. The host computer returns a string which contains a direction flag and the starting address. The emulator stores this string in the user instruction buffer. Next, this string is analyzed to determine what actions to take. If the direction flag is for a download the emulator clears the transmit attempt counter and the checksum register. The emulator then sends the ready signal to the host computer. At this time the host computer begins to download the data. Each byte received is stored in a temporary data buffer until the entire file is verified at the end of the transfer. With each byte received the value of the data is added to the checksum register. This process is continued until the EOT is received. When the EOT

is received the next byte received is interpreted to be the checksum value calculated by the host computer. This value is compared to the emulators checksum and if they match the host computer is sent the acknowledge signal. The emulator then stops the target sytem and changes the user ram circuits to the emulator mode. The data received is transferred to the user ram and the target restarted. If the checksums do not match the NACK signal is returned and the transmit counter is incremented. The transmit counter is compared to three, if the count exceeds three the user procedure is exited. If the transmit counter is less than three the download is repeated. The emulator does not send a retransmit signal to the host computer, because the host computer keeps a separate count of its transmission attempts.

If the string stored after the emulator returns its ready signal is an upload the emulator will follow a course opposite to the one just described. The start address and number of bytes to upload are received from the host computer. The emulator clears its transmit counter and checksum register. Next, the target is halted and the data loaded into the temporary buffer area. The purpose of the buffer is to keep the data on hand for a repeat transmission if an error occurs during the upload. The data is held in the buffer until over written by future transfer operations. After the data is in the data buffer the target is returned to its previous status. The emulator sends the acknowledge

signal to the host computer to indictate that the data is next. Each byte is sent to the host computer and the checksum is tabulated until the proper number of bytes have been uploaded. When the byte limit is reached the emulator will send the EOT, which is then followed by the checksum value. The emulator then waits until an ACK or NACK is received. If the ACK is received the user procedure is exited. If the NACK is received the emulator will increment the transmit counter. If the flag now exceeds three the procedure is exited. The emulator does not handle this error condition since the host computer handles all errors. If the transmit count is less than three the ACK is sent and the data retransmitted. This process is repeated until the data is accurately transferred or the transmit count exceeds three.

Emulator Mode Control

The emulator mode control procedure is not explicitly called by a host computer command. This routine is accessible only by the emulator, but provides service to various host computer commands. For example, the user memory operation uses this procedure during the its memory reads and writes to disable the target section. This procedure simply uses the mode flag to either put the emulator multiplexers in the target or emulator mode. If the target mode is desired a memory 20001 write is executed. Conversely, if the emulator mode is desired the memory write goes to 20000. In either case the mode flag is set to the proper status and the

procedure is exited, with control returning to the calling procedure.

Single-step Control

The single-step procedure is called when the emulator waiting loop receives the (++S) string (figure 6). The procedure responds by sending the ready signal. The host computer then sends the desired action code, which may be to enable the stepper, to put the target in the free-run mode, or to step the target. If the enable code is received the emulator performs a memory write to address 40000, updates the single-step flag, sends the ACK signal, and returns to the waiting loop. If the free-run code is received the emulator performs a memory write to address 30000, updates the single-step flag, sends the ACK signal, and returns to the waiting loop. If the action code is to step the target circuit the emulator checks the single-step status to ensure that the stepper is in the step mode. If the mode is set to step the emulator writes to I/O address 0020, returns the ACK signal, and then returns to the waiting loop. If the mode is set to free-run the emulator returns the NACK signal and exits the procedure.

Break Control

The emulator break circuits are controlled by a combination of the bootup routine and this emulator procedure. The bootup routine took care of initializing the

8259A and inhibiting the break circuits. This procedure is used to load the break registers, enable or disable the break circuits.

When the emulator waiting loop receives the (++B) string it calls this procedure (figure 7). A ready signal is immediately returned to the host computer. The host computer then sends a code to enable, disable, or download to the break registers. If the action desired is to enable the break detection circuits the emulator writes to I/O address 0000. The break flag is updated and then the ACK signal is returned. If the action desired is to disable the break circuits the emulator will write to I/O address 0010. The break status flag will then be updated and the procedure exited.

If the user desires to load the break registers the download signal is used. Following the ready signal to the host computer the emulator clears its checksum register and sends another ready signal. The data bytes received will be placed in the break address buffer until the EOT is detected. Once this flag is received the next byte will be compared to the emulator checksum value. If the checksums match the break circuits are disabled, the break register are loaded, and the break circuits then enabled. Next, the break status flag is updated and the ACK signal returned to the host computer. The procedure then returns to the waiting loop. If the checksums

do not match the NACK signal is returned and the procedure is exited.

When the 8259A receives an interrupt from the break detectors it must direct the response of the CPU. Regardless of which break address is detected the 8259A will respond with an interrupt request to the emulator CPU. The CPU responds by sending two interrupt acknowledge signals, which causes the 8259 to output the interrupt type number. The type ranges from 8 through 12 and is calculated by the 8259A based on the setting of its IR0-IR2 bits (figure 8). These bits are combined with 00001b to indicated the proper type number. Each interrupt type is used to vector to the interrupt handler code address. Each interrupt handler simply moves its type number to the interrupt hold register. The interrupt hold register is simply a memory location used for parameter passing. Each interrupt handler then passes control to the break reporting procedure. This procedure sends the (++B) string and waits for the ACK signal. When the ACK is received the emulator sends the interrupt number in the interrupt hold register. The emulator will wait for instructions from the emulator before resetting the break circuits. If the host computer indicates that the break procedure should be left as is the circuits are disabled and the status flag updated. If the reset command is received the break circuit is disabled and then enabled to reset them. The the enable/disable sequence is required to allow the break address comparators

to overwrite the last address. With either of the above actions the break loading procedure is exited.

8755 I/O Control

The emulator initiates a call to the 8755 control procedure upon receiving the (++P) control signal. This procedure is used to read or write to the target circuit ports and to reconfigure the bit assignments (figure 9). The procedure responds to its call by returning the ready signal to the host computer. Next, the host computer sends the direction flag followed by a port number. If the direction flag indicates an upload, the port number will be to port A or B only. In either case the emulator will hold this infomation in a temporary register. The target system is halted and the mode changed to the emulator. The selected port is read, the data is moved to the temporary register, and the target restarted. The emulator clears its checksum and the transmit counter, then sends the ready signal. The data byte is then uploaded, followed by the EOT and checksum. The emulator then waits for the ACK from the host computer. If the ACK is received the procedure is exited. In the event the NACK is received the transmit counter is increment and compared to three. If the count does not exceed three the transmission process is repeated. If the count does exceed three the procedure is simply exited. When the direction flag indicates a download it is possible for the data to go to the ports or to the data direction registers. The host computer

sends the port number after receiving the ready signal. The emulator halts the target, clears its checksum, and clears the transmit counter. The emulator sends the ready signal again to indicate it is ready for the data byte. The data is received and held until after the checksum is verified. After verification the data is loaded into the appropriate register and the ACK signal is returned. The emulator then restarts the target and returns control to the waiting loop. If the checksum is not correct the emulator performs the usual three tries then terminates.

Host Computer Control Software

The host computer program serves as the interface between the emulator and the user. Most of the actions taken by the host computer will be initiated by the user. However, there are two cases when this is not true. When the system is started the emulator memory is downloaded without user intervention. The other case is in the event of a break address detection. In general the user is prompted for an input and the response is then transmitted to the emulator. The flowchart for this portion of the system programs is located in figure 10.

Upon startup the host computer program displays a welcome message and advises the user to standby. The program then monitors the serial port for two ACK signals from the emulator. One ACK is for the emulator SRAM and the other is

for the target SRAM. If either signal is replaced by the NACK signal the host computer displays an error message and the user should reboot.

When the ACK signals are received the emulator clears its transmit and checksum counters. The host computer sends the download signal, followed by the emulator program. The emulator program is then verified using the checksum method. If the program receives the NACK instead of the ACK, the transmit counter is incremented and the emulator code retransmitted. This is repeated up to three times before an error condition is assumed. If an error condition does turn out to be the case the program displays an error message advising the user to reset the emulator and host computer program. If the emulator program is successfully downloaded the wait prompt is removed and the setup menu is displayed. The display will also contain the status of the single-step, system mode, and break circuits. The single-step will be in the step mode, the system mode set to system, and the break circuits disabled. This is the initial system configuration. These statuses will be displayed anytime the setup or main menus are displayed. They are also displayed individually when their corresponding submenus are displayed.

The user must address each section of the setup menu that is intended to be used. Generally, all three areas should be setup, but this is not required if only partial operations are planned. In any case the program waits for the user

selection and then branches to the proper procedure. Each procedure will return control to the setup menu and the user will eventually need to select item (d) to escape the setup menu. Selections a, b, and c each handle a specific emulator feature and their operation are discussed next.

Pass User Memory

When item (a) is selected the user memory module is called (figure 11). The program sets a flag which indicates which menu was currently in used prior to the call. This flag will be used later to ensure the user display contains the same menu when the module is exited. The user is then given the following set of options.

User Memory Menu:

a. Down load target memory

b. View target memory location

c. Edit target memory location

d. Exit

Which offers the choice of downloading the target memory, viewing a particular memory location, or edit a target location. When the download target option is chosen the user is prompted for the filename, starting and ending addresses. The starting address is use to inform the emulator of which address in the user ram to load the program. The ending address is used to determine when the EOT should be sent to the emulator, but this address is never sent to the emulator.

After receiving this information the procedure uses MS-DOS function calls to open and read the user program into a temporary 2 kbyte buffer. This buffer is maintain at all times after the initial data read. The user must have this buffer loaded before option (b) may be accurately employed. The reason for this restraint is that if the view option is selected the user is shown what is in the temporary user buffer and not what is in the user memory chip. Once the buffer is loaded the program will clear its transmit counter and checksum register. The emulator is sent the (++U) signal and the program then waits on the ready response. The emulator responds with the ready signal and then awaits the direction flag and starting address. When this information is received the emulator sends another ready and the download is accomplished. When the byte count has been reached the program inserts the EOT, and then sends the checksum value. If all went well the emulator will return the ACK signal and the transfer is terminated. In this case the calling menu is displayed. If the NACK signal is received the download is attemped two more times. If the download still has not succeeded the user is informed via an error message display. At this point the entire system should be reset an the download attempted again.

The view target memory option is provided so that the user can look at changes which may have been made to the user memory. It is not designed to alter the user memory since

this requires the target to be halted. If the user does desire to alter the user memory the edit option should be used. The best way for the user to keep track of all changes is to keep the code listing available and to annotate all changes. When the viewing option is selected the user is again prompted for the starting address within the user memory (buffer) area and ending address. The user may view a maximum of 20 consecutive locations at a time. The information will be displayed by location and content. The edit option allows the user to change the data at a given location within the target memory. The maximum number of locations which may be edited is ten. This number is arbitrary, but should be adequate for testing purposes. If the user wishes to do extensive editing the most efficient method is to make the changes at the assembly level and reload the user program. When the edit option is selected the user is prompted for the starting address and byte count. The program setups up the temporary edit buffer and clears its byte counter. The user is then prompted for the new data, which must be entered in consecutive locations. If more than five bytes are attempted the user is informed of the error and the calling menu is displayed. In order for the user to accomplish additional editing the same sequence of steps just described must be accomplished. After the user has entered the new data the carriage return is require to download the data. The download procedure is the same from here on as in

the case of a total user memory download. The temporary user buffer is updated with these changes and the procedure returns control to the calling menu. If the user is performing the original user download it is suggested that the break circuits are configured next. This procedure is discussed in the next section.

Break Control Module

If the user selects item (b) from the setup menu or items (b) or (c) of the main menu, the break control procedure is entered (figure 12). The procedure displays the following break menu.

Break Control Menu:

a. Enable break detection

b. Disable break detection

c. View break addresses

d. Change break addresses

e. Exit break menu

If the user is still setting up the emulator then item (d) should be selected. This is the option to choose to edit the the break registers at any time. When item (d) is selected the user is prompted for the register number, which ranges from 1 to 5. When the desired register is entered the user is prompted for the break address. The break registers consist of two separate registers, but the user may simply enter a three character hex address. This address will be

separated by the program and stored in the break buffer area. All break addresses must fall within the legal memory space of the target ram. These addresses range from XX7FF to XX000 from the target perspective. After each address is entered the user is asked if further address changes are desired. If the reply is yes the process is repeated as many times as the user desires. If the reply is no the emulator is sent the (++B) signal and the normal download process is carried out. The break control menu is displayed and the user may select another option or exit this menu. If the break register download was unsuccessful the user is informed and the break menu exited. The user may try again from the original calling menu. Whenever the break menu is displayed the current status of the break circuit on the emulator is automatically displayed and any changes to the status are automatically updated. To enable the break circuits to monitor the target, the user simply selects item (a) from the break menu. The user must ensure that the break registers have be previously set because the emulator will break on any address match it detects. To disable the break circuits the user simply selects item (b) and the rest is handled automatically. For the enable or disable option the host computer sends the emulator the break control signal, which is followed by the desired change flag. If the user desires to view the current break register contents then option (c) is entered and the local break addresses are displayed. The user should always

note the current status of the break circuits before sending changes to the emulator. There final option on the setup menu is to configure the parallel I/O ports. This setup is not necessary for normal operations, since the user program will do this. The option is provide to give the user added flexiblility and is discussed in the next section.

Parallel Port Control

The user of a normal 8755 I/O chip often uses the device for its flexible ports, rather than for the memory it contains. To allow the user full flexibility during circuit testing, the emulator allows all of the original 8755 capabilities and also permits altering the target ports while the target CPU is in a halt state. This feature will be useful for pulsing circuits connected to the target, reading the data at the ports after execution of an instruction, and changing the data direction registers.

To access the parallel port procedure the user may select item (c) of the setup menu or item (d) from the main menu. When activated the procedure displays the port menu:

Port Menu:

a. Read port A

b. Read port B

c. Write port A

d. Write port B

e. View/Reconfigure port A DDR

f. View/Reconfigure port B DDR

g. Exit port control memu

If the user selects items (a) or (b) the host computer sends the (++P) signal and awaits the ready signal from the emulator (figure 13). When the ready is received the port number corresponding to the user selection is sent. The emulator responds with a byte of data retrieved from the 8755 port. The host computer then sends the ACK signal and displays the port number and data to the user. If item (c), (d), (e), or (f) is selected the user is prompted for the data to send to the emulator. The data is placed in a temporary buffer and the emulator is sent the port operation flag. The emulator returns the ready flag and waits on the direction flag and returns the ready signal. The host computer now sends the data and port number. When the write operation is complete the emulator returns the ACK and the transaction is complete. The port menu will be displayed and the user may enter the next selection. If item (g) is selected the user is returned to the calling menu. After the setup process is initially completed the user does not need to return to this menu. The main menu has access to all the submenus discussed thus far and this is where normal operations are initiated.

Main Menu Control

The main menu is entered from the setup menu and may control the entire emulator during a session. The reason for the setup menu was simply to ensure the user remembers to go through the setup procedure. The main menu uses the same procedures as the setup menu, but the user may not go from the main menu to setup menu. The main menu maintains the status of the target (system/target) the break detectors (enabled/disabled), and the stepper circuit (step/free-run).

When the system is brought up the target is halted by the bootup routine and the break circuits are configured during the host computer initializtion of the target. Both should be left disabled until after the setup operation is complete. Therefore, when the main menu is initially displayed the user should enable the break detects before enabling the target. This ensures that the target is still at its starting address when the break monitoring begans. After the initial start the breaks may be enabled or disabled at any time, but the user will have to keep track of the target address. One way to do this is to enter the single step mode before enabling the break detectors. Other than these precautions the user may select from the main menu freely.

Only options (a), (b), and (d) have not been discussed from the host computer perspective. Option (a) allows the user to start or stop the target at any time (figure 14). To do this the user is prompted for the desired action after (a)

is entered. The command is sent to the emulator and executed. The emulator returns the ACK signal to verify the completion of the requested action. The host computer informs the user of the completion, updates target status flag, and returns to the main menu. If item (b) is entered the user is again prompted for the desired mode and the command is carried out. If item (d) is entered the process is a bit more involved.

In response to the single step selection the host computer displays the single-step menu:

Single-step Menu:

a. Start single-step mode

b. Step target

c. Exit single-step mode

d. Main menu

From this menu the user may enable or disable the stepper circuits, single-step the target if the stepper is enabled, or return to the main menu. The user should note the status of the target before attempting to single step it. If the user selects item (a) or (c) the host computer checks the status flag and if it is the opposite of the requested mode the command is executed. If the requested status already exists the program simply displays the single-step menu. If the user desires to step the target, the mode should be checked an set to "step" if not already in that mode. If the user fails to do this then an error message will be displayed

and the single-step menu displayed after a brief delay. If the target is in the step mode, a request to step will be executed on command. The host computer sends the (++S) flag and receives the ready reply. The step flag is then sent to the emulator, which executes one step and returns the ACK signal for verification. After each step the host computer informs the user and loops back to the single-step menu. The user may select item (d) if other options on the main menu are desired while in the single-step mode. The other options may then be executed and the main menu used to get back to the single-step menu. For example, if the user has entered the single step mode and now desires to disable the break circuit the following action is require. Enter (d) on the single-step menu, select (b) on the main menu, then select (b) on break menu, select (e) on the break menu, and (d) on the main menu to return to the single-step menu. This type of process is necessary if something besides the single-step options is desired while in the single-step mode. Normally, the user would only need to enter (b) to step the target from the single-step mode. Once the user leaves the single-step mode the single-step menu is removed and control is returned to the main menu. From the main menu all emulator features may again be accessed. Figure 15 maintains the current status of the mode flag and informs the emulator of any desired change to the emulator's mode setting.

V. Recommendations

The 8755 emulator provides the user with several features which are not available with the standard 8755 device. These features allow the user of the emulator to test and troubleshoot circuits more efficiently. The emulator's usefulness could be increased still further with some additional changes.

One aspect of using the emulator which needs to be improved involves its hardware setup. This involves making connections to several external devices. The +5 VDC, +12 VDC, and -12VDC power supplies should be built into the emulator to increase the mobility of the device. The 5 volt supply requires 4.2 amperes, which has required connecting two power supplies in parallel in the past. This could be annoying to students moving the emulator from one work station to another. The emulator currently does not need forced air cooling, but this is recommended if a built-in power supply is added.

The UART's transmit and receive clock signals are also provided by an external source. These signals are currently running at the same frequency and are tied together inside the emulator. For convenience this signal could be derived from an onboard oscillator or clock chip. For still greater flexibility the receive and transmit clocks should be independent of each other. These changes to the serial port section would allow the emulator to connect to a wider variety of devices. The emulator hardware and software is not

Z-100 dependent and will work with any device which sends the proper flags and messages. One limitation of the serial port is the requirement for asynchronous communications. The UART used is capable of synchronous operations, but this will require modification to the serial port initialization routine and minor hardware changes.

The number of break addresses which may be monitored simultaneously is currently five. While this is a limited number of addresses, it is not recommend that additional address monitors be added. Each break address requires the addition of two register/comparators and ten logic gates. Additionally, if more than three address monitors are added an additional interrupt handler is required. Therefore, if a significant number of new break address handlers are desired the user should considered a different approach.

Finally, future units of the emulator should use printed circuit boards instead of the wire wrapping method. The emulator uses a large number of microchips and the density of the interconnect wiring is high. This caused crosstalk problems during the design of the emulator. With a well designed printed circuit board this problem would be less noticeable and hence the emulator more reliable.

Appendix A: Hardware Diagram

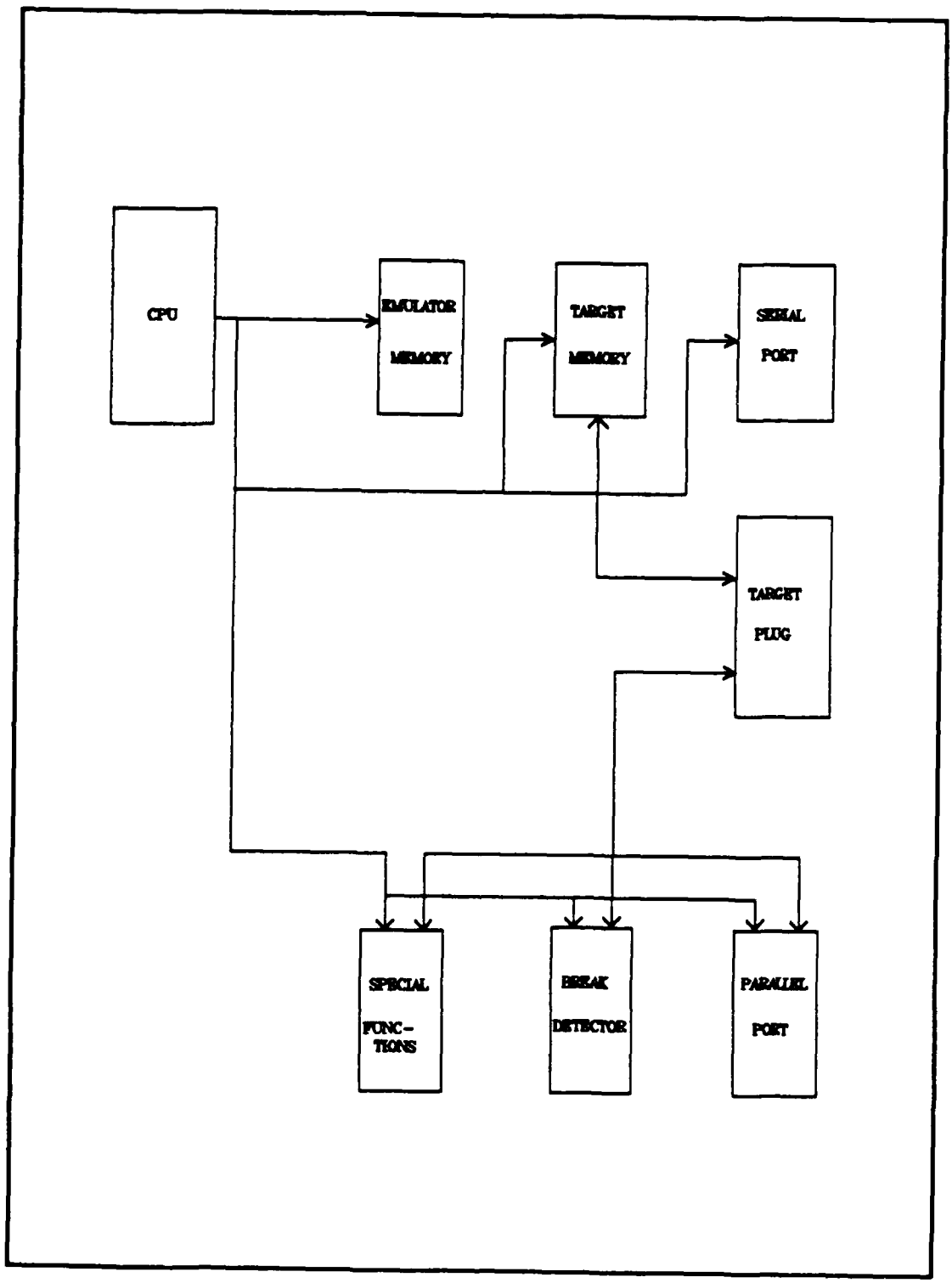

Figure 1. Emulator Block Diagram

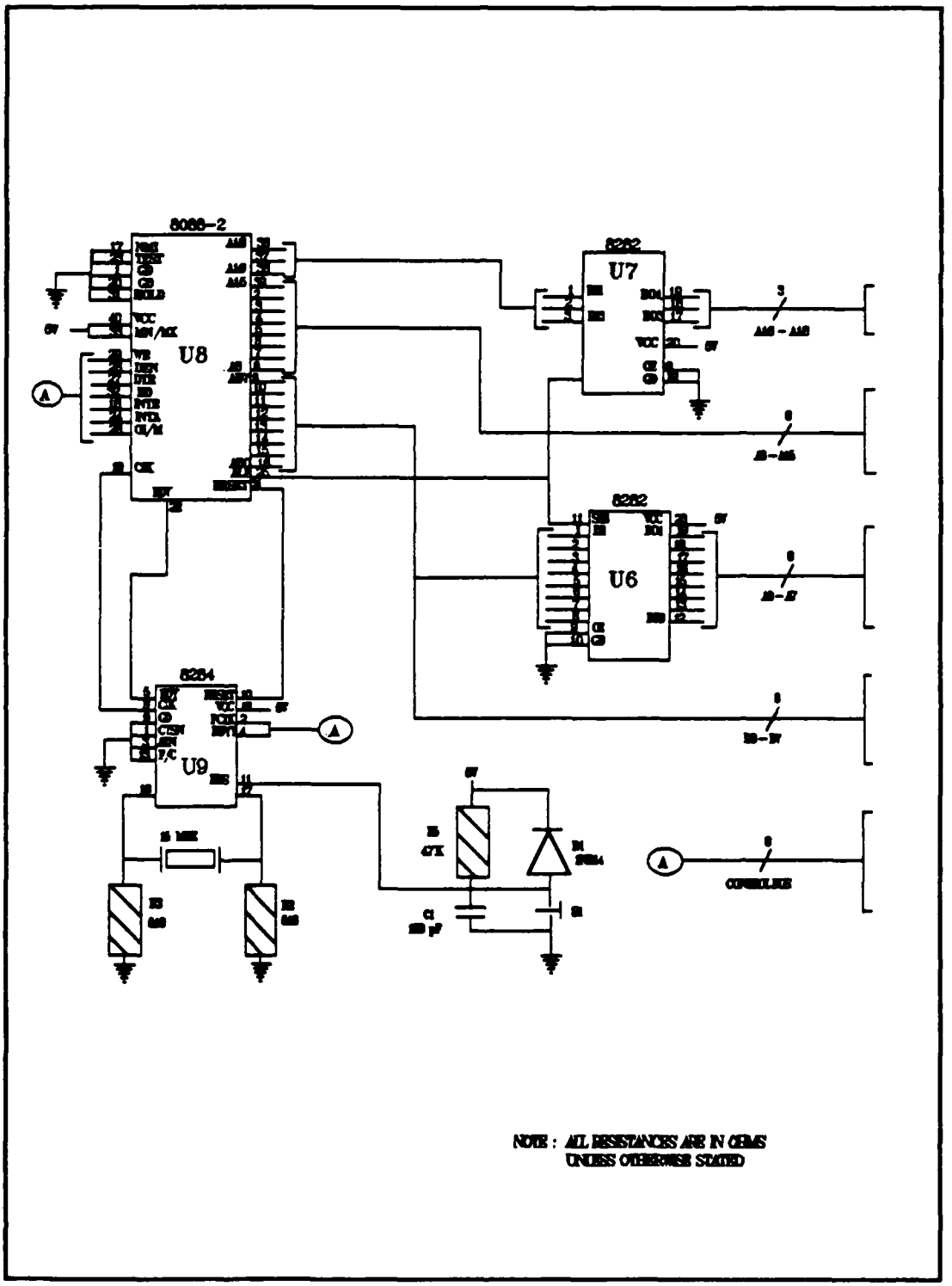

Figure 2a. Schematic Diagram

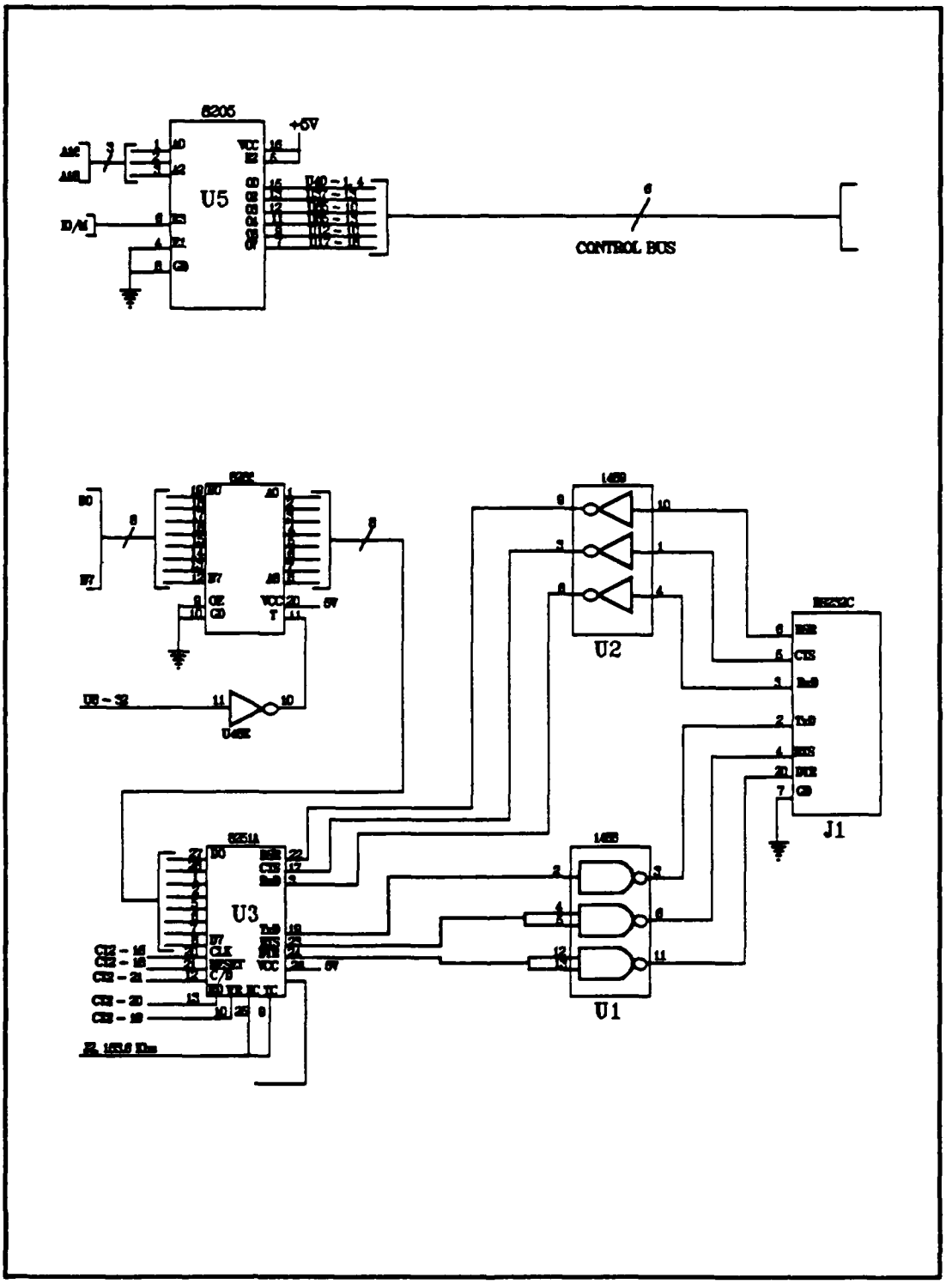

Figure 2b. Schematic Diagram

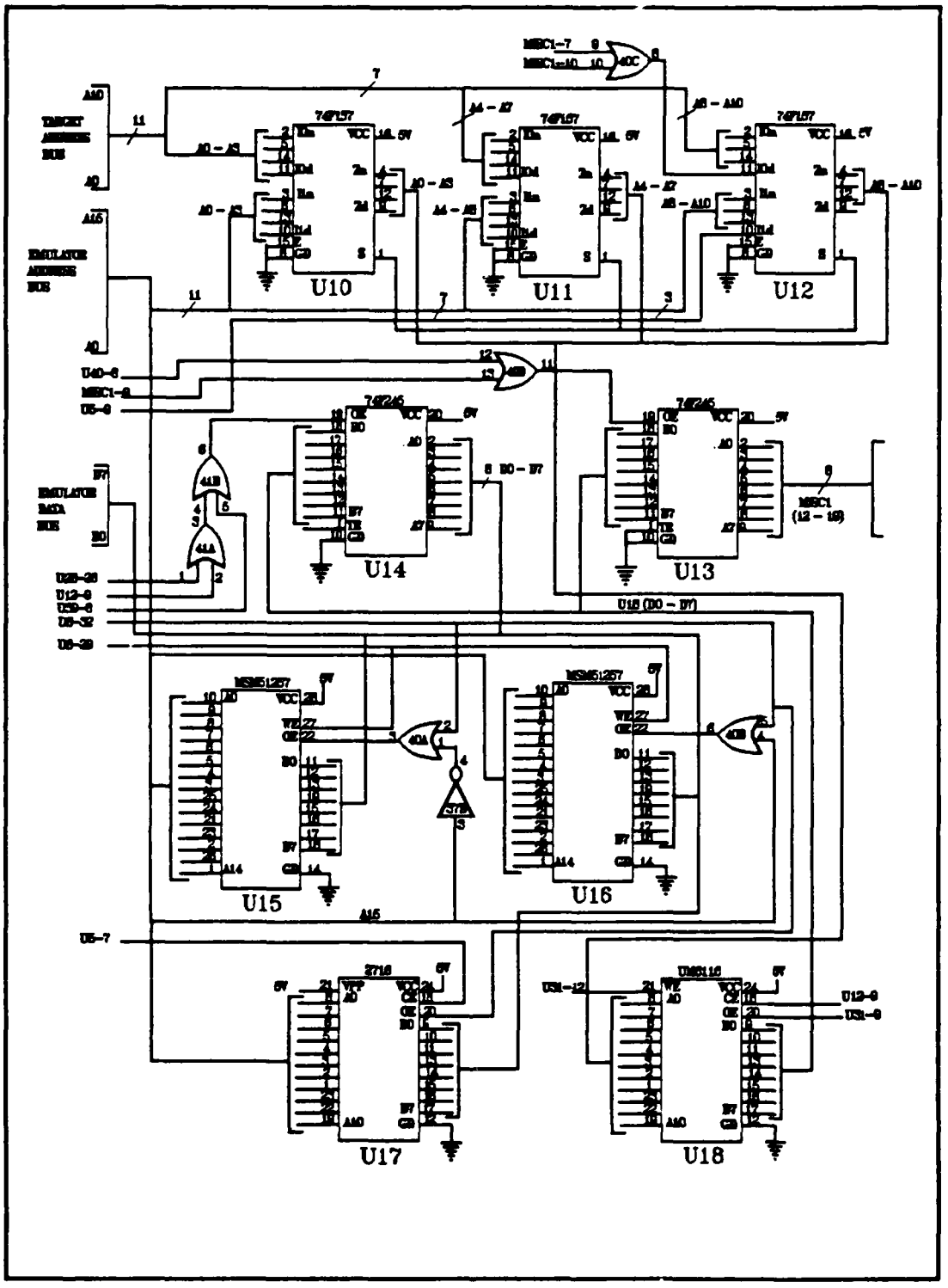

Figure 2c. Schematic Diagram

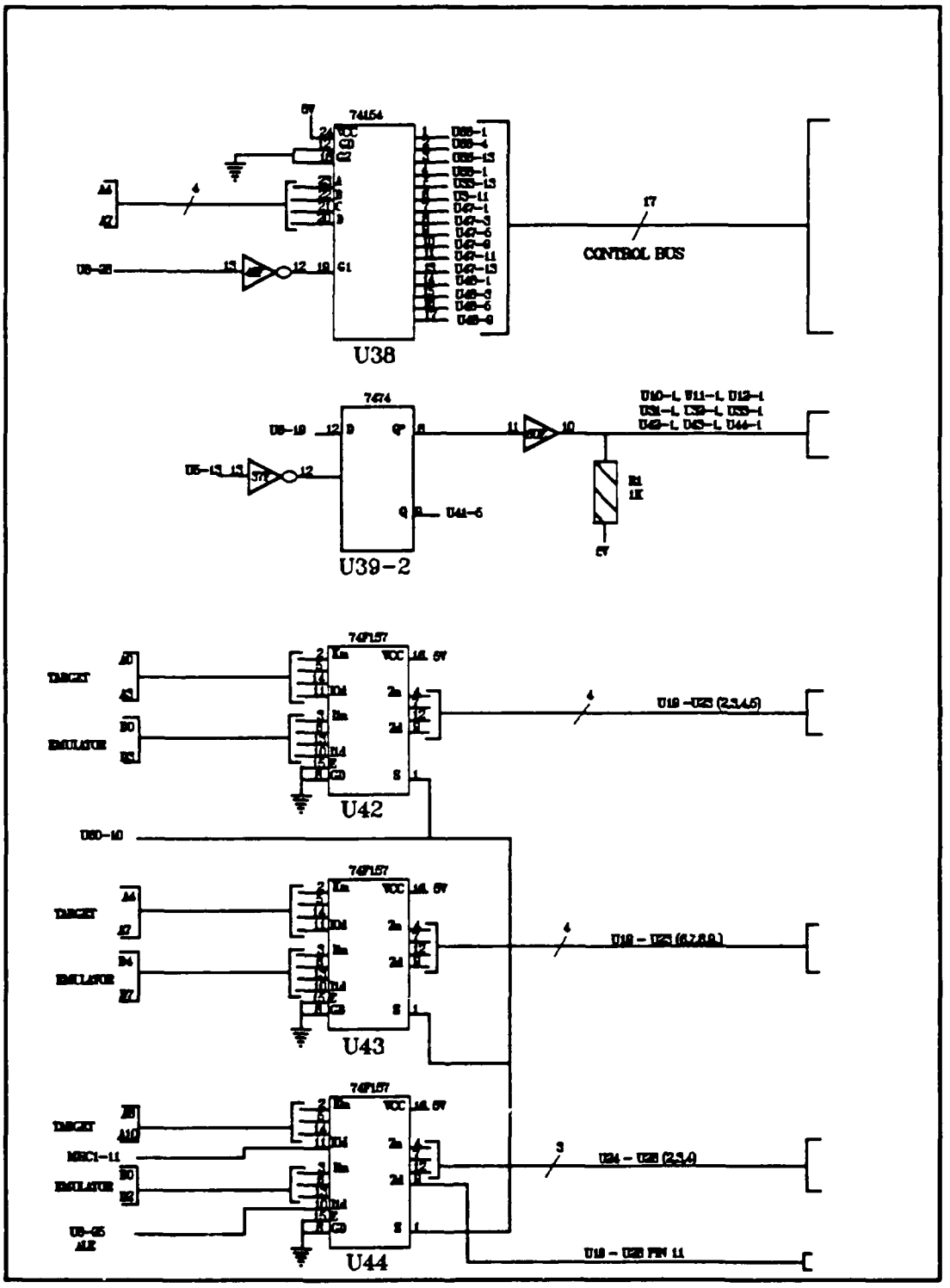

Figure 2d. Schematic Diagram

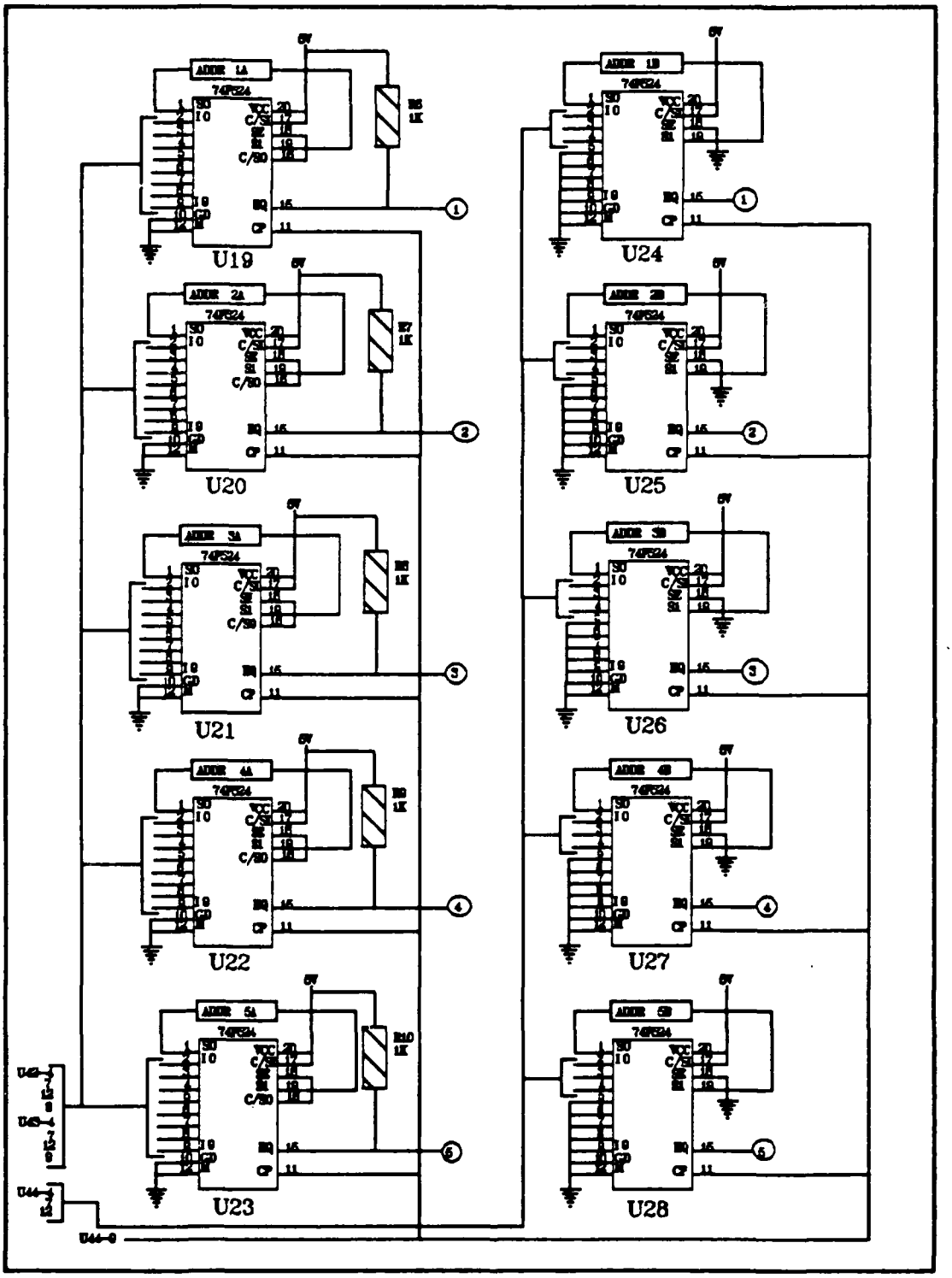

Figure 2e. Schematic Diagram

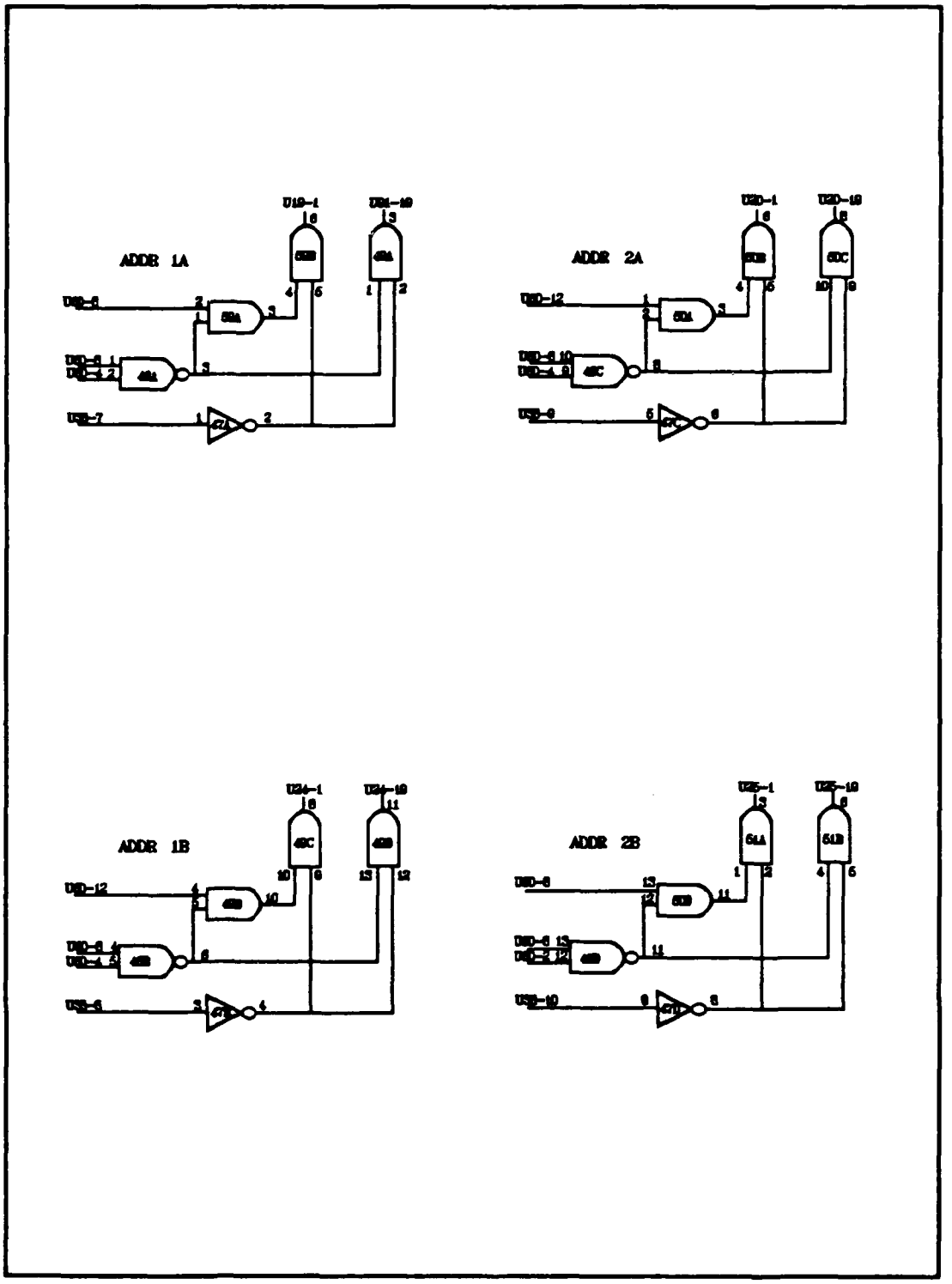

Figure 2f. Schematic Diagram

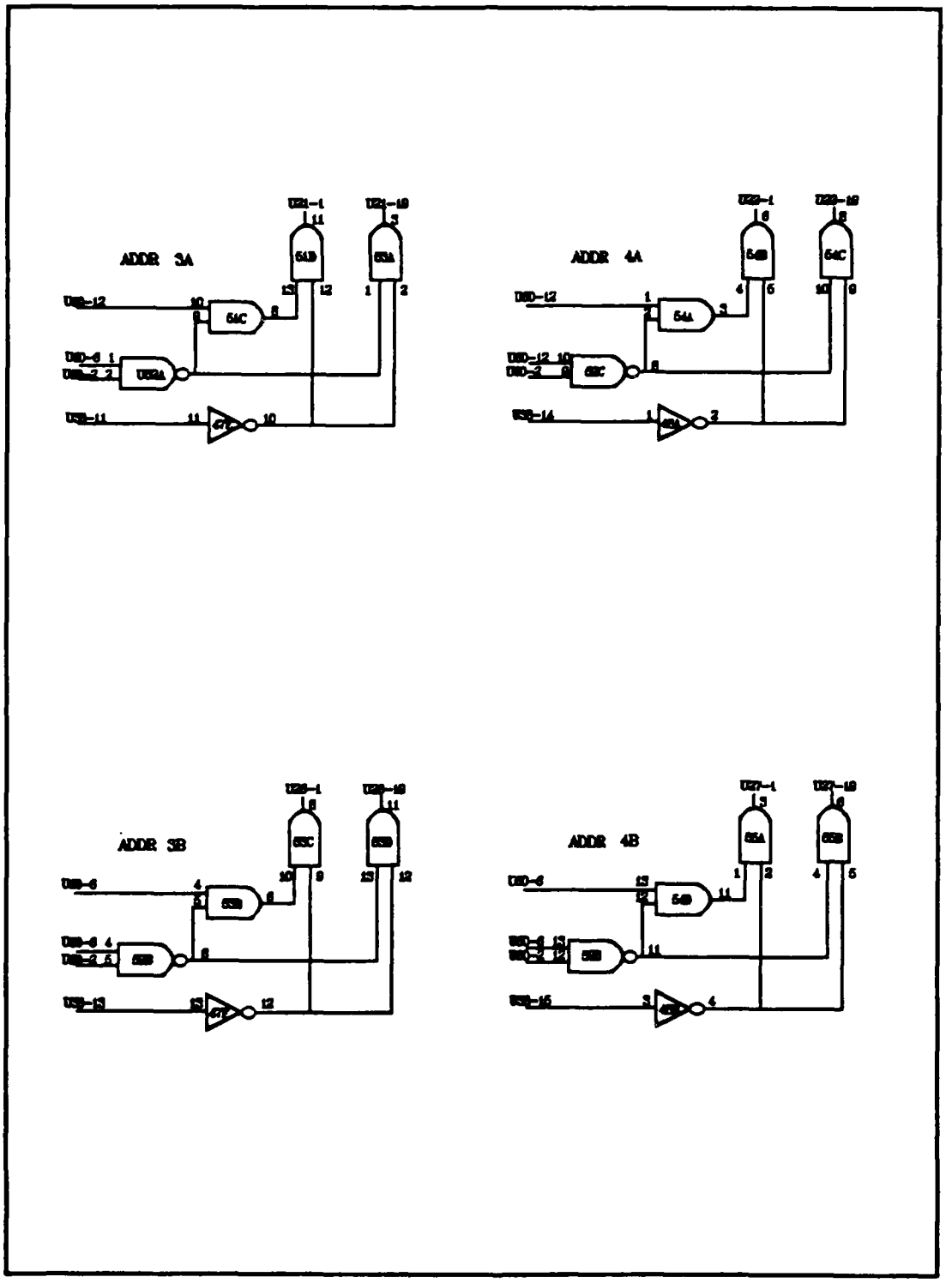

Figure 2g. Schematic Diagram

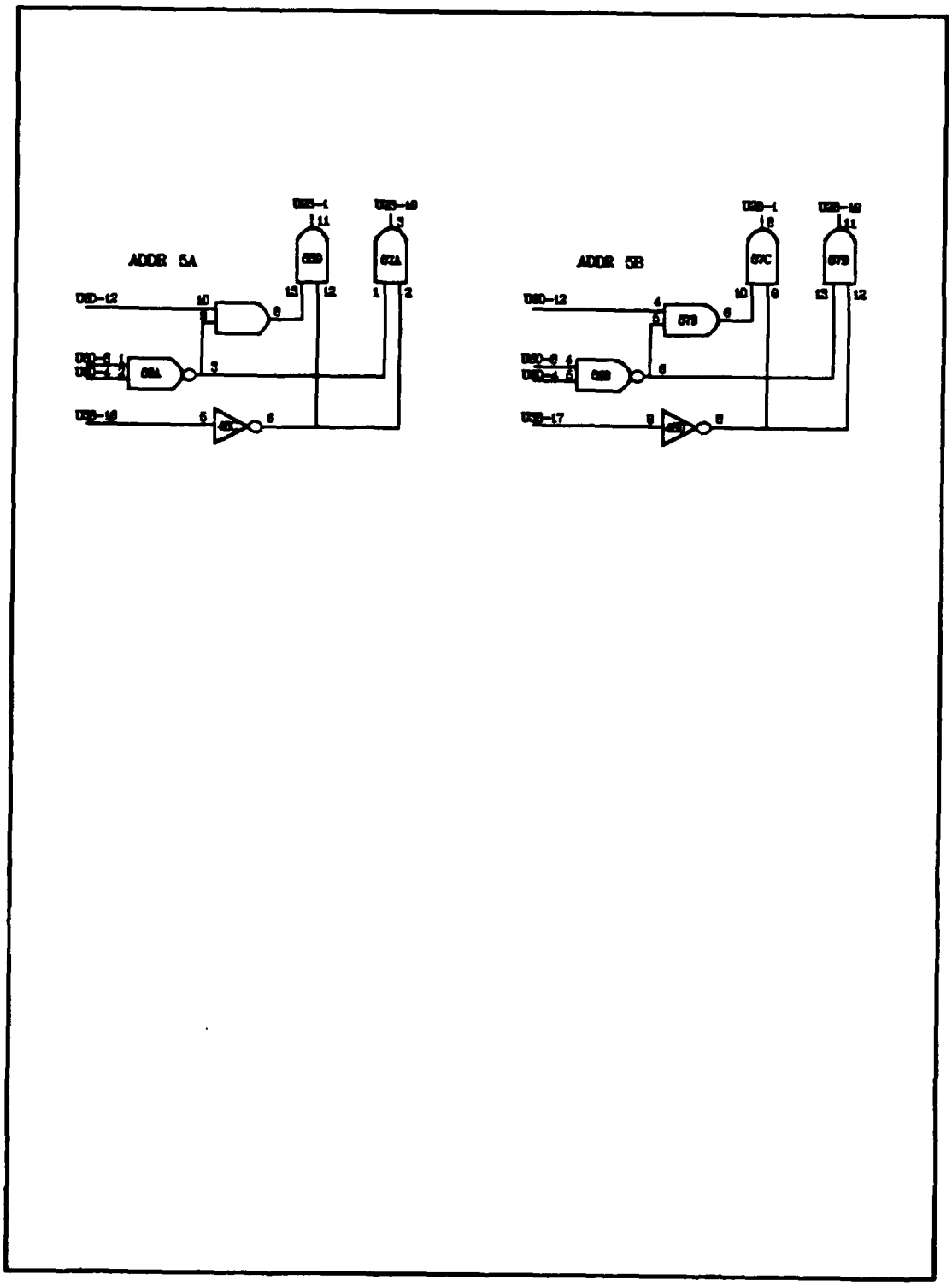

Figure 2h. Schematic Diagram

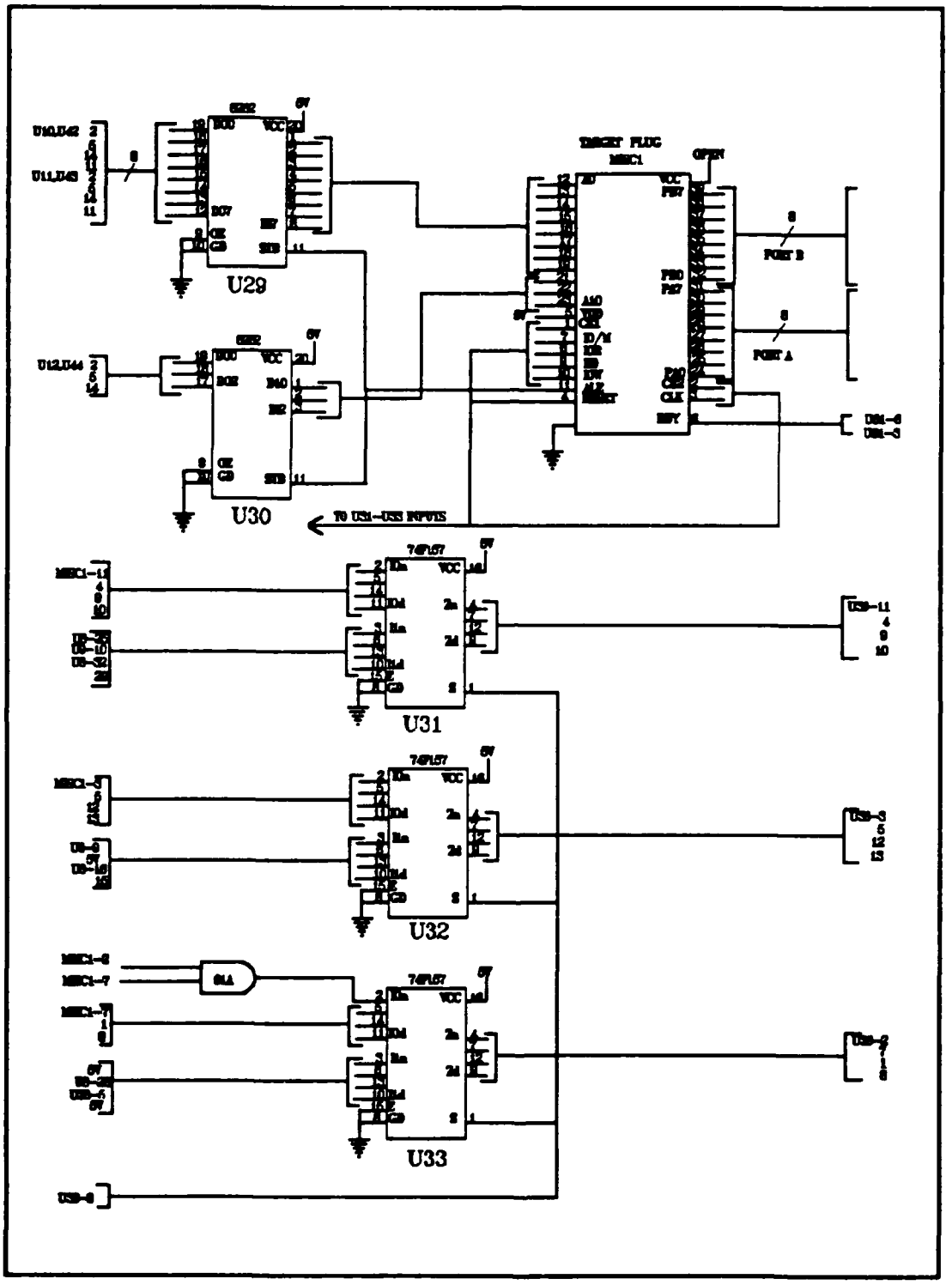

Figure 2i. Schematic Diagram

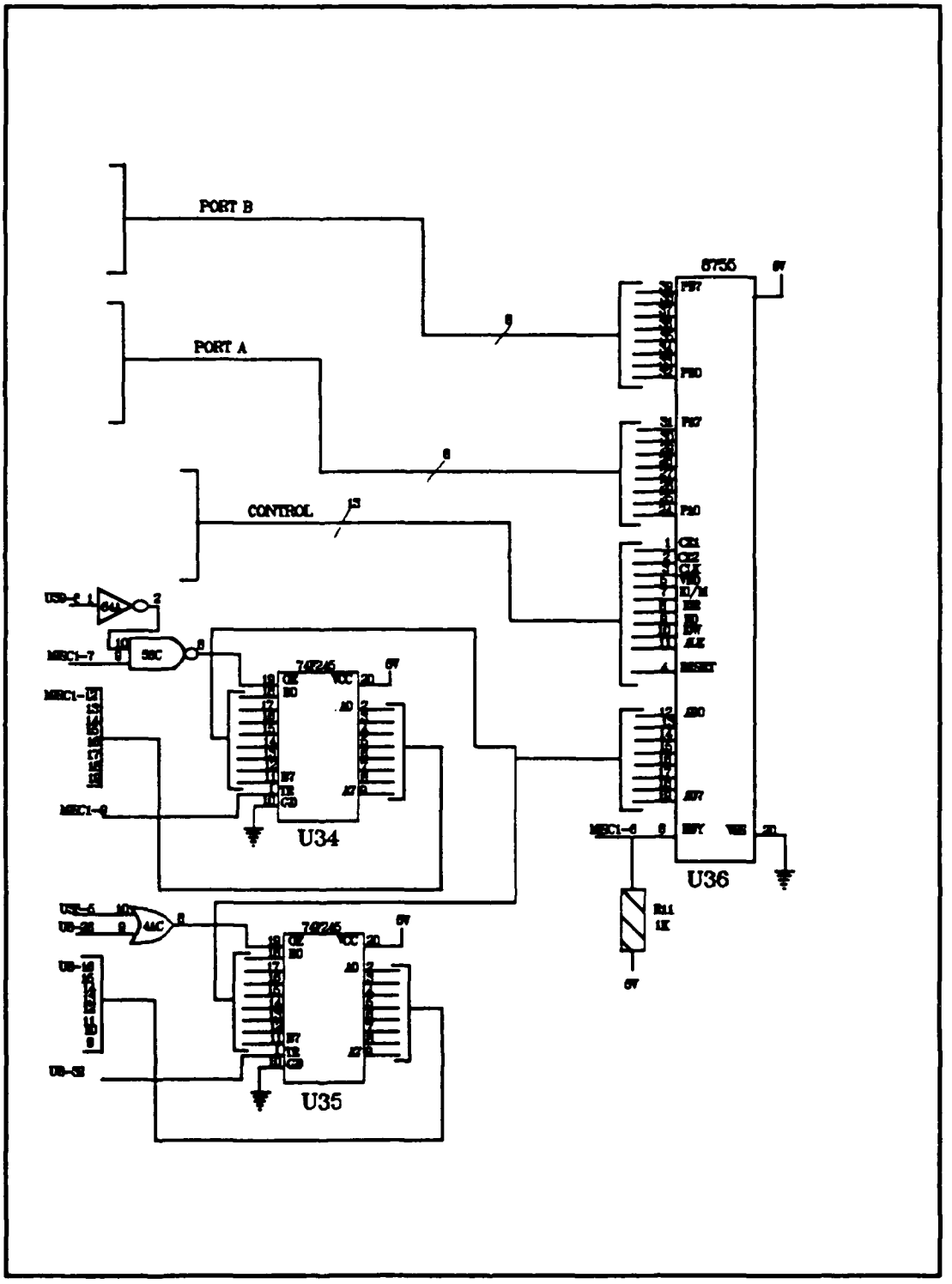

Figure 2j. Schematic Diagram

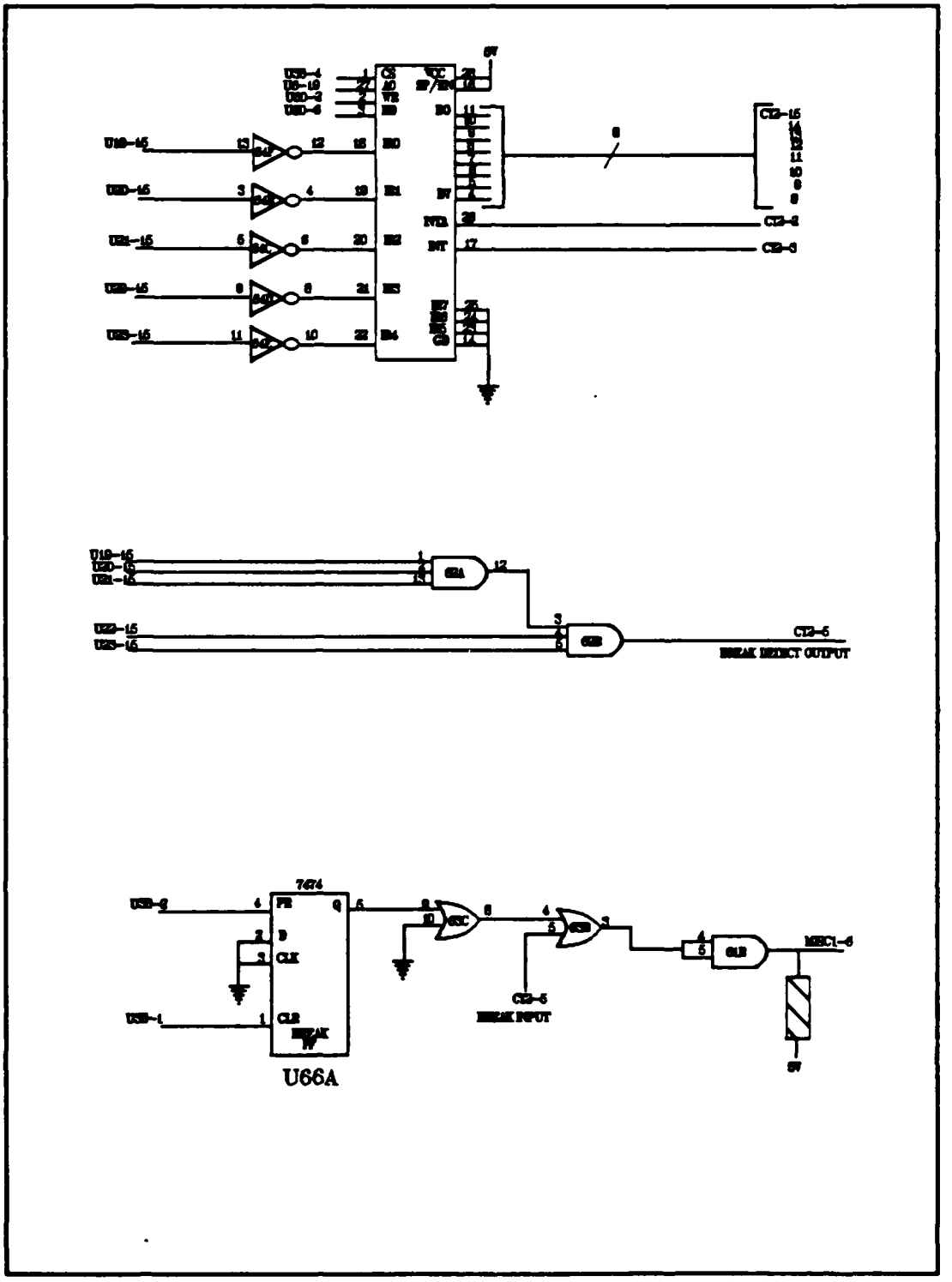

Figure 2k. Schematic Diagram

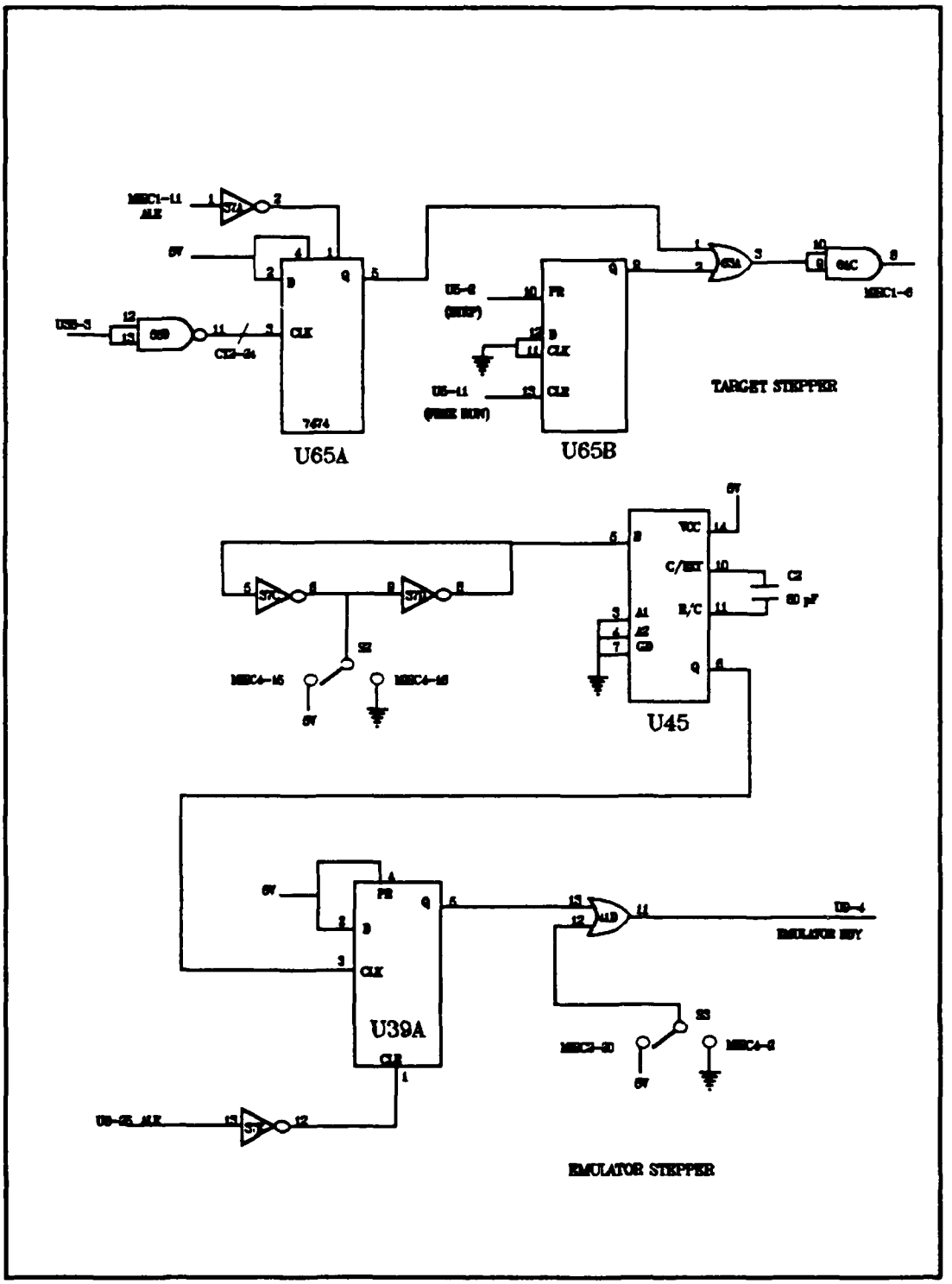

Figure 21. Schematic Diagram

PART		NAME	BOARD	PINS
U1	1488	RS232 DRIVER	2	14
U2	1489	RS232 RECEIVER	2	14
U3	8251A	UART	2	28
U4	8286	TRANSCEIVER	2	20
U5	8205	DECODER	1	16
U6	8282	LATCH	1	20
U7	8282	LATCH	1	20
U8	8088	CPU	1	40
U9	8284	CLOCK	1	18
U10	74F157A	2-TO-1 MULTIPLEXOR	1	16
U11	74F157A	2-TO-1 MULTIPLEXOR	1	16
U12	74F157A	2-TO-1 MULTIPLEXOR	1	16
U13	74F245	FAST TRANSCEIVER	1	20
U14	74F245	FAST TRANSCEIVER	1	20
U15	MSM51257	32Kx8 SRAM	1	28
U16	MSM51257	32Kx8 SRAM	1	28
U17	2716	2Kx8 EEPROM	1	24
U18	MSM5128	2Kx8 SRAM	1	24
U19	74F524	FAST REG/COMPARATOR	2	20
U20	74F524	FAST REG/COMPARATOR	2	20
U21	74F524	FAST REG/COMPARATOR	2	20
U22	74F524	FAST REG/COMPARATOR	2	20
U23	74F524	FAST REG/COMPARATOR	2	20
U24	74F524	FAST REG/COMPARATOR	2	20
U25	74F524	FAST REG/COMPARATOR	2	20
U26	74F524	FAST REG/COMPARATOR	2	20
U27	74F524	FAST REG/COMPARATOR	2	20
U28	74F524	FAST REG/COMPARATOR	2	20
U29	8282	LATCH	1	20
U30	8282	LATCH	1	20
U31	74F157A	FAST 2-TO-1 MUX	1	16
U32	74F157A	FAST 2-TO-1 MUX	1	16
U33	74F157A	FAST 2-TO-1 MUX	1	16
U34	74F245	FAST TRANSCEIVER	1	20
U35	74F245	FAST TRANSCEIVER	1	20
U36	8755	I/O ROM	1	40
U37	7404	HEX INVERTOR	1	14
U38	74154	4-TO-16 DECODER	2	24
U39	7474	DUAL D-FF	1	14
U40	7432	QUAD-2 INPUT OR	1	14
U41	7432	QUAD-2 INPUT OR	1	14
U42	74F157	2-TO-1 MUX	1	16
U43	74F157	2-TO-1 MUX	1	16
U44	74F157	2-TO-1 MUX	1	16

Table 1a. Parts List

U45 74121	ONE SHOT MULTI	1	14
U46 7400	QUAD 2-IN NAND	2	14
U47 7404	HEX INVERTOR	2	14
U48 7404	HEX INVERTOR	2	14
U49 7408	QUAD-2 IN AND	2	14
U50 7408	QUAD-2 IN AND	2	14
U51 7408	QUAD-2 IN AND	2	14
U52 7400	QUAD-2 IN NAND	2	14
U53 7408	QUAD-2 IN AND	2	14
U54 7408	QUAD-2 IN AND	2	14
U55 7408	QUAD-2 IN AND	2	14
U56 7400	QUAD-2 IN NAND	2	14
U57 7408	QUAD-2 IN AND	2	14
U58 8259A	INTERRUPT CONTROLLER	2	28
U59 7408	QUAD-2 IN AND	2	14
U60 7407	HEX BUFFER (OC)	2	14
U61 7409	QUAD-2 IN AND (OC)	1	14
U62 7411	TRIPLE-3 IN AND	2	14
U63 7432	QUAD-2 IN OR	1	14
U64 7404	HEX INVERTER	2	14
U65 7474	DUAL D FLIPFLOP	1	14
U66 7474	DUAL D FLIPFLOP	2	14

MISCELLANEOUS PARTS

MISC1	40 PIN USER PLUG SOCKET
MISC2	40 PIN BOARD CONNECTOR # 1
MISC3	40 PIN BOARD CONNECTOR # 2
MISC4	20 PIN COMPONENT HOLDER BOARD 1
MISC5	20 PIN COMPONENT HOLDER BOARD 2
J1	RS232 CONNECTOR
J2	+5 VOLT INPUT
J3	+12 VOLT INPUT
J4	-12 VOLT INPUT
J5	COMMON GROUND INPUT
S1	RESET SWITCH
S2	EMULATOR STEPPER CONTROL
S3	EMULATOR STEPPER SWITCH

Table 1b. Parts List

Appendix B: Net Wiring List

U-1 1488 DRIVER

1	-12 VDC
2	U3-19
3	RE232-2
4	U3-23
5	U3-23
6	RS232-4
7	GND
8	NC
9	NC
10	NC
11	RS232-20
12	U3-24
13	U3-24
14	+12 VDC

U-2 1489 RECEIVER

1	RS232-5,MISC5-1
2	NC
3	U3-17
4	RS232-3
5	NC
6	U3-3
7	GND
8	NC
9	U3-22
10	RS232-6
11	NC
12	NC
13	NC
14	+12 VDC

U-3 8251A UART

1	U4-3
2	U4-4
3	U2-6
4	GND
5	U4-5
6	U4-6
7	U4-7
8	U4-8
9	153.6 Khz
10	CT2-19
11	U38-6

12	CT2-19
13	CT2-20
14	NC
15	NC
16	NC
17	U2-3
18	NC
19	U1-2
20	CT2-16
21	CT2-18
22	U2-9
23	U1-4,U1-5
24	U1-12,U1-13
25	153.6 Khz
26	+5 VDC
27	U4-1
28	U4-2

U-4 8286 TRANSCEIVER

1	U3-27
2	U3-28
3	U3-1
4	U3-2
5	U3-5
6	U3-6
7	U3-7
8	U3-8
9	U38-6
10	GND
11	U48-10
12	CT2-8
13	CT2-9
14	CT2-10
15	CT2-11
16	CT2-12
17	CT2-13
18	CT2-14
19	CT2-15
20	+5 VDC

U-5 8205 DECODER

1	U7-19
2	U7-18
3	U7-17
4	GND
5	U8-28
6	+5 VDC
7	U17-18

8	GND
9	U12-10
10	NC
11	NC
12	NC
13	NC
14	NC
15	U40-1,U40-4
16	+5 VDC

U-6 8282 LATCH

1	CT1-15
2	CT1-14
3	CT1-13
4	CT1-12
5	CT1-11
6	CT1-10
7	CT1-9
8	CT1-8
9	GND
10	GND
11	U8-25
12	CT1-28
13	CT1-27
14	CT1-26
15	CT1-25
16	CT1-24
17	CT1-23
18	CT1-22
19	CT1-21
20	+5 VDC

U-7 8282 LATCH

1	U8-38
2	U8-37
3	U8-36
4	NC
5	NC
6	NC
7	NC
8	NC
9	GND
10	GND
11	U8-25
12	NC
13	NC
14	NC

15	NC
16	NC
17	U5-3
18	U5-2
19	U5-1
20	+5 VDC

U-8 8088-2 CPU

1	GND
2	U15-1,U16-1
3	U15-26,U16-26
4	U15-2,U16-2
5	U15-23,U16-23
6	U17-19,U15-21,U16-21,U12-3
7	U17-22,U15-24,U16-24,U12-6
8	U17-23,U12-13,U15-25,U16-25
9	U6-8,U4-12,U14-9,U35-9
10	U6-7,U4-13,U14-8,U35-8
11	U6-6,U4-14,U14-7,U35-7
12	U6-5,U4-15,U14-6,U35-6
13	U6-4,U4-16,U14-5,U35-5
14	U6-3,U4-17,U14-4,U35-4
15	U6-2,U4-18,U14-3,U35-3
16	U6-1,U4-19,U14-2,U35-2
17	GND
18	CT1-22
19	U9-8
20	GND
21	U9-10
22	U9-5
23	GND
24	CT1-23
25	U6-11,U7-11,U37-11
26	U41-1
27	U35-1,U14-1
28	U33-6,U5-5,CT1-17
29	CT1-19
30	NC
31	GND
32	CT1-20
33	+5 VDC
34	NC
35	NC
36	U7-1
37	U7-2
38	U7-3
39	U37-1,U40-4
40	+5 VDC

U-9 8284 CLOCK

1	GND
2	CT1-16
3	GND
4	U41-11
5	U8-22
6	GND
7	+5VDC
8	U8-19,U32-3
9	GND
10	U8-21,U31-6,CT1-18
11	MISC4-16 & 17
12	NC
13	GND
14	NC
15	GND
16	CRYSTAL
17	CRYSTAL
18	+5 VDC

U-10 74F157A MULTIPLEXER

1	U39-8
2	U29-19
3	U6-19
4	U18-8
5	U29-18
6	U6-18
7	U18-7
8	GND
9	U18-5
10	U6-16
11	U29-16
12	U18-6
13	U6-17
14	U29-17
15	GND
16	+5VDC

U-11 74F157A MULTIPLEXER

1	U39-8
2	U29-15
3	U6-15
4	U18-4
5	U29-14
6	U6-14
7	U18-3
8	GND
9	U18-1
10	U6-12

11	U29-12
12	U18-2
13	U6-13
14	U29-13
15	GND
16	+5VDC

U-12 74F157A MULTIPLEXER

1	U39-8
2	U30-19
3	U8-8
4	U18-23
5	U30-18
6	U8-7
7	U18-22
8	GND
9	U18-18
10	U5-9
11	U40-8
12	U18-19
13	U8-6
14	U30-17
15	GND
16	+5VDC

U-13 74F245 DATA LATCH

1	GND
2	MISC1~12
3	MISC1~13
4	MISC1~14
5	MISC1~15
6	MISC1~16
7	MISC1~17
8	MISC1~18
9	MISC1~19
10	GND
11	U18-17,U34-9
12	U18-16,U34-8
13	U18-15,U34-7
14	U18-14,U34-6
15	U18-13,U34-5
16	U18-11,U34-4
17	U18-10,U34-3
18	U18- 9,U34-2
19	U40-11
20	+5VDC

U-14 74F245 DATA LATCH

1	U8-27
2	U8-16
3	U8-15
4	U8-14
5	U8-13
6	U8-12
7	U8-11
8	U8-10
9	U8-9
10	GND
11	U18-17
12	U18-16
13	U18-15
14	U18-14
15	U18-13
16	U18-11
17	U18-10
18	U18-9
19	U41-6
20	+5VDC

U-15 MSM51257 SRAM

1	U8-2
2	U8-4
3	U6-12
4	U6-13
5	U6-14
6	U6-15
7	U6-16
8	U6-17
9	U6-18
10	U6-19
11	U8-16
12	U8-15
13	U8-14
14	GND
15	U8-13
16	U8-12
17	U8-11
16	U8-10
19	U8-9
20	U40-3
21	U8-6
22	U60-8
23	U8-5
24	U8-7
25	U8-8
26	U8-3
27	U60-4
28	+5VDC

U-16 MSM51257 SRAM

1	U8-2
2	U8-4
3	U6-12
4	U6-13
5	U6-14
6	U6-15
7	U6-16
8	U6-17
9	U6-18
10	U6-19
11	U8-16
12	U8-15
13	U8-14
14	GND
15	U8-13
16	U8-12
17	U8-11
16	U8-10
19	U8-9
20	U40-6
21	U8-6
22	U60-8
23	U8-5
24	U8-7
25	U8-8
26	U8-3
27	U60-4
28	+5VDC

U-17 2716 EPROM

1	U6-12
2	U6-13
3	U6-14
4	U6-15
5	U6-16
6	U6-17
7	U6-18
8	U6-19
9	U8-16
10	U8-15
11	U8-14
12	GND
13	U8-13
14	U8-12
15	U8-11
16	U8-10
17	U8-9
18	U5-7
19	U8-6

20	U8-32
21	+5VDC
22	U8-7
23	U8-8
24	+5VDC

U-18 MSM5128 SRAM

1	U11-9
2	U11-12
3	U11-7
4	U11-4
5	U10-9
6	U10-12
7	U10-7
8	U10-4
9	U13-18,U14-18
10	U13-17,U14-17
11	U13-16,U14-16
12	GND
13	U13-15,U15-15
14	U13-14,U14-14
15	U13-13,U14-13
16	U13-12,U14-12
17	U13-11,U14-11
18	U12-9
19	U12-12
20	U31-12
21	U36-10
22	U12-7
23	U12-4
24	+5VDC

U-19 74F524 REGISTER/COMPARATOR

1	U59-3
2	U42-4
3	U42-7
4	U42-12
5	U42-9
6	U43-4
7	U43-7
8	U43-12
9	U43-9
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U24-15,U64-13

16	U24-17
17	+5VDC
18	U24-17
19	U49-3
20	+5VDC

U-20 74F524 REGISTER/COMPARATOR

1	U50-6
2	U42-4
3	U42-7
4	U42-12
5	U42-9
6	U43-4
7	U43-7
8	U43-12
9	U43-9
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U25-15,U64-3
16	U25-17
17	+5VDC
18	U25-17
19	U50-8
20	+5VDC

U-21 74F524 REGISTER/COMPARATOR

1	U51-3
2	U42-4
3	U42-7
4	U42-12
5	U42-9
6	U43-4
7	U43-7
8	U43-12
9	U43-9
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U26-15,U64-5
16	U26-17
17	+5VDC
18	U26-17
19	U53-3

20	+5VDC

U-22 74F524 REGISTER/COMPARATOR

1	U54-6
2	U42-4
3	U42-7
4	U42-12
5	U42-9
6	U43-4
7	U43-7
8	U43-12
9	U43-9
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U27-15,U64-9
16	U27-17
17	+5VDC
18	U27-17
19	U54-8
20	+5VDC

U-23 74F524 REGISTER/COMPARATOR

1	U55-11
2	U42-4
3	U42-7
4	U42-12
5	U42-9
6	U43-4
7	U43-7
8	U43-12
9	U43-9
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U28-15,U64-11
16	U28-17
17	+5VDC
18	U28-17
19	U57-3
20	+5VDC

U-24 74F524 REGISTER/COMPARATOR

1	U49-8
2	U44-4
3	U44-7
4	U44-12
5	GND
6	GND
7	GND
8	GND
9	GND
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U19-15
16	NC
17	U19-16,U19-18
18	GND
19	U49-11
20	+5VDC

U-25 74F524 REGISTER/COMPARATOR

1	U51-3
2	U44-4
3	U44-7
4	U44-12
5	GND
6	GND
7	GND
8	GND
9	GND
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U20-15
16	NC
17	U20-16,U20-18
18	GND
19	U51-6
20	+5VDC

U-26 74F524 REGISTER/COMPARATOR

1	U53-8
2	U44-4

3	U44-7
4	U44-12
5	GND
6	GND
7	GND
8	GND
9	GND
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U21-15
16	NC
17	U21-16,U21-18
18	GND
19	U53-11
20	+5VDC

U-27 74F524 REGISTER/COMPARATOR

1	U55-3
2	U44-4
3	U44-7
4	U44-12
5	GND
6	GND
7	GND
8	GND
9	GND
10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U22-15
16	NC
17	U22-16,U22-18
18	GND
19	U55-6
20	+5VDC

U-28 74F524 REGISTER/COMPARATOR

1	U57-8
2	U44-4
3	U44-7
4	U44-12
5	GND
6	GND
7	GND
8	GND
9	GND

10	GND
11	U44-9
12	GND
13	NC
14	NC
15	U23-15
16	NC
17	U23-16,U23-18
18	GND
19	U57-11
20	+5VDC

U-29 8282 LATCH

1	MISC1-12
2	MISC1-13
3	MISC1-14
4	MISC1-15
5	MISC1-16
6	MISC1-17
7	MISC1-18
8	MISC1-19
9	GND
10	GND
11	MISC1-11
12	U43-11,U11-11
13	U43-14,U11-14
14	U43-5 ,U11-5
15	U43-2 ,U11-2
16	U42-11,U10-11
17	U42-14,U10-14
18	U42-5 ,U10-5
19	U42-2 ,U10-2
20	+5 VDC

U-30 8282 LATCH

1	MISC1-21
2	MISC1-22
3	MISC1-23
4	NC
5	NC
6	NC
7	NC
8	NC
9	GND
10	GND
11	MISC1-11
12	NC
13	NC

14	NC
15	NC
16	NC
17	U44-14,U12-14
18	U44-5,U12-5
19	U44-2,U12-2
20	+5 VDC

U-31 74F157A MULTIPLEXER

1	U39-8
2	MISC1-11
3	U8-25
4	U36-11
5	MISC1-4
6	U9-10
7	U36-4
8	GND
9	U36-10
10	U8-29
11	MISC1-10
12	U36-9
13	U8-32
14	MISC1-9
15	GND
16	+5VDC

U-32 74F157A MULTIPLEXER

1	U39-8
2	MISC1-3
3	U9-8
4	U36-3
5	MISC1-5
6	+5VDC
7	U36-5
8	GND
9	U36-13
10	U8-15
11	MISC1-13
12	U36-12
13	U8-16
14	MISC1-12
15	GND
16	+5VDC

U-33 74F157A MULTIPLEXER

1	U39-8
2	U61-3
3	+5VDC

4	U36-2
5	MISC1-7
6	U8-28
7	U36-7
8	GND
9	U36-8
10	+5VDC
11	MISC1-8
12	U36-1
13	U38-5
14	MISC1-1
15	GND
16	+5VDC

U-34 74F245 LATCH

1	MISC1-9
2	U13-2
3	U13-3
4	U13-4
5	U13-5
6	U13-6
7	U13-7
8	U13-8
9	U13-9
10	GND
11	U36-19
12	U36-18
13	U36-17
14	U36-16
15	U36-15
16	U36-14
17	U36-13
18	U36-12
19	CT1-4
20	+5 VDC

U-35 74F245 LATCH

1	U8-32
2	U8-16
3	U8-15
4	U8-14
5	U8-13
6	U8-12
7	U8-11
8	U8-10
9	U8-9
10	GND
11	U36-19
12	U36-18

13	U36-17
14	U36-16
15	U36-15
16	U36-14
17	U36-13
18	U36-12
19	U41-8
20	+5 VDC

U-36 8755 I/O PORTS

1	U33-12
2	U33-4
3	U32-4
4	U31-7
5	U32-7
6	R-16
7	U33-7
8	U33-9
9	U31-12
10	U31-9
11	U31-4
12	U32-12,U34-18,U35-18
13	U32- 9,U34-17,U35-17
14	U34-16,U35-16
15	U34-15,U35-15
16	U34-14,U35-14
17	U34-13,U35-13
18	U34-12,U35-12
19	U34-11,U34-11
20	GND
21	NC
22	NC
23	NC
24	MISC1-24
25	MISC1-25
26	MISC1-26
27	MISC1-27
28	MISC1-28
29	MISC1-29
30	MISC1-30
31	MISC1-31
32	MISC1-32
33	MISC1-33
34	MISC1-34
35	MISC1-35
36	MISC1-36
37	MISC1-37
38	MISC1-38
39	MISC1-39
40	+5 VDC

U-37 7404 HEX INVERTER

1	MISC1-11
2	U65-1
3	U8-39
4	U40-1
5	U37-8,U45-5
6	U37-9,MISC4-7
7	GND
8	U37-5,U45-5
9	U37-6,MISC4-7
10	U39-1
11	U7-11
12	U39-11
13	U5-13
14	+5VDC

U-38 74154 4 TO 16 DECODER

1	U66-1
2	U66-4
3	U56-13
4	U58-1
5	U41-10,U33-13
6	U4-9,U3-11
7	U47-1
8	U47-3
9	U47-5
10	U47-9
11	U47-11
12	GND
13	U47-13
14	U48-1
15	U48-3
16	U48-5
17	U48-9
18	GND
19	U48-12
20	CT2-28
21	CT2-27
22	CT2-26
23	CT2-25
24	+5VDC

U-39 7474 DUAL D FLIP-FLOP

1	U37-10
2	U39-4,+5VDC
3	U45-6
4	U39-2,+5VDC

5	U41-13
6	NC
7	GND
8	U10-1,U11-1,U12-1,U32-1 U31-1,U64-1,CT1-2,U33-1
9	U41-5
10	U39-13,+5VDC
11	U37-12
12	U6-19
13	U39-10,+5VDC
14	+5VDC

U-40 7408 QUAD 2-INPUT AND

1	U37-4
2	U5-15
3	U15-20
4	U8-39
5	U5-15
6	U16-20
7	GND
8	U12-11
9	MISC1-7
10	MISC1-1
11	U13-19
12	MISC1-9
13	U40-8
14	+5VDC

U-41 7432 QUAD 2-INPUT OR

1	U8-26
2	U12-9
3	U41-4
4	U41-3
5	U39-9
6	U14-19
7	GND
8	U35-19
9	U8-26
10	U38-5
11	U9-4
12	S3
13	U39-5
14	GND

U-42 74F157A MULTIPLEXER

1	U60-10
2	U29-19
3	U8-16
4	U(19-23)-2
5	U29-18
6	U8-15
7	U(19-23)-3
8	GND
9	U(19-23)-5
10	U8-13
11	U29-16
12	U(19-23)-4
13	U8-14
14	U29-17
15	GND
16	+5VDC

U-43 74F157A MULTIPLEXER

1	U60-10
2	U29-15
3	U8-12
4	U(19-23)-6
5	U29-14
6	U8-11
7	U(19-23)-7
8	GND
9	U(19-23)-9
10	U8-9
11	U29-12
12	U(19-23)-8
13	U8-10
14	U29-13
15	GND
16	+5VDC

U-44 74F157A MULTIPLEXER

1	U60-10
2	U30-19
3	U8-16
4	U(24-28)-2
5	U30-18
6	U8-15
7	U(24-28)-3
8	GND
9	U(19-28)-11
10	U9-8
11	MISC1-3
12	U(24-28)-4
13	U8-14

14	U30-17
15	GND
16	+5VDC

U-45 74121 ONE SHOT

1	NC
2	NC
3	U45-4,GND
4	U45-3,GND
5	U37-8
6	U39-3
7	GND
8	NC
9	+5VDC
10	+80 pf cap
11	-80 pf cap
12	NC
13	NC
14	+5VDC

U-46 7400 QUAD 2-INPUT NAND

1	U60-8
2	U60-4
3	U59-1
4	U60-8
5	U60-4
6	U49-5
7	GND
8	U50-2
9	U60-4
10	U60-8
11	U50-12
12	U60-4
13	U60-8
14	+5VDC

U-47 7404 HEX INVERTER

1	U38-7
2	U59-1,U49-2
3	U38-8
4	U49-9,U49-12
5	U38-9
6	U50-5,U50-9
7	GND
8	U51-2,U51-5
9	U38-10
10	U51-12,U53-2

11	U38-11
12	U53-9,U53-12
13	U38-13
14	+5VDC

U-48 7404 HEX INVERTER

1	U38-14
2	U54-5,U54-9
3	U38-15
4	U55-2,U55-5
5	U38-16
6	U55-12,U57-2
7	GND
8	U57-9,U57-12
9	U38-17
10	U4-11
11	U3-13
12	NC
13	NC
14	+5VDC

U-49 7408 QUAD 2-INPUT AND

1	U46-3
2	U47-2
3	U19-19
4	U60-12
5	U46-6
6	U49-10
7	GND
8	U24-1
9	U47-4
10	U49-6
11	U24-19
12	U47-4
13	U46-6
14	+5VDC

U-50 7408 QUAD 2-INPUT AND

1	U60-12
2	U46-8
3	U50-4
4	U50-3
5	U47-6
6	U20-1
7	GND
8	U20-19
9	U47-6
10	U46-8

11	U51-1
12	U46-11
13	U60-12
14	+5VDC

U-51 7408 QUAD 2-INPUT AND

1	U50-11
2	U47-8
3	U25-1
4	U46-11
5	U47-8
6	U25-19
7	GND
8	U51-13
9	U52-3
10	U60-12
11	U21-1
12	U47-10
13	U51-8
14	+5VDC

U-52 7400 QUAD 2-INPUT NAND

1	U60-6
2	U60-2
3	U51-9
4	U60-6
5	U60-2
6	U53-5
7	GND
8	U54-2
9	U60-2
10	U60-6
11	U54-12
12	U60-2
13	U60-6
14	+5VDC

U-53 7408 QUAD 2-INPUT AND

1	U52-3
2	U47-10
3	U21-19
4	U60-6
5	U52-6
6	U53-10
7	GND
8	U26-1
9	U47-12
10	U53-6

11	U26-19
12	U47-12
13	U53-6
14	+5VDC

U-54 7408 QUAD 2-INPUT AND

1	U60-12
2	U52-8
3	U54-4
4	U54-3
5	U48-2
6	U22-1
7	GND
8	U22-19
9	U48-2
10	U52-8
11	U55-1
12	U60-2
13	U60-6
14	+5VDC

U-55 7408 QUAD 2-INPUT AND

1	U54-11
2	U48-4
3	U27-1
4	U52-11
5	U48-4
6	U27-19
7	GND
8	U55-13
9	U56-3
10	U60-12
11	U23-1
12	U48-6
13	U55-8
14	+5VDC

U-56 7400 QUAD 2-INPUT NAND

1	U60-8
2	U60-4
3	U55-9,U57-1
4	U60-8
5	U60-4
6	U57-5
7	GND
8	U34-19
9	MISC1-7

10	U64-2
11	CT2-24
12	U56-13,U38-3
13	U56-12,U38-3
14	+5VDC

U-57 7408 QUAD 2-INPUT AND

1	U56-3
2	U48-6
3	U23-19
4	U60-12
5	U56-6
6	U57-10
7	GND
8	U28-1
9	U48-8
10	U57-6
11	U28-19
12	U48-8
13	U56-6
14	+5VDC

U-58 8259A INTERRUPT CONTROLLER

1	U38-4
2	U60-2
3	U60-6
4	CT2-8
5	CT2-9
6	CT2-10
7	CT2-11
8	CT2-12
9	CT2-13
10	CT2-14
11	CT2-15
12	NC
13	NC
14	GND
15	NC
16	+5V
17	CT2-22
18	U64-12
19	U64-4
20	U64-6
21	U64-8
22	U64-10
23	GND
24	GND

25	GND
26	CT2-23
27	CT2-21
28	+5V

U-59 7408 QUAD 2-INPUT AND

1	U46-3
2	U60-12
3	U59-4
4	U59-3
5	U47-2
6	U19-1
7	GND
8	NC
9	NC
10	NC
11	NC
12	NC
13	NC
14	+5VDC

U-60 7407 HEX BUFFER

1	CT2-19
2	U52-(2,5,9,12),U3-10
3	CT2-19
4	U46-(2,5,9,12),U56-(2,5)
5	CT2-20
6	U52-(1,4,10,13)
7	GND
8	U3-13,U46-(1,4,10,13), U56-(1,4)
9	CT2-20
10	U42-1,U43-1,U44-1
11	CT2-2
12	U49-4,U59-2,U50-(1,13),U51-10, U53-4,U54-(1,13),U55-10,U57-4
13	CT2-20
14	+5VDC

U-61 7409 QUAD 2-INPUT AND (OC)

1	MISC1-2
2	MISC1-7
3	U33-2
4	U63-6,U61-5
5	U63-6,U61-4

6	MISC1-6
7	GND
8	MISC1-6
9	U63-3,U61-10
10	U63-3,U61-9
11	NC
12	NC
13	NC
14	+5VDC

U-62 7411 TRIPLE-3-INPUT-AND

1	U19-15
2	U20-15
3	U62-12
4	U22-15
5	U23-15
6	CT2-5
7	GND
8	NC
9	NC
10	NC
11	NC
12	U62-3
13	U21-5
14	+5VDC

U-63 7432 QUAD-2-INPUT OR

1	U65-5
2	U65-9
3	U61-9,U61-10
4	U66-5
5	CT2-5
6	U61-4,U61-5
7	GND
8	NC
9	NC
10	NC
11	NC
12	NC
13	NC
14	+5VDC

U-64 7404 HEX INVERTER

1	CT2-2
2	U56-10
3	U20-15
4	U58-19

5	U21-15
6	U58-20
7	GND
8	U58-21
9	U22-15
10	U58-22
11	U23-15
12	U58-18
13	U19-15
14	+5VDC

U-65 7474 DUAL D FLIPFLOP

1	U37-2
2	+5V,U65-4
3	CT1-24
4	+5V,U64-2
5	U63-1
6	NC
7	GND
8	NC
9	U63-2
10	U5-12
11	GND
12	NC
13	U5-11
14	+5VDC

U-66 7474 DUAL D FLIPFLOP

1	U38-1
2	NC
3	GND
4	U38-2
5	U63-4
6	NC
7	GND
8	NC
9	NC
10	NC
11	NC
12	NC
13	NC
14	+5VDC

MISC1 TARGET PLUG

1	U33-14
2	U33-2

3	U32-2
4	U31-5
5	U32-5
6	U61-8,U61-6
7	U33-5
8	U33-11
9	U31-14
10	U31-11
11	U31-2
12	U32-14,U29-1,U13-2
13	U32-11,U29-2,U13-3
14	U29-3,U13-4
15	U29-4,U13-5
16	U29-5,U13-6
17	U29-6,U13-7
18	U29-7,U13-8
19	U29-8,U13-9
20	GND
21	U30-1
22	U30-2
23	U30-3
24	U36-24
25	U36-25
26	U36-26
27	U36-27
28	U36-28
29	U36-29
30	U36-30
31	U36-31
32	U36-32
33	U36-33
34	U36-34
35	U36-35
36	U36-36
37	U36-37
38	U36-38
39	U36-39
40	+5 VDC

MISC2 CONNECTOR BOARD-1

1	U33-13
2	U39-8
3	MISC1-7
4	U34-19
5	U63-5
6	U63-4
7	U9-8
8	U8-9,U6-8,U17-8
9	U8-10,U6-7,U17-7
10	U8-11,U6-6,U17-6
11	U8-12,U6-5,U17-5

12	U8-13,U6-4,U17-4
13	U8-14,U6-3,U17-3
14	U8-15,U6-2,U17-2
15	U8-16,U6-1,U17-1
16	U9-2
17	U8-28
18	U8-21
19	U8-29
20	U8-32
21	U6-19
22	U8-18
23	U8-24
24	U65-15
25	U6-15
26	U6-14
27	U6-13
28	U6-12
29	U29-19
30	U29-18
31	U29-17
32	U29-16
33	U29-15
34	U29-14
35	U29-13
36	U29-12
37	U30-19
38	U30-18
39	U30-17
40	MISC1-3

MISC3 CONNECTOR BOARD-2

1	U38-5
2	U60-11
3	U56-9
4	U56-8
5	U62-6
6	U66-5
7	U44-10
8	U4-12
9	U4-13
10	U4-14
11	U4-15
12	U4-16
13	U4-17
14	U4-18
15	U4-19
16	U3-20
17	U48-13
18	U3-21
19	U60-1,3
20	U60-5,9,13

21	U3-10
22	U58-17
23	U58-26
24	U56-11
25	U38-23
26	U38-22
27	U38-21
28	U38-20
29	U42-2
30	U42-5
31	U42-14
32	U42-11
33	U43-2
34	U43-5
35	U43-14
36	U43-11
37	U44-2
38	U44-5
39	U44-14
40	U44-11

Appendix C: Software Flowcharts

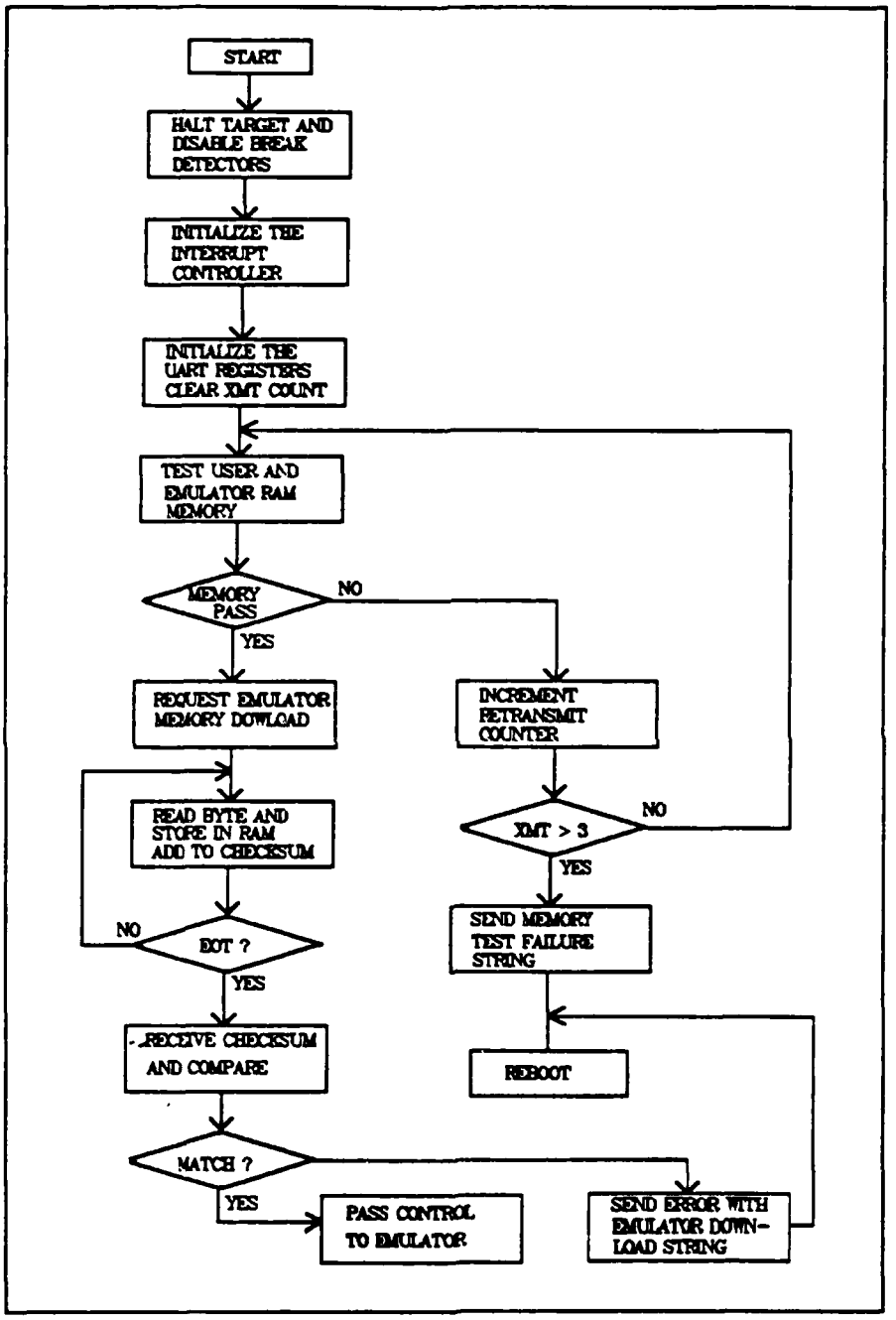

Figure 3. Bootup Flowcharts

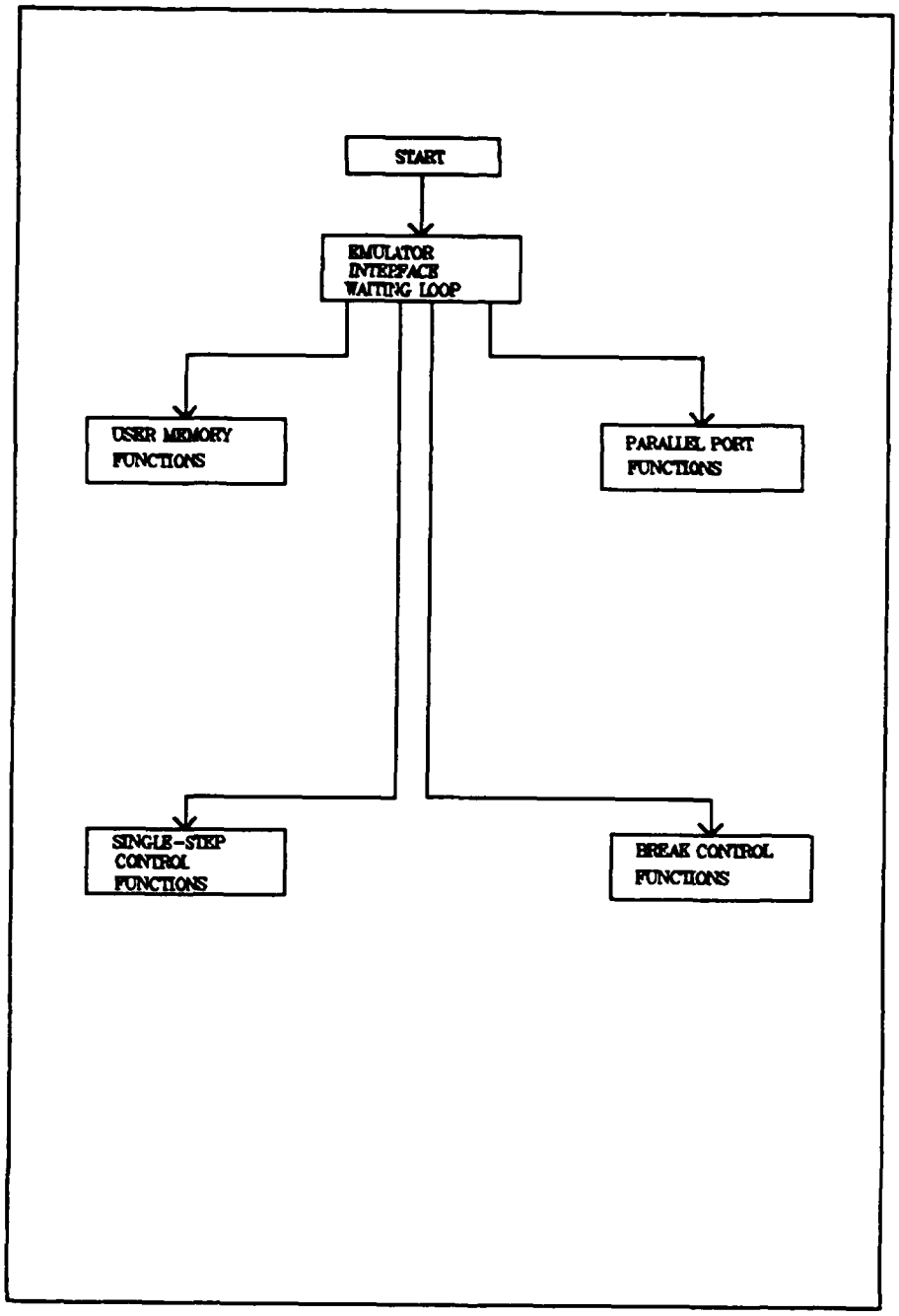

Figure 4. Emulator Main Flowchart

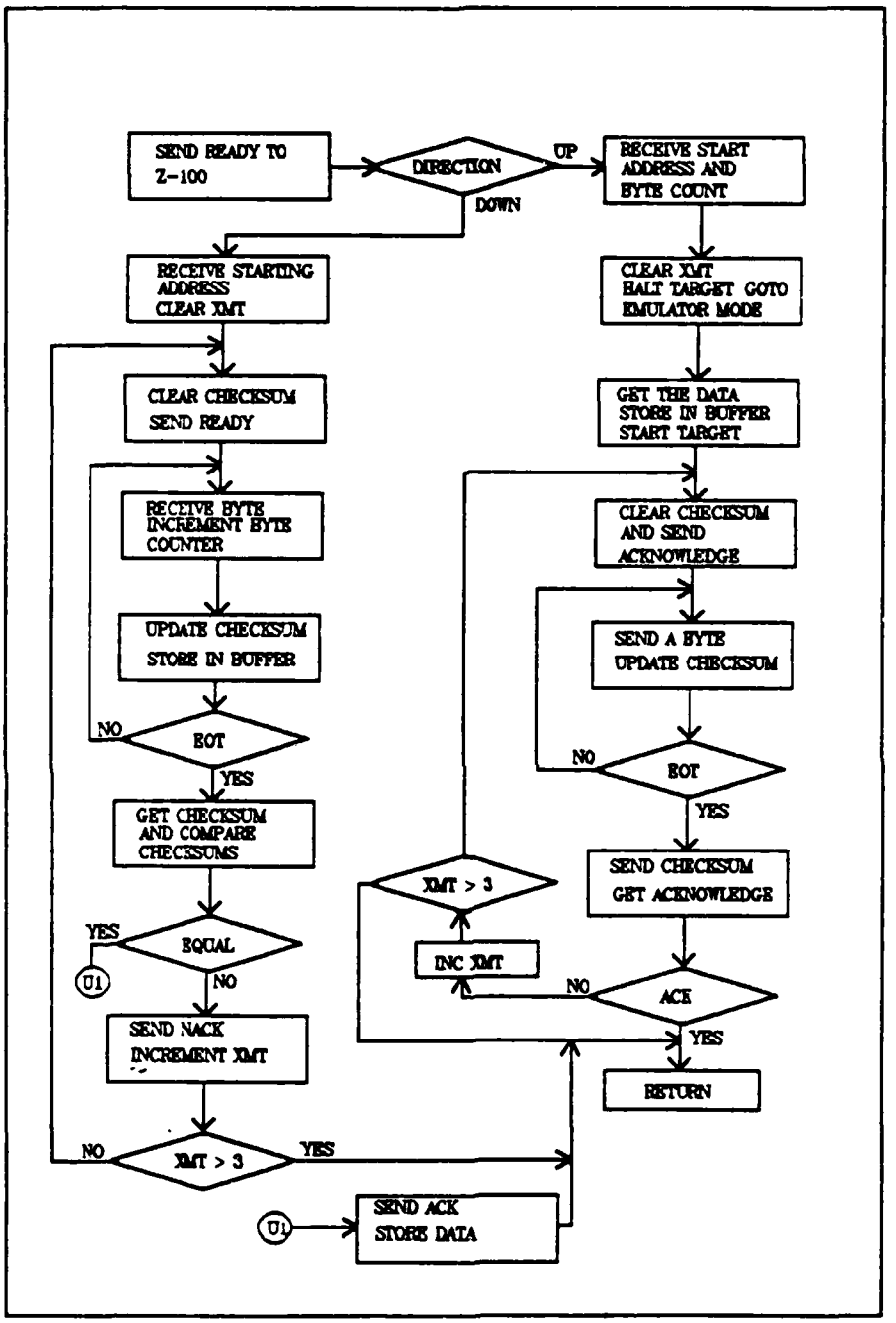

Figure 5. Emulator User Memory Flowchart

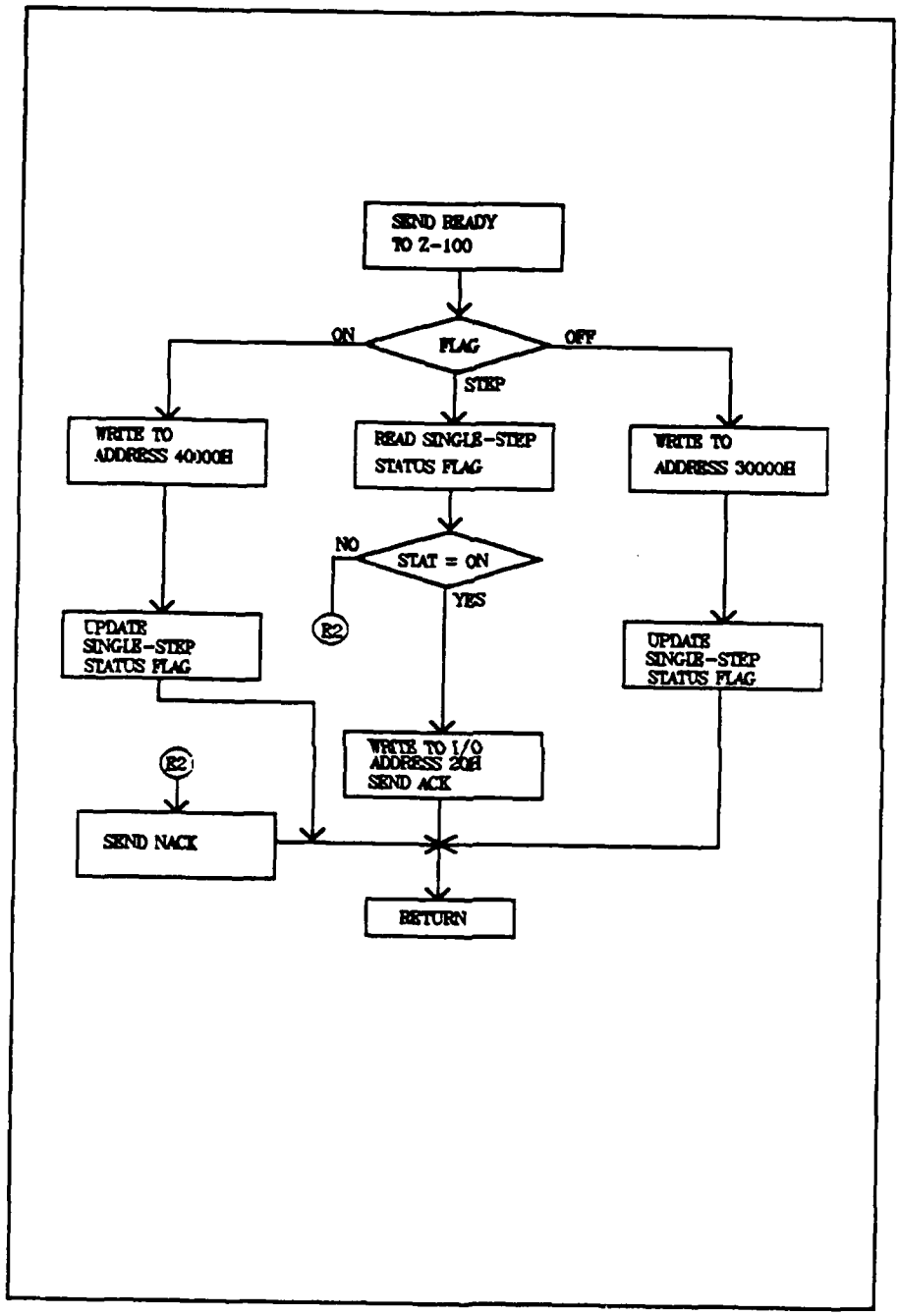

Figure 6. Emulator Single-step Flowchart

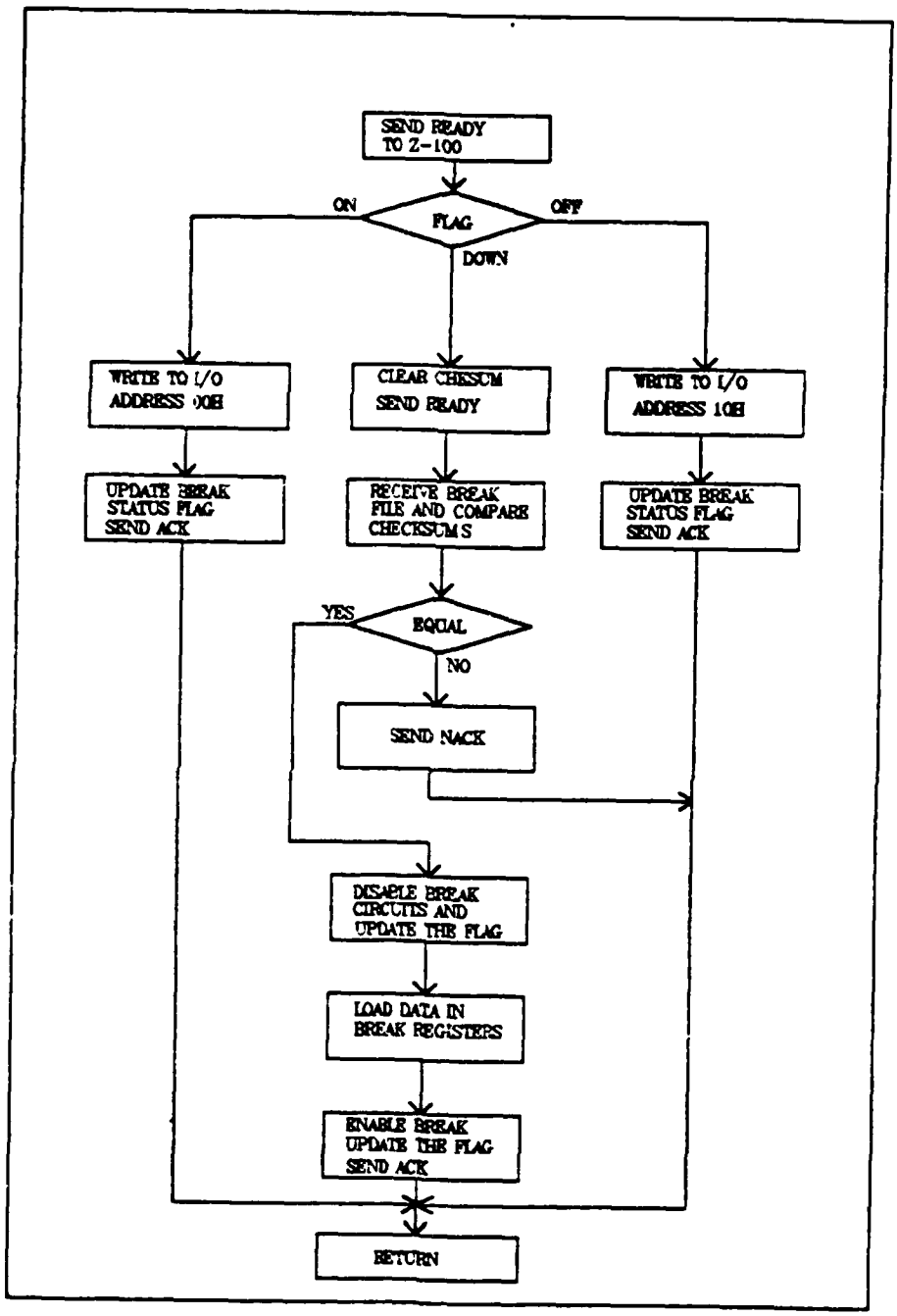

Figure 7. Emulator Break Control Flowchart

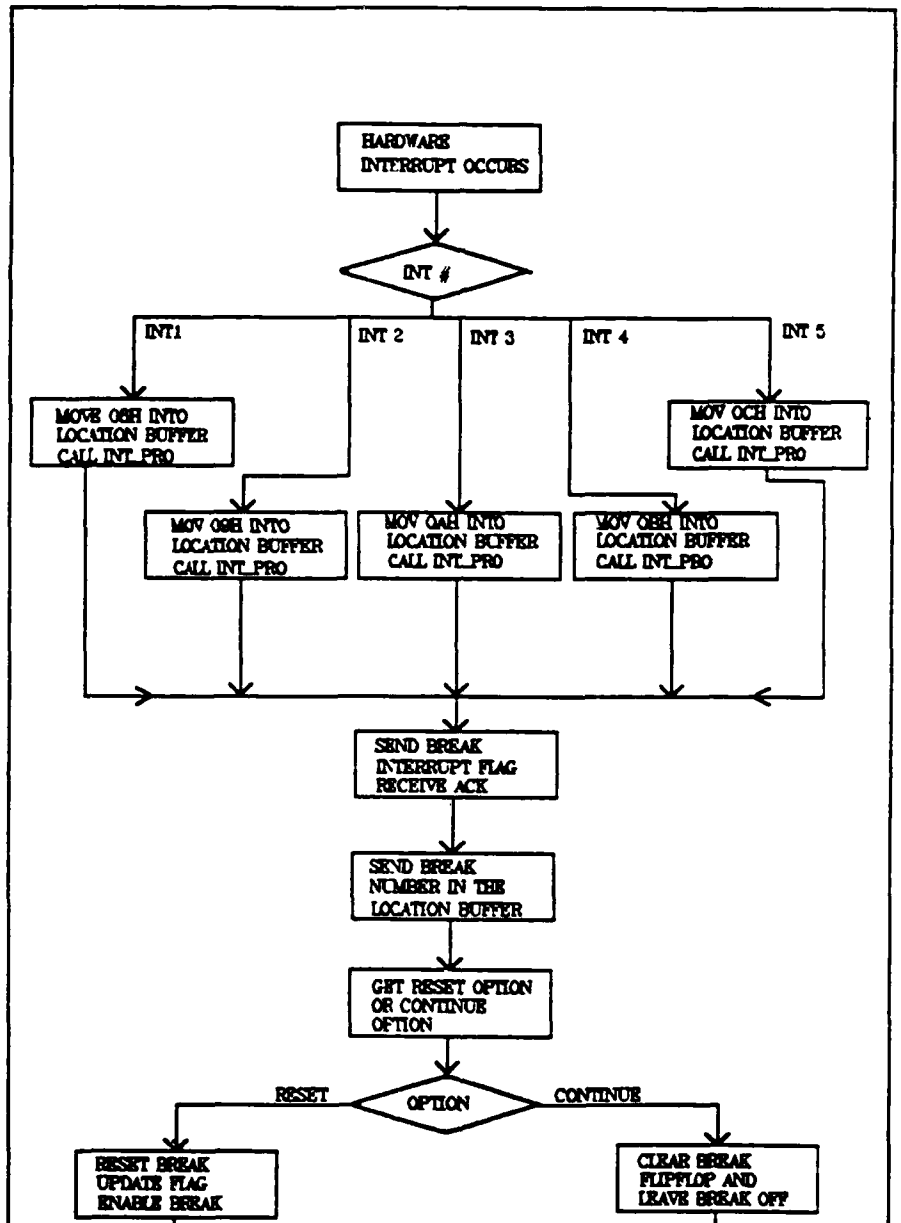

Figure 8. Emulator Break Interrupt Flowchart

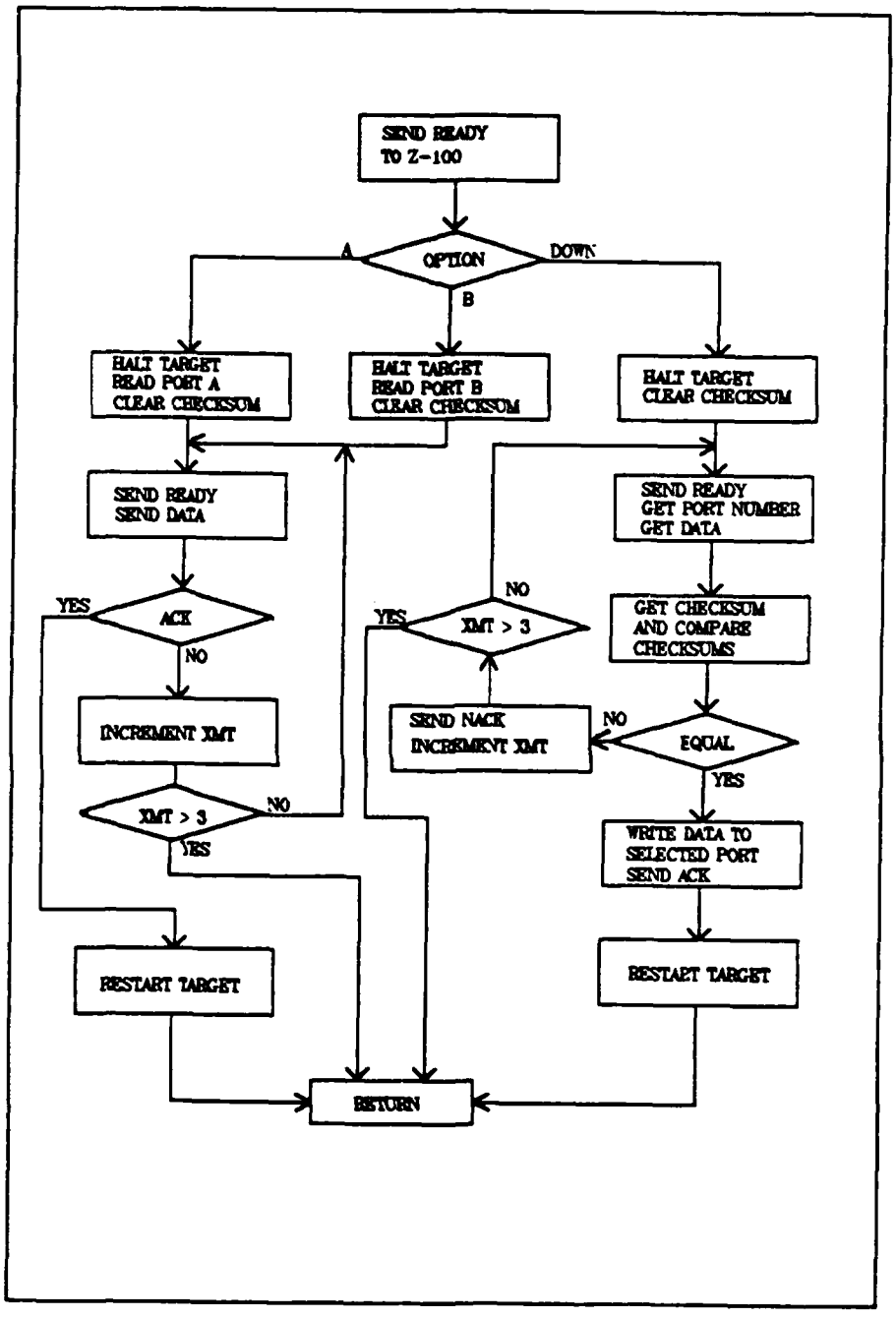

Figure 9. Emulator Parallel I/O Flowchart

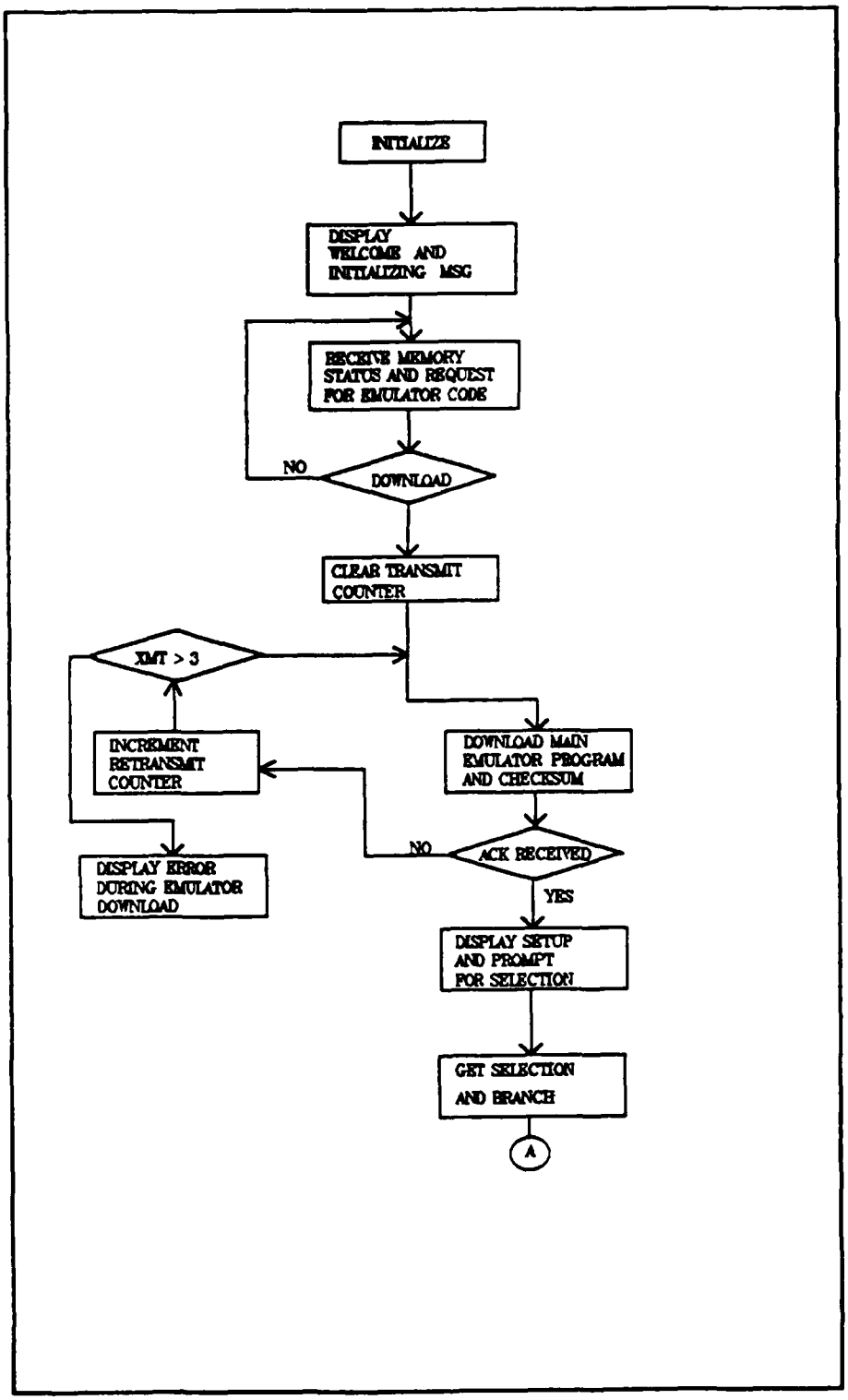

Figure 10a. Z-100 Main Flowchart

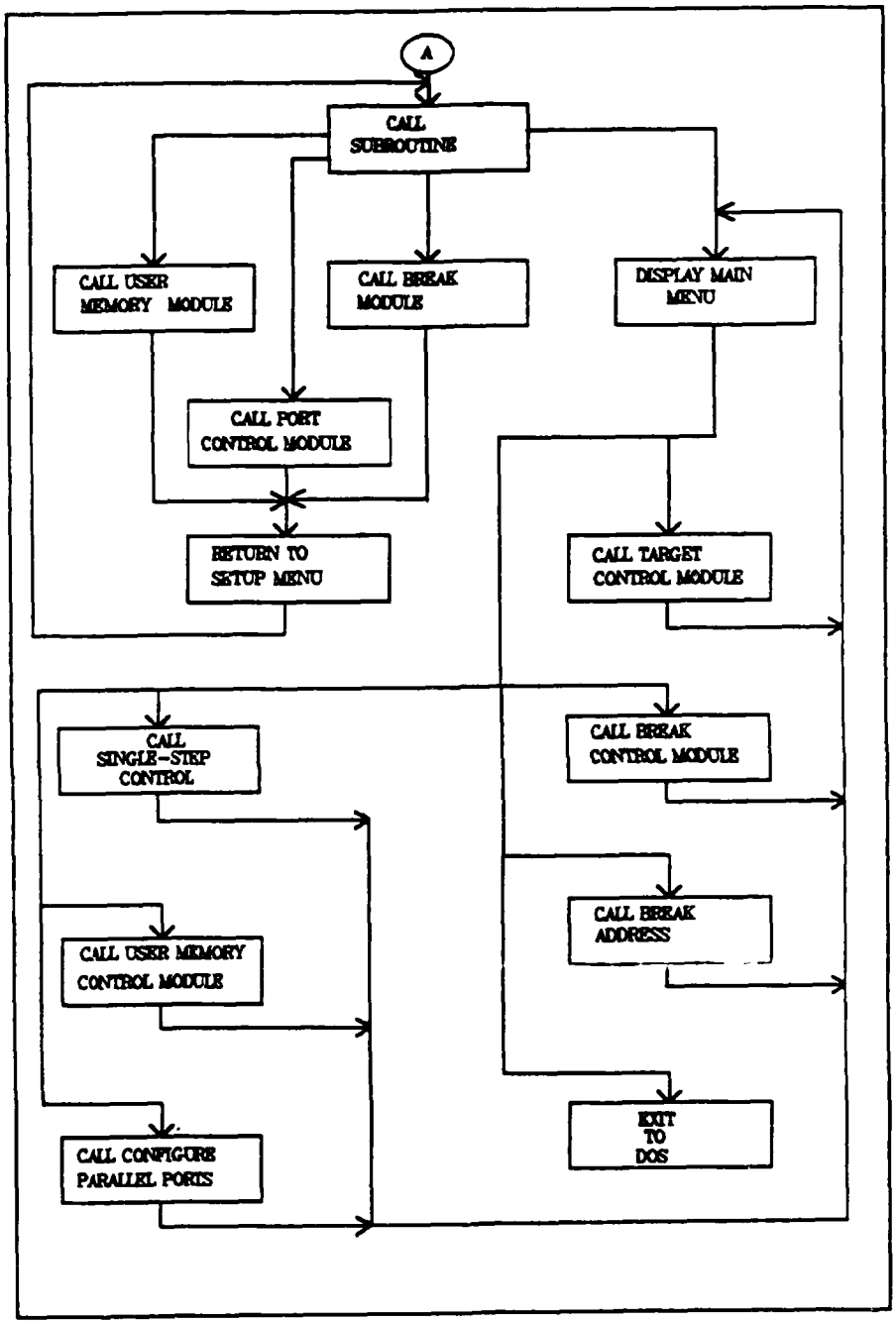

Figure 10b. Z-100 Main Flowchart

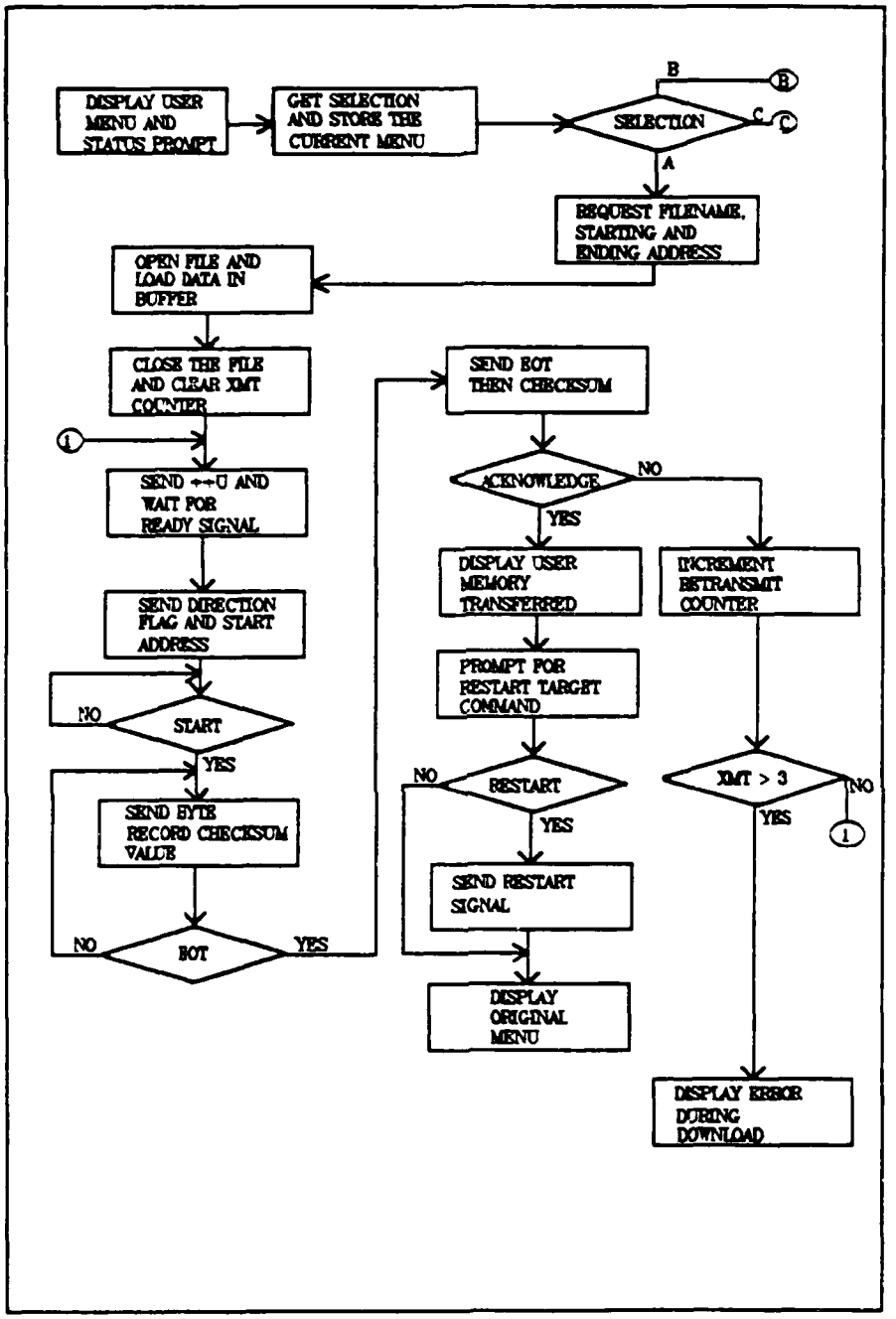

Figure 11a. Z-100 User Memory Flowchart

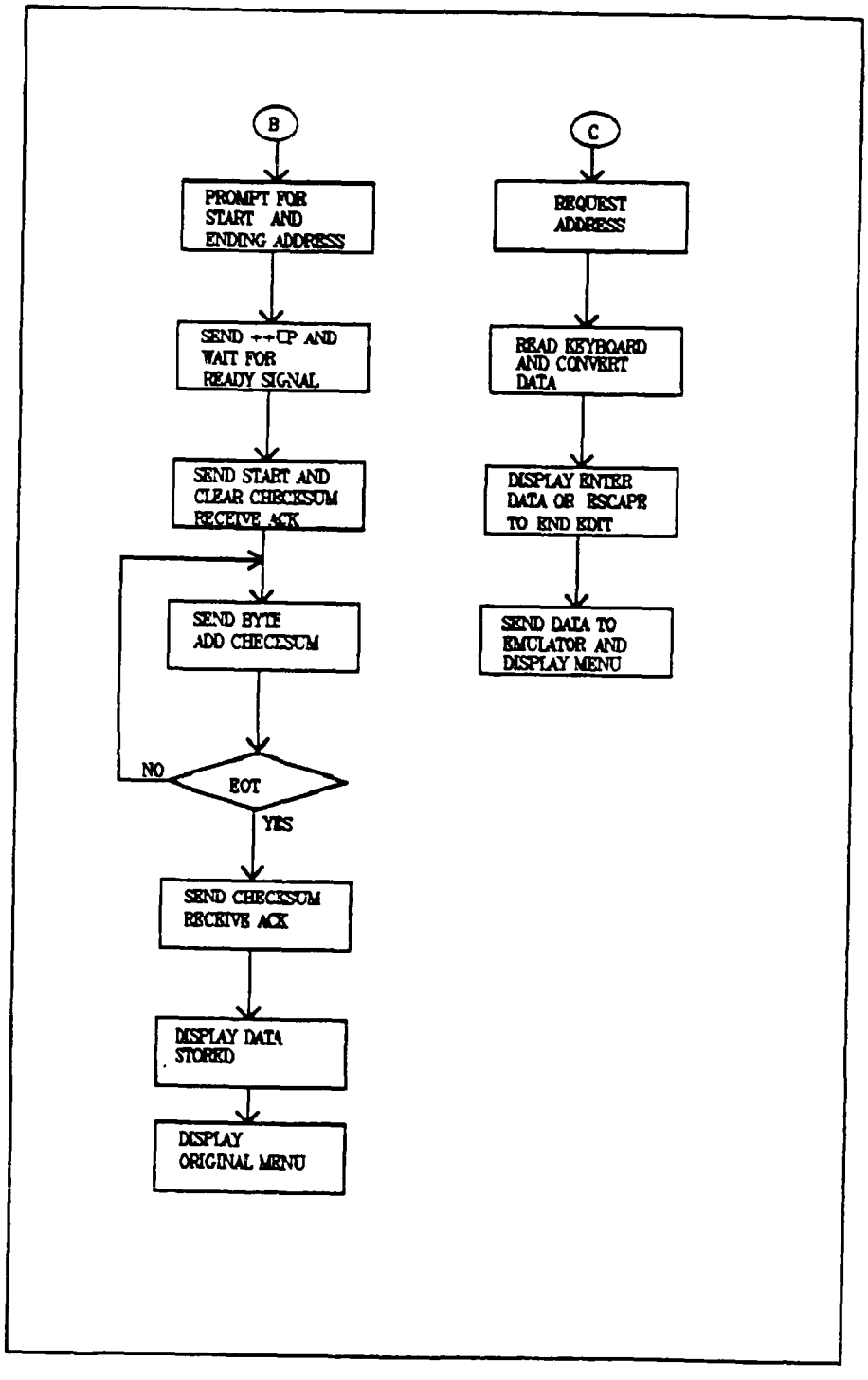

Figure 11b. Z-100 User Memory Flowchart

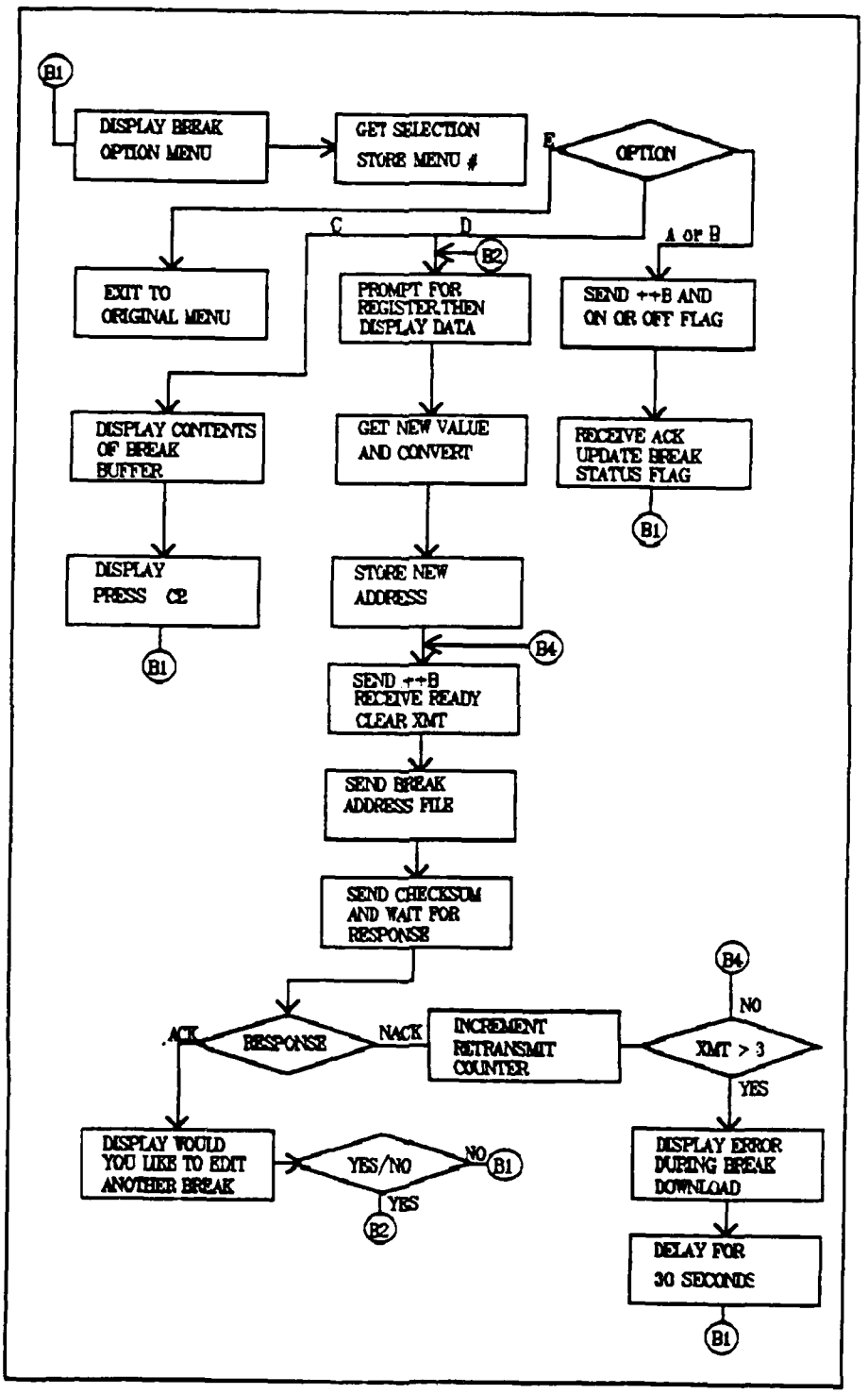

Figure 12. Z-100 Break Control Flowchart

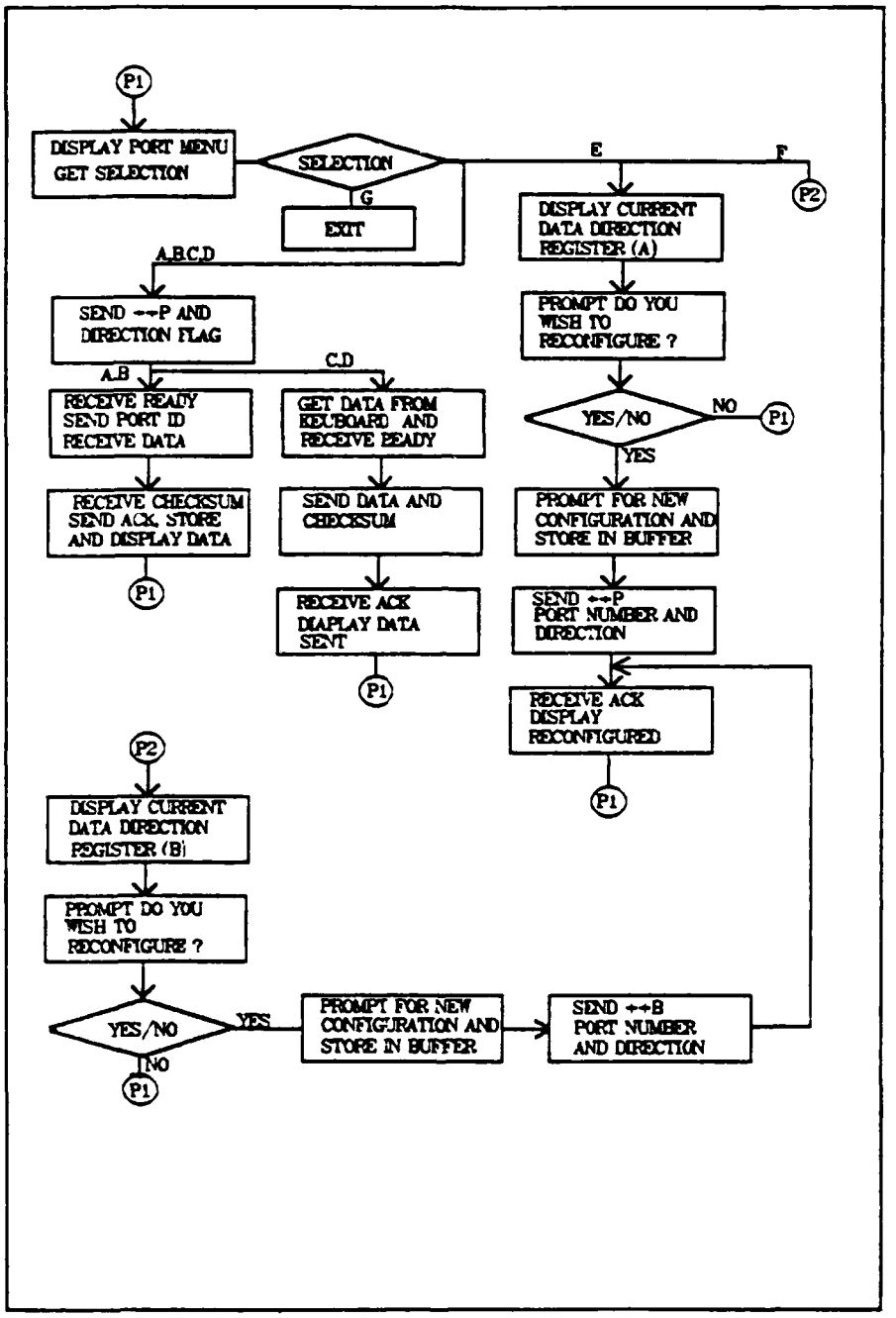

Figure 13. Z-100 Parallel Port Flowchart

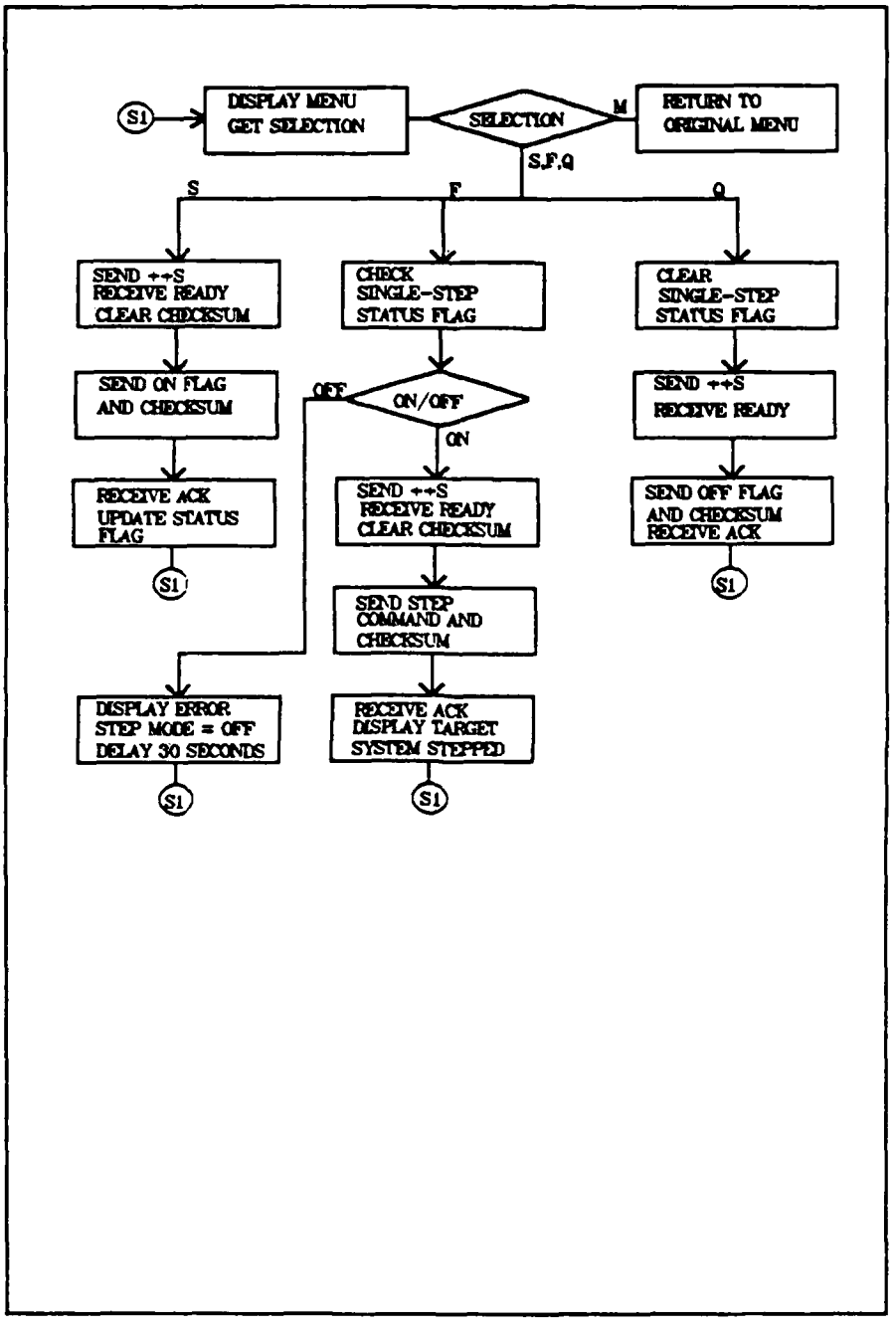

Figure 14. Z-100 Single-step Flowchart

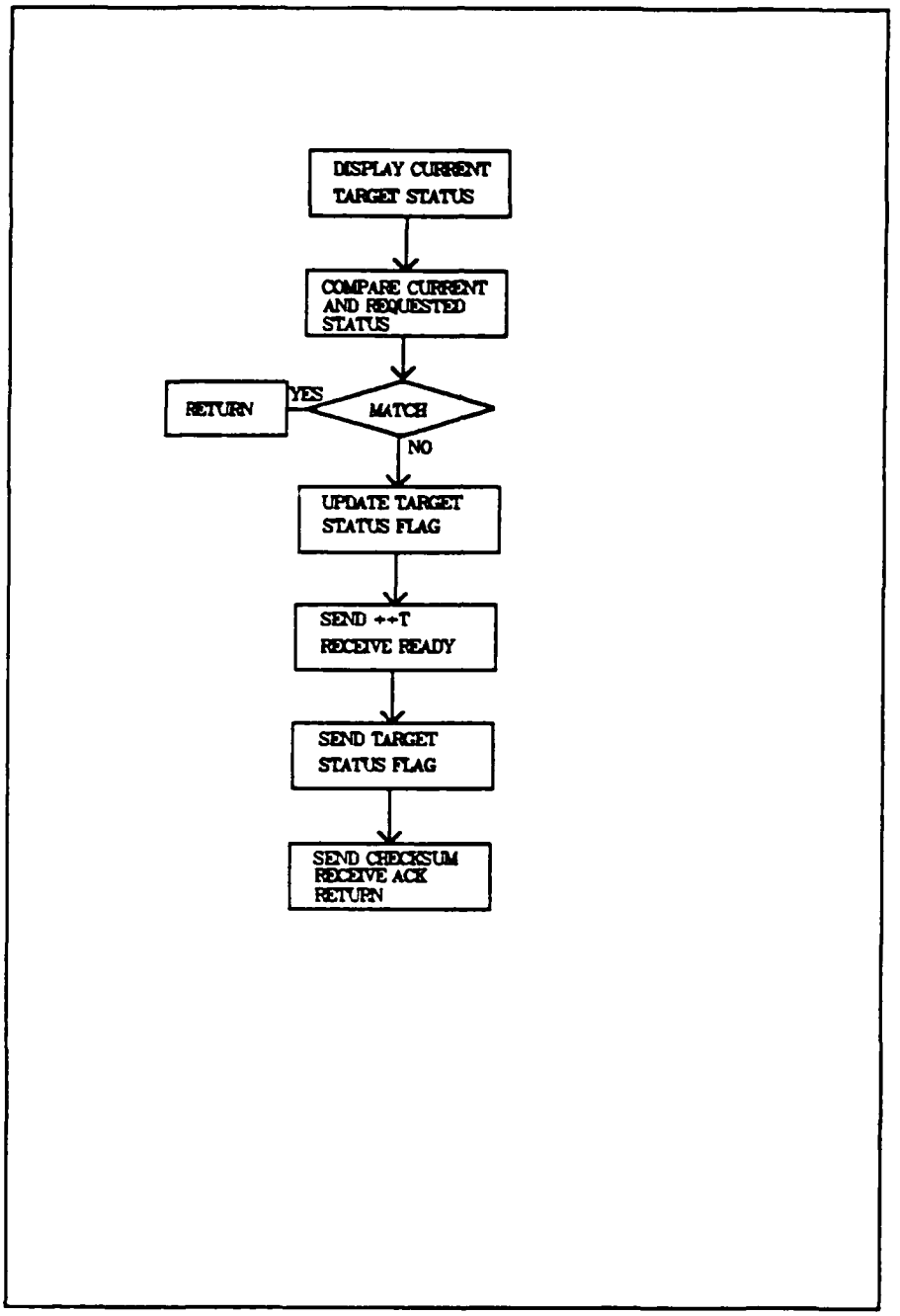

Figure 15. Z-100 Target Control Flowchart

Memory Map

Address	Use
FFFF0 - 80000	Not Used
7FFFF - 70800	Not Used
707FF - 70000	U17 Monitor Rom
6FFFF - 60800	Not Used
607FF - 60000	U18 Target Ram
5FFFF - 50000	Not Used
4FFFF - 40001	Not Used
40000	Enable Stepper Mode
3FFFF - 30001	Not Used
30000	Return Stepper to Freerun
2FFFF - 20002	Not Used
20001	Enable Target Mode
20000	Enable Emulator Mode
1FFFF - 10000	Not Used
0FFFF - 00000	U15 and U16 Emulator Ram

I/O MAP

00F0	Break Reg 5b, U28
00E0	Break Reg 5a, U23
00D0	Break Reg 4b, U27
00C0	Break Reg 4a, U22
00B0	Break Reg 3b, U26
00A0	Break Reg 3a, U21
0090	Break Reg 2b, U25
0080	Break Reg 2a, U20
0070	Break Reg 1b, U24
0060	Break Reg 1a, U19
0050	Serial Data Port
0051	Serial Control Port
0040	Parallel Port A
0041	Parallel Port
0042	Data Direction Reg A
0043	Data Direction Reg B
0030	Interrupt Port 1
0031	Interrupt Port 2
0020	Step Trigger
0010	Break Disable
0000	Break Enable

Table 2. Emulator Address Map

Appendix D: Source Code

```
                    TITLE EMULATOR BOOTUP PROGRAM

page 60,132

;*****************************************************************
; This program initializes emulator hardware and downloads
; the MAIN EMULATOR program
;*****************************************************************

          cseg    segment

          assume  cs:cseg, ds:cseg, ss:cseg, es:cseg

start:    mov     ax,cs
          mov     ds,ax



;*****************************************************************
; The bootup program starts here
;*****************************************************************


;halt target
          mov     ax,4000h          ;set es to page 4
          mov     es,ax
          mov     es:[bx],al        ;set stepper to step mode
                                    ;(i.e halt target)
;set mode
          mov     ax,2000h          ;set es to page 2
          mov     es,ax
          mov     bx,0
          mov     es:[bx],al        ;set mode to emulator

;disable break detectors

          out     Bk_off,al         ;set break flipflop to off

;setup 8259A interrupt controller

          mov     al,13h            ;set ICW1
          out     port0,al
          mov     al,18h            ;set ICW2
          out     port1,al
          mov     al,0dh            ;set ICW4
          out     port1,al

;initialize 8251 serial port
```

```
                                        ;configure 8251A
            mov     al,0                ;put UART into worst
                                        ;case mode
            out     51h,al              ;sync mode, with 2
                                        ;characters
            out     51h,al
            out     51h,al
            mov     al,reset            ;reset UART
            mov     dx,msc_reg
            out     dx,al
            mov     al,set_mode         ;set UART for async mode
            out     dx,al               ;odd parity, 8 bit data
                                        ;baud factor 16
            mov     al,set_control      ;set control register as
            out     dx,al               ;transmit & receive
                                        ;enabled, DTR* enabled,
                                        ;clear error bits,
                                        ;set RTR*, no reset or
                                        ;hunt

;*****************************************************************
; 64k Emulator ram test
;*****************************************************************


            mov     dx,0100h            ;clear xmt counter and set
                                        ;test # to 64k
test_64k:   mov     ax,0000h            ;setup extra segment
            mov     es,ax
            mov     bx,0
            mov     cx,0
loop1:      mov     al,0                ;write all zeroes to ram
            mov     es:[bx],al
            inc     bx
            cmp     bx,0                ;check for all done
            je      chk_read            ;if done then read
            mov     al,0ffh             ;write all 1's to ram
            mov     es:[bx],al
            inc     bx
            cmp     bx,0
            je      chk_read
            jmp     loop1
chk_read:   mov     bx,0
loop2:      mov     al,es:[bx]          ;read zero bytes
            inc     bx
            cmp     al,0                ;test data against zero
            je      no_err              ;skip counter if match is
                                        ;found
            inc     cx
no_err:     mov     al,es:[bx]          ;read all one bytes
            inc     bx
            cmp     al,0ffh             ;test data against ones
            je      counter             ;skip if match is found
```

```
          inc     cx
counter:  cmp     bx,0              ;test for end of memory
          jne     loop2             ;continue if not done

;*****************************************************************
;look at 64k error count and send ack or nack
;*****************************************************************


          mov     ax,0              ;compare total errors
          cmp     cx,ax
          jz      skip
          jmp     error

skip:     mov     di,offset ack_stg
buf_full_1:
          in      al,msc_reg        ;read status register
          test    al,tx_rdy         ;test for empty register
          jz      buf_full_1        ;loop until UART is free

          mov     al,[di]           ;load acknowledge character
                                    ;signals all clear on ram
          out     data_reg,al       ;test to Z100
          inc     di                ;send ++ ack EOT string
          cmp     al,eot
          mov     cx,01ffh
slow:     nop
          loop    slow
          jne     buf_full_1

;*****************************************************************
;test user ram memory
;*****************************************************************


          mov     dx,0200h          ;clear xmt counter and set
                                    ;test # to 2k
test_2k:  mov     ax,6000h          ;setup extra segment
          mov     es,ax
          mov     bx,0
          mov     cx,0
loop3:    mov     al,0              ;write all zeroes to ram
          mov     es:[bx],al
          inc     bx
          cmp     bx,07ffh          ;check for all done
          je      chk_read2         ;if done then read
          mov     al,0ffh           ;write all 1's to ram
          mov     es:[bx],al
          inc     bx
          cmp     bx,07ffh          ;check for all done
          je      chk_read2
          jmp     loop3
chk_read2:
```

```
          mov     bx,0
loop4:    mov     al,es:[bx]      ;read zero bytes
          inc     bx
          cmp     bx,07ffh
          je      eval
          cmp     al,0            ;test data against zero
          je      no_err2         ;skip counter if match is
          inc     cx              ;found
no_err2:  mov     al,es:[bx]      ;read all one bytes
          inc     bx
          cmp     al,0ffh         ;test data against ones
          je      counter2        ;skip if match is found
          inc     cx
counter2: cmp     bx,07ffh        ;test for end of memory
          jl      loop4           ;continue if not done

;*****************************************************************
;look at 2k user ram error count and send ack or nack
;*****************************************************************


eval:     mov     ax,0            ;compare total errors
          cmp     cx,ax
          je      skip4
          jmp     error

skip4:    mov     di,offset ack_stg

buf_full_3:
          in      al,msc_reg      ;read status register
          test    al,tx_rdy       ;test for empty register
          jz      buf_full_3      ;loop until UART is free

          mov     al,[di]         ;load acknowledge character
                                  ;signals all clear on ram
          out     data_reg,al     ;test to Z100
          inc     di              ;send ++ ack EOT string
          cmp     al,eot          ;when end of string get
          mov     cx,01ffh
slow1:    nop
          loop    slow1

          jne     buf_full_3      ;download string

;*****************************************************************
;  input string then test for ++ dw eot, download signal
;*****************************************************************


          mov     ax,0
          mov     es,ax
          mov     bx,0400h
get_char_1:
```

```
            in      al,msc_reg      ;read status register
            test    al,char_rdy     ;check for a character
            jz      get_char_1      ;loop until character available

skip1:      in      al,data_reg     ;read character
            mov     es:[bx],al      ;save Z-100 response
            cmp     al,EOT
            je      done

            inc     bx
            jmp     get_char_1

done:       cld                     ;test for download string
            mov     cx,04
            lea     si,down_ld      ;point to download string
            mov     di,0400h        ;point to received string
            repe    cmpsb           ;compare strings
            cmp     cx,0
            jne     error


;*****************************************************************
; Load the emulator memory
;*****************************************************************


            mov     dx,0300h        ;clear xmt counter and
                                    ;set test # to emulator
emul_code:
            mov     ax,0            ;point to page zero of ram
            mov     es,ax
            mov     bx,0            ;point to 00000h memory
            mov     cl,0            ;use cl as checksum register
emul:       in      al,msc_reg      ;read status register
            test    al,char_rdy     ;check for a character
            jz      emul            ;loop until character
                                    ;available

            in      al,data_reg     ;read character
            cmp     al,ETB          ;check for end of download
            je      emul2           ;if end get checksum
            mov     es:[bx],al      ;store the emulator program
            add     cl,al           ;add to checksum
            inc     bx
            jmp     emul
emul2:      mov     ah,al           ;save first byte of
emul3:      in      al,msc_reg      ;terminator
            test    al,char_rdy
            jz      emul3
            in      al,data_reg
            cmp     al,ETB          ;if yes then end of file
            je      emul1
            mov     es:[bx],ah      ;else save the first data
```

```
          inc     bx
          add     cl,ah           ;update chk_sum
          mov     es:[bx],al      ;save second byte
          add     cl,al           ;update chk_sum
          inc     bx
          jmp     emul
emul1:    in      al,msc_reg      ;read status register
          test    al,char_rdy     ;check for a character
          jz      emul1           ;loop until character
                                  ;available

          in      al,data_reg     ;read character
          cmp     cl,al           ;test checksum
          jne     error

;*****************************************************************
; Send acknowledge to Z-100 for emulator memory download
;*****************************************************************


          mov     di,offset ack_stg  ;load ack character
buf_full_2:
          in      al,msc_reg      ;read status register
          test    al,tx_rdy       ;test for empty register
          jz      buf_full_2      ;loop until UART is free


          mov     al,[di]         ;signals all clear
          out     data_reg,al     ;test to host computer
          inc     di              ;send ++ ack EOT st.ing
          cmp     al,eot
          mov     cx,01ffh
slow2:    nop
          loop    slow2

          jne     buf_full_2

;*****************************************************************
; Pass control to main emulator program
;*****************************************************************


          jmp     far ptr emulator ;pass control to main
                                   ;emulator program

;*****************************************************************
; Return negative acknowledge NACK to Z-100
;*****************************************************************


error:    mov     bx,dx           ;save test number and
                                  ;current xmt count value
          mov     di,offset nack_stg  ;load nack character
```

```
error1:   in      al,msc_reg      ;read status register
          test    al,tx_rdy       ;test for empty register
          jz      error1          ;loop until UART is free

          mov     al,[di]
          out     data_reg,al
          inc     di              ;send ++ nack EOT string
          cmp     al,eot
          mov     cx,01ffh
slow3:    nop
          loop    slow3

          jne     error1

          mov     dx,bx           ;restore test number and
                                  ;current xmt count
          inc     dl
          cmp     dl,03h          ;test for three attempts
          je      fail            ;go to infinite loop
          cmp     dh,01h
          jne     skip2
          jmp     test_64k        ;test 64k sram again
skip2:    cmp     dh,02h
          jne     skip3
          jmp     test_2k
skip3:    cmp     dh,03h
          jmp     emul_code


; infinite loop which waits for a hardware reset

fail:     nop
          jmp     fail

;*****************************************************************
reset     equ     40h             ;reset UART code
err_chk   equ     38h             ;code for PE,OF,FE errors
char_rdy  equ     02h             ;test for full input
                                  ;register
tx_rdy    equ     01h             ;test for empty transmit
                                  ;register
set_mode  equ     4eh             ;code for mode select
set_control equ   37h             ;control parameters
data_reg  equ     50h             ;data register
msc_reg   equ     51h             ;mode,status,control reg
EOT       equ     2fh             ;code for end of file
ETB       equ     1bh             ;end block transfer
no_ack    equ     15h             ;for error during data read
ack       equ     06h             ;acknowledge code
attn      equ     2bh             ;attention definition
nack      equ     15h             ;negative acknowledge code
portB     equ     41h             ;8755 port B
portA     equ     40h             ;8755 port A
```

```
        ddra        equ     42h                 ;8755 port A DDR
        ddrb        equ     43h                 ;8755 port B DDR
        port1       equ     31h                 ;8259 port 1
        port0       equ     30h                 ;8259 port 0
        step        equ     20h                 ;perform step increment
        dwn         equ     09h                 ;down direction flag
        bk_off      equ     10h                 ;disable break registers
        bk_on       equ     00h                 ;enable break incremnet
        ack_stg     db      attn,attn,ack,eot ;ack string
        down_ld     db      attn,attn,dwn,eot ;down next from Z-100
        nack_stg    db      attn,attn,nack,eot ;negative ack string
        emulator    dw      0400h,0000h         ;emulator starting address


                    org     7f0h
                    jmp     far ptr start   ;go to top of program
                    org     7ffh
                    nop
cseg                ends
                    end     start
```

```
                    TITLE EMULATOR main program

; This program must be downloaded tc page zero of the
; emulator address space to be functional.


cseg     segment

         assume  cs:cseg,ds:cseg,ss:cseg,es:cseg

         org     0

         db      0

         org     0400h

start:   mov     ax,cs
           mov   ds,ax

           mov   ax,0effh          ;ss address
           mov   ss,ax             ;set stack
           mov   ax,1000h          ;segment
           mov   sp,ax
                                   ;setup stack
                                   ;vector
           mov   bx,0
           mov   di,0020h          ;type 8 IP
           mov   ax,offset type8
           mov   [di],ax
           mov   di,0022h
           mov   [di],bx           ;cs to 0000

           mov   di,0024h          ;type 9 IP
           mov   ax,offset type9
           mov   [di],ax
           mov   di,0026h
           mov   [di],bx           ;cs to 0000

           mov   di,0028h          ;type A IP
           mov   ax,offset typeA
           mov   [di],ax
           mov   di,002Ah
           mov   [di],bx           ;cs to 0000

           mov   di,002Ch          ;type B IP
           mov   ax,offset typeB
           mov   [di],ax
           mov   di,002Eh
           mov   [di],bx           ;cs to 0000

           mov   di,0030h          ;type C IP
```

```
          mov     ax,offset typeC
          mov     [di],ax
          mov     di,0032h
          mov     [di],bx          ;cs to 0000

          jmp     waiting

;*****************************************************************

reset           equ       40h     ;reset UART code
err_chk         equ       38h     ;code for PE,OF,FE
                                    ;errors
char_rdy        equ       02h     ;test for full input
                                    ;register
tx_rdy          equ       04h     ;test for empty
                                    ;transmit register
set_mode        equ       5eh     ;code for mode select
set_control     equ       37h     ;code for control
                                    ;parameters
data_reg        equ       50h     ;data register
msc_reg         equ       51h
                                    ;mode,status,control
                                    ;register address
EOT             equ       2fh     ;code for end of
                                    ;file
nack            equ       15h     ;code for error
                                    ;during data read
ack             equ       06h     ;acknowledge code
plus            equ       2bh     ;attention definition
rd_err          equ       18h     ;serial read error
                                    ;code
rdy             equ       1ah     ;ready code
portA           equ       40h     ;8755 port A
portB           equ       41h     ;8755 port B
ddra            equ       42h     ;8755 port A DDR
ddrb            equ       43h     ;8755 port B DDR
port1           equ       31h     ;8259 port 1
port0           equ       30h     ;8259 port 0

step            equ       20h     ;perform step
                                    ;increment
down            equ       09h     ;down transfer
                                    ;direction flag
up              equ       08h     ;upload direction
                                    ;flag
on              equ       4eh     ;on flag
off             equ       4fh     ;off flag
bk_off          equ       10h     ;disable break
                                    ;registers
bk_on           equ       00h     ;enable break
                                    ;detectors
```

```
ack_stg         db          2bh,2bh,06h,2fh
                                ;acknowledge string
down_ld         db          2bh,2bh,09h,2fh
                                ;data from Z-100
                                ;next
nack_stg        db          2bh,2bh,15h,2fh
                                ;negative ack string
user_stg        db          2bh,2bh,55h,2fh
                                ;user memory function flag
break_stg       db          2bh,2bh,42h,2fh
                                ;break function flag
step_stg        db          2bh,2bh,53h,2fh
                                ;stepper function flag
port_stg        db          2bh,2bh,50h,2fh
                                ;port function flag
mode_stg        db          2bh,2bh,4dh,2fh
                                ;mode function flag
rdy_stg         db          2bh,2bh,1ah,2fh
                                ;ready response string

emulator        equ         0
target          equ         1
brk_stat        db          ?
                                ;break is initially off

user            db          2049 dup (?)
                                ;user memory buffer
brk_buf         dw          5 dup (?)
                                ;break address buffer
work            db          100 dup (?)
                                ;working buffer area
mode_flg        db          ?       ;mode flag
                                ;00=emulator, ff=target
step_flg        db          ?       ;flag for step or freerun
                                ;00=freerun, ff=stepper
u_start         dw          ?       ;starting address
                                ;of user memory download
byte_num        dw          ?       ;byte count storage
temp_buf        db          10 dup (?)
                                ;user read buffer
chk_sum         db          ?       ;checksum count buffer
xmt             db          ?       ;retransmit count buffer
int_brk_num     db          ?       ;interrupt type buffer

break_1a        equ         60h     ;register 1 part A
break_1b        equ         70h     ;register 1 part B
break_2a        equ         80h     ;register 2 part A
break_2b        equ         90h     ;register 2 part B
break_3a        equ         0A0h    ;register 3 part A
break_3b        equ         0B0h    ;register 3 part B
break_4a        equ         0C0h    ;register 4 part A
break_4b        equ         0D0h    ;register 4 part B
break_5a        equ         0E0h    ;register 5 part A
```

```
break_5b        equ         0F0h    ;register 5 part B

;*****************************************************************
;
; This macro compares the contents of two strings. The source
; and destination offset must be provide and the CX register
; must contain the string length. For a match the AX register
; will return zero. Any other value means no match.
;
;*****************************************************************

compare    macro    string1,string2

           push     si
           push     di
           push     cx

           cld
           mov      cx,4
           lea      si,string1      ;point to input string
           lea      di,string2      ;point to test string
           repe     cmpsb           ;compare strings
           mov      ax,cx           ;set al to the value of
                                    ;cx. If cx equal zero
                                    ;strings match
           pop      cx              ;restore original string
           pop      di              ;length
           pop      si

           endm

;*****************************************************************
;
; This macro sets the user memory U-18 to either the emulator
; access or target access.
;
;*****************************************************************


u18_to     macro    system

           push     ax

           call     set_step        ;halt the target
           mov      al,system
           mov      mode_flg,al     ;set mode flag
           call     mode
           pop      ax

           endm
```

```
;*****************************************************************
;
; This macro clears the memory location designated as "slot".
;
;*****************************************************************

clear     macro   slot

          mov     slot,0

          endm

;*****************************************************************
;
; This macro transmits the designated flag to the Z-100.
;
;*****************************************************************

send      macro   flag_name

          lea     di,flag_name
          call    send_flag

          endm



;*****************************************************************
;
; This procedure sets and resets the mode flipflop.
; The mode is determined by the value in the mode flag,
; which must be set prior to calling this procedure.
;
;*****************************************************************

mode      proc    near

          push    ax
          push    bx
          mov     bx,0              ;clear bx
          mov     ax,2000h          ;set es to page 2
          mov     es,ax
          mov     bl,mode_flg       ;set IP
          mov     es:[bx],al        ;set mode to emulator bl=0h
                                    ;or target bl=01h
          pop     bx
          pop     ax

          ret
mode      endp

;*****************************************************************
```

```
;
; This procedure reads in a string of characters from the
; serial port and stores it in the working buffer. The string
; length is returned in the CX register.
;
;*****************************************************************

get_flag  proc    near

          push    di

          mov     di,offset work  ;point to working buffer
          mov     cx,0            ;clear character counter

more1:    call    serial_rd       ;read serial port
          mov     [di],al         ;save data
          cmp     al,eot          ;check for end of string
          je      all_done
          inc     di
          inc     cx              ;at the end of the string
          jmp     more1           ;cx contains the count

all_done: pop     di
          ret
get_flag  endp


;*****************************************************************
;
; This procedure sends a string of characters from the
; serial port to the Z-100. The string offset must be
; in the DI register when the call is made.
;
;*****************************************************************

send_flag proc    near

          push    ax

continue: mov     al,[di]         ;load a byte of data
          call    serial_wr       ;transmit the data
          cmp     al,eot          ;check for end of string
          je      return
          inc     di              ;point to next byte
          jmp     continue        ;repeat process
return:   pop     ax

          ret
send_flag endp


;*****************************************************************
;
```

```
; This procedure sets the emulator to the single-step mode
; and updates the stepper status flag
;
;*****************************************************************

set_step  proc    near

          push    ax
          push    bx
          mov     bx,0
          mov     ax,4000h        ;set es to page 4
          mov     es,ax
          mov     es:[bx],al      ;set stepper to step mode
          mov     bl,0ffh
          mov     step_flg,bl     ;set step flag to ff = step
          pop     bx
          pop     ax

          ret
set_step  endp


;*****************************************************************
;
; This procedure sets the emulator to the freerun mode and
; updates the stepper status flag
;
;*****************************************************************

freerun   proc    near

          push    ax
          push    bx
          mov     bx,0
          mov     ax,3000h        ;set es to page 3
          mov     es,ax
          mov     es:[bx],al      ;set stepper to free run
          mov     step_flg,bl     ;set flag to 00 = freerun
          pop     bx
          pop     ax

          ret
freerun   endp


;*****************************************************************
;
; This procedure simply pulses the target ready line.
;
;*****************************************************************

inc_step  proc    near
```

```
          out     step,al          ;pulses the target step
                                   ;circuit

          ret
inc_step  endp

;*****************************************************************
;
; This procedure disables the break detectors and
; updates the break status flag
;
;*****************************************************************

brk_off   proc    near

          push    ax
          out     Bk_off,al        ;set break flipflop to off
          mov     brk_stat,off     ;store break status
          pop     ax

          ret
brk_off   endp


;*****************************************************************
;
; This procedure enables the break detectors and
; updates the break status flag.
;
;*****************************************************************

brk_on    proc    near

          out     Bk_on,al         ;set break flipflop to on
          mov     brk_stat,on      ;store break status

          ret
brk_on    endp


;*****************************************************************
;
; This procedure moves the data pointed to by "Start" and the
; amount specified by "Byte" from the user buffer to the
; target ram in the same locations. Start and byte# must be
; updated prior to the call.
;
;*****************************************************************

load      proc    near

          push    ax
          push    bx
```

```
            push    cx
            push    dx

            cld
            mov     ax,6000h        ;point to user ram
            mov     es,ax
            mov     di,u_start      ;set starting address of
                                    ;memory in target ram
            mov     cx,byte_num     ;number of bytes to move
            mov     si,offset user  ;point to starting address
            add     si,u_start      ;of user buffer data
            repe    movsb           ;transfer data

            pop     dx
            pop     cx
            pop     bx
            pop     ax

            ret
load        endp


;*****************************************************************
;
; This procedure retrieves a specified of number bytes from
; user memory and places them in the temporary buffer
;
;*****************************************************************

user_rd     proc    near

            push    ax
            push    di
            push    bx
            push    cx

            clear   chk_sum
            mov     ax,6000h        ;point to user memory
            mov     es,ax
            mov     cx,0
            mov     cx,byte_num     ;load byte counter
            mov     bx,offset temp_buf ;setup temporary buffer
            mov     di,u_start      ;at the specified location
rd_loop:    mov     al,es:[di]      ;read the byte
            mov     [bx],al         ;put data in the holding
                                    ;buffer
            add     chk_sum,al      ;update checksum
            inc     di
            inc     bx
            loop    rd_loop

            inc     bx              ;tack on the eot to buffer
            mov     al,eot
```

```
            mov     [bx],al
            pop     cx
            pop     bx
            pop     di
            pop     ax

            ret
user_rd     endp

;*****************************************************************
;
; This procedure reads the 8755 ports A and B, and returns
; the results in the AL register. Which port to read is
; placed in the DX register prior to the call.
;
;*****************************************************************

in_8755     proc    near

            push    ax

            u18_to  emulator        ;halt target
            in      al,dx           ;read 8755 port
            mov     chk_sum,al      ;save data in checksum
                                    ;because in this case
                                    ;data = checksum
            u18_to  target          ;restart target
            call    freerun

            pop     ax

            ret
in_8755     endp


;*****************************************************************
;
; This procedure performs output operations to 8755 data and
; data direction ports. The port to access is loaded in DX
; and the data out is in AL prior to the call.
;
;*****************************************************************

out_8755    proc    near

            u18_to  emulator        ;halt target
            out     dx,al           ;write to 8755 port
            u18_to  target          ;restart target
            call    freerun

            ret
out_8755    endp
```

```
;*****************************************************************
;
;  This procedure performs a serial read of port 50h. The
;  data is returned in the AL register. Each data read is
;  preceded by a status check of the receiver.
;
;*****************************************************************

serial_rd proc    near

          push    dx

get_char_1:
          mov     dx,msc_reg
          in      al,dx             ;read status register
          test    al,char_rdy       ;check for a character
          jz      get_char_1        ;loop until character
                                    ;available

          test    al,err_chk        ;check for errors
          jnz     exit
          mov     dx,data_reg       ;read data byte and return
          in      al,dx             ;in al register
          jmp     exit1

exit:     mov     al,rd_err         ;set read error
                                    ;flag
          call    serial_wr
          jmp     get_char_1

exit1:    pop     dx
          ret
serial_rd endp


;*****************************************************************
;
; This procedure performs a serial output to port 50h. The
; data to be transmitted must be in the AL register prior
; to the call. Each data output is preceded by a status check
; of the UART to ensure its availability.
;
;*****************************************************************

serial_wr proc    near

          push    ax                ;save data

buf_full_1:
          in      al,msc_reg        ;read status register
          test    al,tx_rdy         ;test for empty transmit
                                    ;register
```

```
          jz      buf_full_1      ;loop until UART is free
          pop     ax              ;restore data
          out     data_reg,al     ;pass the data to Z100

          ret
serial_wr endp


;*****************************************************************
;
;  This procedure loads the break detection registers. The
;  break detect address buffer must be loaded prior to the
;  call.
;
;*****************************************************************

brk_addr  proc    near

          push    di
          push    dx
          push    cx

          mov     cx,0Ah
          mov     di,offset brk_buf ;point to break address
                                    ;buffer
          mov     dx,60h            ;load first break address

next1:    mov     al,[di]           ;output one byte of
                                    ;address
          out     dx,al             ;at a time. Two bytes are
                                    ;required for each address
          inc     di
          add     dx,10h            ;move to next byte
          loop    next1

          pop     cx
          pop     dx
          pop     di

          ret
brk_addr  endp

;*****************************************************************
;
; This procedure handles all emulator functions involving
; the user ram U18. The user code may be downloaded or the
; user may upload up to ten consecutive bytes.
;
;*****************************************************************

user_pro  proc    near

          send    rdy_stg           ;send the ready string
```

```
            call    get_flag        ;get the direction flag
                                    ;for next operation
            compare work,down_ld    ;if equal go to upload
            cmp     al,0
            je      leap
            jmp     up_data         ;else download was
                                    ;requested
leap:       call    serial_rd       ;get high byte of user
                                    ;start address
            mov     bh,al           ;move high byte to AH
            call    serial_rd
            mov     bl,al           ;store the starting address
            mov     u_start,bx
            clear   xmt             ;clear retransmit counter

user_loop:
            clear   chk_sum         ;clear check sum
            call    serial_rd       ;get byte count
            mov     bh,al
            call    serial_rd
            mov     bl,al
            mov     byte_num,bx
            xor     bx,bx
            send    rdy_stg

            lea     di,user         ;point to top of user
                                    ;buffer
            add     di,u_start      ;point to starting location
                                    ;of desired data move
continue1:
            call    serial_rd       ;get the data
            mov     [di],al         ;store the data in the
                                    ;user buffer
            add     chk_sum,al      ;update check sum
            inc     bx
            cmp     bx,07ffh
            je      no_more         ;if all done test chk_sum
            inc     di
            jmp     continue1

no_more:    call    serial_rd       ;get checksum
            jmp     johnny
            cmp     chk_sum,al      ;test checksum
            jne     retrans

johnny:     send    ack_stg         ;send Z-100 the acknowledge
            u18_to  emulator        ;allow access to user ram

            call    load            ;move data to user memory
            u18_to  target          ;restart target
            call    freerun
            jmp     exit_user_pro   ;return to waiting loop
```

```
retrans:  add     xmt,1           ;increment xmt counter
          cmp     xmt,3           ;if three tries exit
          je      exit_user_pro
          send    nack_stg        ;inform Z-100 bad data
          jmp     user_loop       ;was received


up_data:  call    serial_rd       ;get high byte of user
                                  ;start address
          mov     bx,offset u_start
          mov     [bx]+1,al       ;move high byte to AH
          call    serial_rd
          mov     [bx],al         ;store the starting address
          call    serial_rd       ;get number of bytes
                                  ;desired
          mov     byte_num,ax     ;and store it

          clear   xmt             ;clear retransmit counter

          u18_to  emulator        ;grants emulator access

          call    user_rd         ;get the specified bytes
                                  ;and store in buffer
          call    freerun         ;restart target
retrans1: send    ack_stg
          send    temp_buf        ;upload data
          mov     al,chk_sum      ;send check sum
          call    serial_wr
          call    get_flag        ;read Z-100 response
          compare work,ack_stg    ;if not ack then error
          cmp     ax,0
          jne     skip_it
          jmp     exit_user_pro   ;transaction complete, exit

skip_it:  add     xmt,1           ;increment xmt counter
          cmp     xmt,3           ;if three tries exit
          je      exit_user_pro
          jmp     retrans1        ;was received

exit_user_pro:    ret
user_pro  endp

;*****************************************************************
;
; This procedure enables or disables the single-step
; function and updates the single-step status flag. It
; also steps the target upon demand.
;
;*****************************************************************

step_pro  proc     near
```

```
            send    rdy_stg         ;send ready
            call    serial_rd       ;get desired mode
            cmp     al,on           ;check for turn on
            jne     skip_over
            call    set_step        ;set to single-step
            jmp     exit_step_pro
skip_over:
            cmp     al,off          ;check for turn off
            jne     skip_over1
            call    freerun         ;set to freerun
            jmp     exit_step_pro
skip_over1:
            cmp     al,step         ;check for step
            jne     out_pro
            cmp     step_flg,on     ;makes sure in single_step
            jne     step_nack
            call    inc_step        ;step the target
            jmp     exit_step_pro
step_nack:
            send    nack_stg        ;return nack signal
            jmp     out_pro
exit_step_pro:
            send    ack_stg         ;return ack signal

out_pro:    ret
step_pro    endp


;*****************************************************************
;
; This procedure reads or writes the 8755 I/O ports and
; writes to the data direction ports. The success or
; failure of each transaction is returned to the Z-100.
;
;*****************************************************************

port_pro    proc    near

            send    rdy_stg         ;return ready
            call    serial_rd
            cmp     al,up           ;test direction flag
            jne     wr_port         ;branch on down flag

            call    serial_rd       ;get port number
            cmp     al,porta        ;test for port A read
            jne     read_portb
            mov     dx,porta
            jmp     read_porta
read_portb:
            mov     dx,portb        ;point to port B
read_porta:
            call    in_8755         ;read port A data
```

```
            clear   xmt
retrans3:   send    rdy_stg
            mov     al,chk_sum      ;checksum = data here
            call    serial_wr       ;send data to Z-100
            mov     al,eot          ;send eot
            call    serial_wr
            mov     al,chk_sum      ;send checksum
            call    serial_wr
            call    get_flag        ;get response
            compare work,ack_stg
            cmp     ax,0
            jne     bad_try
            jmp     exit_port_pro   ;data transferred
bad_try:    add     xmt,1           ;increment counter
            cmp     xmt,3
            je      exit_port_pro
            jmp     retrans3

wr_port:    clear   xmt
retrans4:   clear   chk_sum
            send    rdy_stg         ;return ready
            mov     di,offset work  ;point to working buffer
get_more:   call    serial_rd       ;read serial port
            mov     [di],al         ;save data
            cmp     al,eot          ;check for end of string
            je      thats_it
            add     chk_sum,al
            inc     di
            jmp     get_more
thats_it:   call    serial_rd       ;get checksum
            cmp     al,chk_sum
            jne     too_bad

            mov     di,offset work
            mov     dx,0            ;clear dx
            mov     dl,[di]         ;point to port number
            inc     di
            mov     al,[di]         ;load data byte
            call    out_8755        ;send data byte
            send    ack_stg
            jmp     exit_port_pro

too_bad:    send    nack_stg        ;return nack signal
            add     xmt,1           ;increment counter
            cmp     xmt,3
            je      exit_port_pro
            jmp     retrans4

exit_port_pro:  ret

port_pro    endp
```

```
;*****************************************************************
;
; This procedure enables or disables the emulator break
; detection circuits. It also downloads the break address
; file from the Z-100 and loads it into the break registers.
;
;*****************************************************************

break_pro proc    near

          push    ax
          push    di

          send    rdy_stg           ;return ready
          call    serial_rd         ;get instruction
          cmp     al,on             ;check for turn break on
          jne     chk_off
          call    brk_on            ;turn break on
          send    ack_stg           ;return ack signal
          jmp     exit_break_pro
chk_off:  cmp     al,off            ;check for turn break off
          jne     load_brk
          call    brk_off           ;turn break off
          send    ack_stg           ;return ack signal
          jmp     exit_break_pro
load_brk: mov     xmt,0
          clear   chk_sum           ;get break addresses
load_brk_1:
          send    rdy_stg
          mov     di,offset brk_buf  ;setup break address
          mov     cx,10             ;buffer
next_brk: call    serial_rd         ;get byte
          mov     [di],al           ;store data
          add     chk_sum,al
          inc     di
          loop    next_brk
all_here: call    serial_rd         ;get checksum
          cmp     chk_sum,al        ;and test
          jne     no_good

          call    brk_off           ;disable break circuits
          call    brk_addr          ;load break registers
          call    brk_on            ;enable break circuits

          send    ack_stg           ;return ack signal
          jmp     exit_break_pro
no_good:  send    nack_stg          ;return nack signal
          clear   chk_sum
          add     xmt,1
          cmp     xmt,3
          jne     load_brk_1
exit_break_pro:
```

```
          pop     di
          pop     ax
          ret

break_pro endp


;*****************************************************************
;
; This procedure processes the emulator break detect
; interrupt response. The break address type number  is
; passed to the Z-100. The break condition is cleared and
; the break circuits restarted on command.
;
;*****************************************************************

brk_handler proc near

          clear   xmt
retrans11:
          send    break_stg
          call    get_flag
          compare work,ack_stg    ;get the ack signal
          cmp     ax,0
          jne     retrans11

          mov     al,int_brk_num  ;get the type number
          mov     chk_sum,al
          call    serial_wr       ;send break type
          mov     al,eot
          call    serial_wr       ;send eot
          mov     al,chk_sum
          call    serial_wr       ;send checksum

          call    get_flag
          compare work,ack_stg    ;get the ack signal
          cmp     ax,0
          je      done11

          add     xmt,1
          cmp     xmt,3
          jle     retrans11
done11:   ret

brk_handler endp


;*****************************************************************
;*****************************************************************
;
; This is the main waiting loop of the emulator code. Its
; function is to monitor the Z-100 and call the approriate
; subroutine to handle a given request.
```

```
;
;*****************************************************************

waiting:  mov     di,offset work  ;point to working buffer
          mov     cx,0            ;clear character counter

more_w:   call    serial_rd       ;read serial port
          mov     [di],al         ;save data
          cmp     al,eot          ;check for end of string
          je      all_in
          inc     di
          inc     cx              ;at the end of the string
          jmp     more_w          ;cx contains the count


all_in:   compare work,user_stg   ;determine which instr
          cmp     al,0            ;was received
          jne     skip1
          call    user_pro        ;pass control to user
          jmp     waiting         ;memory module, the return
                                  ;main waiting loop

skip1:    compare work,break_stg
          cmp     al,0
          jne     skip2
          call    break_pro       ;pass control to
                                  ;break module
          jmp     waiting

skip2:    compare work,step_stg
          cmp     al,0
          jne     skip3
          call    step_pro        ;pass control to stepper
          jmp     waiting         ;control module

skip3:    compare work,port_stg
          cmp     al,0
          jmp     waiting
          call    port_pro        ;pass control to port
          jmp     waiting         ;control module


type8:    mov     int_brk_num,08h  ;set pointer
          call    brk_handler
          iret


type9:    mov     int_brk_num,09h  ;set pointer
          call    brk_handler
          iret
```

```
typeA:    mov     int_brk_num,0Ah  ;set pointer
          call    brk_handler
          iret

typeB:    mov     int_brk_num,0Bh  ;set pointer
          call    brk_handler
          iret

typeC:    mov     int_brk_num,0Ch  ;set pointer
          call    brk_handler
          iret

cseg      ends

          end
```

```
                TITLE EMULATOR Z100 CONTROL PROGRAM

        page 60,132

        dseg        segment

        ;define varibles and labels
        ;*****************************************************************

        cls         db      26 dup (13,10), "$"

        xlat_ascii_char_2_hex_value  label byte  ;hex lookup table

                    db      48 dup (0ffh),0,1,2,3,4,5,6,7,8,9
                    db      7 dup (0ffh),0ah,0bh,0ch,0dh,0eh,0fh
                    db      26 dup (0ffh),0ah,0bh,0ch,0dh,0eh,0fh
                    db      153 dup (0ffh)

      ascii_table label byte                ;ascii lookup table

                    db      30h,31h,32h,33h,34h,35h,36h,37h,38h,39h
                    db      41h,42h,43h,44h,45h,46h
                    db      31h,15 dup (?),32h,15 dup (?),33h
                    db      15 dup (?),34h,15 dup (?),35h,15 dup (?)
                    db      36h,15 dup (?),37h,15 dup (?),38h
                    db      15 dup (?),39h,15 dup (?),41h,15 dup (?)
                    db      42h,15 dup (?),43h,15 dup (?),44h
                    db      15 dup (?),45h,15 dup (?),46h

        cr          equ     13
        lf          equ     10
        EOT         equ     2fh             ;code for end of file
        ETB         equ     1bh             ;end of block transfer
        nack        equ     15h             ;code for error
                                            ;during data read
        ack         equ     06h             ;acknowledge code
        plus        equ     2bh             ;attention definition
        err_flg     equ     18h             ;error code
        rdy         equ     1ah             ;ready code
        portA       equ     40h             ;8755 port A
        portB       equ     41h             ;8755 port B
        ddra        equ     42h             ;8755 port A DDR
        ddrb        equ     43h             ;8755 port B DDR

        step        equ     20h             ;perform step increment
        down        equ     09h             ;down transfer direction
                                            ;flag
        up          equ     08h             ;upload direction flag
        on          equ     4eh             ;on flag
        off         equ     4fh             ;off flag
        bk_off      equ     10h             ;disable break registers
        bk_on       equ     00h             ;enable break detectors
```

```
ack_stg    db      2bh,2bh,06h,2fh
                                    ;acknowledge string
down_ld    db      2bh,2bh,09h,2fh
                                    ;data from Z-100 next
nack_stg   db      2bh,2bh,15h,2fh
                                    ;negative ack string
user_stg   db      2bh,2bh,55h,2fh
                                    ;user memory function flag
break_stg  db      2bh,2bh,42h,2fh
                                    ;break function flag
step_stg   db      2bh,2bh,53h,2fh
                                    ;stepper function flag
port_stg   db      2bh,2bh,50h,2fh
                                    ;port function flag
mode_stg   db      2bh,2bh,4dh,2fh
                                    ;mode function flag
rdy_stg    db      2bh,2bh,1ah,2fh
                                    ;ready response string

emulator   equ     0h              ;emulator identification
target     equ     01h             ;target identification

user_code  label   byte            ;user filename buffer
           db      (?)             ;maximum length
char_ct    db      (?)             ;actual length
user_file  db      15 dup (?)      ;filename
                                    ;buffer for user filename

filnam     db      'EMULATOR.BIN',00
                                    ;filename for emulator
user       db      2048 dup (?)
           db      (?)             ;user memory buffer
brk_buf    db      10 dup (?)
                                    ;break address buffer
work       dw      ?               ;working buffer area
mode_flg   db      ?
                                    ;flag for target or
                                    ;emulator
break_flg  db      ?
                                    ;break circuit status flag
                                    ;on or off
step_flg   db      ?               ;on or off
                                    ;off = freerun, on =
                                    ;stepper
u_start1   dw      ?
u_start    dw      ?               ;starting address of user
                                    ;memory download
u_stop     dw      ?               ;stop address of user
                                    ;memory download
byte_num   dw      ?               ;byte count storage

temp_buf   label   byte            ;temporary
```

```
          db      (?)               ;user read buffer
bytes_in  db      (?)
byte_1    db      10 dup (?)

view_buf  db      5 dup (" ")
view_addr db      3 dup (?)
          db      10 dup (" ")
view_data db      2 dup (?)
          db      cr,lf,"$"
msg_buf   db      10 dup (?)
                                    ;string control buffer
porta_sto db      (?)               ;parallel port holding
portb_sto db      (?)               ;buffers
ddra_sto  db      (?)
ddrb_sto  db      (?)

chk_sum   db      ?                 ;chksum buffer
xmt       db      ?                 ;retransmit count buffer
addr_hold dw      ?                 ;temporary buffer

break_1   dw      ?                 ;register 1 part A
                                    ;register 1 part B
break_2   dw      ?                 ;register 2 part A
                                    ;register 2 part B
break_3   dw      ?                 ;register 3 part A
                                    ;register 3 part B
break_4   dw      ?                 ;register 4 part A
                                    ;register 4 part B
break_5   dw      ?                 ;register 5 part A
                                    ;register 5 part B

; message definition area

;*****************************************************************


disk_err  db      "Disk read error: Emulator.bin "
          db      "must be on the default drive",cr,lf,"$"

port_err  db      "Port circuit did not respond",cr,lf,"$"

chg_ddr   db      cr,lf,"Change DDR (y/n): ","$"

step_error  db    "Step circuit did not respond",cr,lf,"$"

bad_step  db      "Stepper must be enabled",cr,lf,"$"

stepped   db      "Step Action Complete",cr,lf,"$"

ddr_out   db      "DDR downloaded",cr,lf,"$"

port_cont db      cr,lf,lf,"Selected port content is: "
port_view db      (?)
```

```
          db      cr,lf,"$"
port_data db      cr,lf,"Please enter the port data :","$"
port_hold db      ?

brk_head  db      cr,lf,lf,"Current break address listing"
          db      cr,lf,cr,lf
          db      "Break Number",10 dup (" ")
          db      "Break Address",cr,lf,cr,lf,"$"

view_brk  label   byte              ;break display format

          db      5 dup (" "),"1",20 dup (" ")
brk_addr1 db      3 dup (?),cr,lf
          db      5 dup (" "),"2",20 dup (" ")
brk_addr2 db      3 dup (?),cr,lf
          db      5 dup (" "),"3",20 dup (" ")
brk_addr3 db      3 dup (?),cr,lf
          db      5 dup (" "),"4",20 dup (" ")
brk_addr4 db      3 dup (?),cr,lf
          db      5 dup (" "),"5",20 dup (" ")
brk_addr5 db      3 dup (?),cr,lf,"$"

edit_data db      "Enter data for location "
location  db      3 dup (?)
          db      " : ","$"

view_head db      "User memory buffer contents",cr,lf,cr,lf
          db      "     Address      Data",cr,lf,"$"

view_nr   db   cr,lf,"Please enter the last address to view"
          db      cr,lf,"Note: Maximum length is 20 bytes!!"
          db      cr,lf," : ","$"

edit_nr   db   cr,lf,"Please enter the last address to edit"
          db      cr,lf,"Note: Maximum length is 10 bytes!!"
          db      cr,lf," : ","$"

dw_err    db      "ERROR :: During memory download !!!"
          db      cr,lf,"$"

dw_brk_err db     "ERROR :: During break download !!!"
          db      cr,lf,"$"

start_addr db     cr,lf,"Please enter the starting "
          db      "address as (XXXh): ","$"

stop_addr db      cr,lf,"Please enter the ending "
          db      "address as (XXXh): ","$"

file      db      cr,lf,lf,10 dup (" ")
          db      "Please enter the name of your file,"
          db      cr,lf,10 dup (" ")
          db      "this must be a binary file: ","$"
```

```
data_out  db      "User Memory has been downloaded",13,10,"$"

bad_sel   db      cr,lf,10 dup (" ")
          db      "Invalid Selection : Enter <return>","$"

enter_3   db      cr,lf,"Enter 3 hexidecimal digits only!"
          db      cr,lf,"$"

enter_2   db      cr,lf,"Enter 2 hexidecimal digits only!"
          db      cr,lf,"$"

standby   db      "    STANDBY EMULATOR INITIALIZING !!!"
          db      cr,lf,"$"
sram_err  db      "EMULATOR RAM ERROR: RESET EMULATOR",cr,lf
          db      "$"

setup_menu db     10 dup (" "),"SETUP MENU",cr,lf,cr,lf
          db      10 dup (" "),"a. Pass User Memory",cr,lf
          db      10 dup (" "),"b. Set Break Address",cr,lf
          db      10 dup (" "),"c. Configure Ports",cr,lf
          db      10 dup (" "),"d. Main Menu",cr,lf,"$"

main_menu db      10 dup (" "),"MAIN MENU",cr,lf,cr,lf
          db      10 dup (" "),"a. Start/Stop Target",cr,lf
          db      10 dup (" "),"b. Enable/Disable Break",cr,lf
          db      10 dup (" "),"c. Change Break Address",cr,lf
          db      10 dup (" "),"d. Access Single-step",cr,lf
          db      10 dup (" "),"e. Access User Memory",cr,lf
          db      10 dup (" "),"f. Access Parallel Ports",cr,lf
          db      10 dup (" "),"g. Exit to Dos",cr,lf,"$"

user_menu db      10 dup (" "), "User Memory Menu",cr,lf,cr,lf
          db      10 dup (" "),"a. Download User Memory",cr,lf
          db      10 dup (" "),"b. View User Memory",cr,lf
          db      10 dup (" "),"c. Edit User Memory",cr,lf
          db      10 dup (" "),"d. Exit ",cr,lf,lf,"$"

select    db      cr,lf,10 dup (" "),"Make a selection: ","$"

break_menu db     10 dup (" "),"BREAK MENU",cr,lf,cr,lf
          db      10 dup (" "),"a. Enable Break",cr,lf
          db      10 dup (" "),"b. Disable Break",cr,lf
          db      10 dup (" "),"c. View Break Address",cr,lf
          db      10 dup (" "),"d. Change Break Address",cr,lf
          db      10 dup (" "),"e. Exit ",cr,lf,lf,"$"

sel_brk_reg  db   cr,lf,lf,"Select break register (1-5): "
             db   "$"
new_brk_addr db   "Enter new break address: ","$"
more_brk     db   "Change another address (Y/N): ","$"

brk_now      db   "Break address "
```

```
brk_sel      db   ?
             db   " = "
brk_sel_data db   3 dup (?),cr,lf,"$"

port_menu db      10 dup (" "),"PORT MENU",cr,lf,cr,lf
          db      10 dup (" "),"a. Read port A",cr,lf
          db      10 dup (" "),"b. Read port B",cr,lf
          db      10 dup (" "),"c. Write port A",cr,lf
          db      10 dup (" "),"d. Write port B",cr,lf
          db      10 dup (" "),"e. View/Reconfigure DDR-A"
          db      cr,lf
          db      10 dup (" "),"f. View/Recongigure DDR-B"
          db      cr,lf
          db      10 dup (" "),"g. Exit port menu",cr,lf,"$"

step_menu db      10 dup (" "),"SINGLE-STEP MENU",cr,lf,cr,lf
          db      10 dup (" "),"a. Start single-step mode"
          db      cr,lf
          db      10 dup (" "),"b. Step target",cr,lf
          db      10 dup (" "),"c. Exit single-step mode",cr,lf
          db      10 dup (" "),"d. Main menu",cr,lf,lf,"$"

Brk_stat_off      db      "Break   = Disable",cr,lf,"$"
Brk_stat_on       db      "Break   = Enable",cr,lf,"$"
step_stat_off     db      "Stepper = off",cr,lf,lf,lf,"$"
step_stat_on      db      "Stepper = Freerun",cr,lf,lf,lf,"$"
emul_stat_on      db      "Mode    = System",cr,lf,"$"
emul_stat_off     db      "Mode    = Target",cr,lf,"$"

continue  db      10,13,"Press Enter to continue: ","$"
waiting   db      10,13,"Standing by for Emulator data","$"
port_done db      "Port data downloaded",cr,lf,"$"


dseg      ends


;*****************************************************************
sseg      segment stack

          db      100 dup (?)

sseg      ends


;*****************************************************************
cseg      segment

          assume cs:cseg,ds:dseg,ss:sseg,es:cseg

;*****************************************************************
;
; Macro COMPARE - This macro compares the contents of two
```

```
; strings. The soucre and destination offset must be provide
; and the CX register must contain the string length. For a
; match the AX register will return zero. Any other value
; means no match.
;
;*************************************************************

compare   macro   string1,string2

          lea     si,string1      ;point to input string
          lea     di,string2      ;point to test string
          call    cmp_strg
          endm


;*************************************************************
;
; Macro BUF_SIZE - This macro setups function call 08h by
; setting the maximum length and passing the buffer name.
;
;*************************************************************


buf_size  macro length,identity   ;variable lentgh buffer

          push    ax
          push    dx
          lea     dx,identity
          mov     identity,length  ;the max length
          mov     ah,0ah           ;is 255 characters
          int     21h
          pop     dx
          pop     ax

          endm

;*************************************************************
;
; Subroutine CMP_STRG - This procedure compares two strings.
;
;*************************************************************

cmp_strg  proc    near

          push    cx

          mov     cx,04
chk_it:   mov     ah,[si]
          mov     al,[di]
          cmp     ah,al
          jne     no_match
          inc     si
          inc     di
```

```
          loop    chk_it
no_match: mov     ax,cx           ;compare strings
                                  ;set al to the value of
                                  ;cx. If cx equal zero
                                  ;strings match
          pop     cx              ;restore original string
                                  ;length
          ret
cmp_strg endp

;*****************************************************************
;
; Subroutine CLR - This procedure clears the crt display.
;
;*****************************************************************

clr       proc

          push    dx
          push    ax
          mov     dx,offset cls
          mov     ah,09h
          int     21h
          pop     ax
          pop     dx
          ret
clr       endp


;*****************************************************************
;
; Subroutine DISPLAY - This procedure displays the message
; string pointed to by the dummy variable "string".
;
;*****************************************************************

display   macro   string

          push    dx
          mov     dx,offset string
          mov     ah,09h
          int     21h
          pop     dx
          endm


;*****************************************************************
;
; Subrountine READ-KEY - This procedure reads the Z-100
; keyboard and echoes it to the crt.
;
;*****************************************************************
```

```
read_key  proc

          mov     ah,01h
          int     21h
          ret
read_key  endp


;*****************************************************************
;
; Subroutine RCV_B - This procedure reads the auxillary
; serial port and returns the data in the AL register.
;
;*****************************************************************

rcv_B     proc

          mov     ah,03h          ;preform serial read
          int     21h             ;data returned in al
          ret
rcv_B     endp


;*****************************************************************
;
; Subroutine SEND_B - This procedure outputs the data in the
; AL register to the auxillary serial port.
;
;*****************************************************************

send_b    proc

          push    cx
          push    ax

          mov     dl,al
          mov     ah,04h          ;transmit serial byte
          int     21h             ;data must be in the dl
          mov     al,dl           ;register

          pop     ax
          pop     cx
          ret
send_b    endp


;*****************************************************************
;
; Subroutine DISP_STAT - This procedure checks the current
; status of the emulators break, single-step, and mode flags.
; It then displays the appropriate status message for each.
;
;*****************************************************************
```

```
disp_stat proc    near

          push    ax

          mov     al,on             ;load the on flag
          cmp     al,mode_flg       ;check mode of emulator
          jne     skip_z1
          display emul_stat_on
          jmp     skip_z21
skip_z1:  display emul_stat_off
skip_z21: mov     al,on
          cmp     al,break_flg
          jne     skip_z2
          display brk_stat_on
          jmp     skip_z22
skip_z2:  display brk_stat_off
skip_z22: mov     al,on
          cmp     al,step_flg
          jne     skip_z3
          display step_stat_on
          jmp     skip_z31
skip_z3:  display step_stat_off

skip_z31: pop     ax

          ret
disp_stat endp


;*****************************************************************
;
; Macro CONVERT - This macro sets up the parameters
; to the hex_to_ascii procedure which converts an hex value
; to ascii.
;
;*****************************************************************

convert   macro   bits,destination,data_info

          mov     bx,data_info    ;load the data pointer
          mov     di,offset destination
                                  ;load the destination
                                  ·pointer
          mov     cx,bits
          call    hex_to_ascii    ;call the converter

          endm


;*****************************************************************
;
;  Subroutine GET_FLG - This procedure reads in a string
```

```
;  from the emulator and stores it in the message buffer.
;
;*************************************************************


get_flg   proc    near

          push    bx

          mov     bx,offset msg_buf
get_next_1:
          call    rcv_b             ;read emulator port
          mov     [bx],al
          cmp     al,eot            ;if not end of string
                                    ;store the data
          je      done1             ;and get the next byte

          mov     dl,al             ;look at data
          mov     ah,02h
          int     21h

          inc     bx
          jmp     get_next_1

done1:    pop     bx

          ret
get_flg   endp


;*************************************************************
;
;  This macro transmits the designated flag to the emuulator.
;
;*************************************************************

send      macro   flag_name

          lea     di,flag_name
          call    send_flag

          endm


;*************************************************************
;
; Subroutine SEND_FLAG - This procedure sends a string of
; characters from thethe Z-100 to the serial port the string
; offset must be in the DI register when the call is made.
;
;*************************************************************
```

```
send_flag proc    near

          push    ax

continue1: mov    al,[di]          ;load a byte of data
          call    send_b           ;transmit the data
          cmp     dl,eot           ;check for end of string
          je      return
          inc     di               ;point to next byte
          jmp     continue1        ;repeat process
return:   pop     ax

          ret
send_flag endp


;*****************************************************************
;
; MODULE NAME:    LOAD_BUF
; MODULE NUMBER:
; FUNCTION:       This procedure transfer a designated
;                 file from disk to a temporary storage area.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************


load_buf  proc    near

          push    ax
          push    cx

          lea     dx,user_file     ;point to user file
          mov     ax,3d00h         ;open the file for reading
          int     21h              ;pointer at beginning
          jc      load_error
          mov     bx,ax            ;store file handle in BX
          lea     dx,user          ;destination buffer
          mov     cx,2048          ;load number of bytes
          mov     ah,3fh
          int     21h

          mov     ah,3eh           ;close the file
          int     21h
          jmp     no_ld_err
load_error:
          jmp     no_way
no_ld_err:
          pop     cx
          pop     ax
          ret
```

```
load_buf  endp


;*****************************************************************
;
; MODULE NAME:     SEND_BUF
; MODULE NUMBER:
; FUNCTION:        This procedure transfer a file from the
;                  designated buffer to the emulator.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************


send_buf  proc    near


          mov     si,offset user
          add     si,u_start        ;point to start of data
          send    user_stg          ;call emulator
send1:    call    get_flg           ;get response
          compare msg_buf,rdy_stg
          cmp     ax,0
          jne     send1
          send    down_ld           ;send download flag
          mov     bx,0              ;point to start address
          mov     al,bh             ;send first byte of
          call    send_b            ;the start address
          mov     al,bl             ;send second byte
          call    send_b
          mov     bx,byte_num       ;send byte count
          mov     al,bh
          call    send_b
          mov     al,bl
          call    send_b
send2:    call    get_flg
          compare msg_buf,rdy_stg ;get ready signal
          cmp     ax,0
          jne     send2
          mov     chk_sum,0         ;clear checksum
          lea     si,user           ;point to code
          mov     bx,0              ;set counter
send_next:
          mov     al,[si]           ;load data
          call    send_b            ;send data
          add     chk_sum,al        ;update checksum
          inc     bx
          cmp     bx,byte_num
          je      x_send_buf
          inc     si                ;point to the next byte
          jmp     send_next
```

```
x_send_buf:
          mov     al,chk_sum
          call    send_b          ;send checksum

          ret

send_buf  endp

;*****************************************************************
;
; MODULE NAME:    INPUT_DATA
; MODULE NUMBER:
; FUNCTION:       This procedure gets an ascii data byte
;                 from the Z-100 keyboard, converts it to
;                 hex and stores it in the location pointed
;                 to by SI.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************

input_data proc   near

          push    di
          push    bx
          push    cx
          push    dx

again2:   buf_size 3,temp_buf     ;accept 2 characters + cr
          mov     al,bytes_in
          cmp     al,2
          je      passed_11
          display   enter_2       ;display an error message
          jmp     again1
passed_11:
          xor     dx,dx
          mov     cl,4
          lea     di,byte_1
          lea     bx,xlat_ascii_char_2_hex_value
          mov     al,[di]         ;get the first byte
          xlat                    ;exchange a hex character
                                  ;with its binary value
          cmp     al,0ffh
          jne     passed_12
          display enter_2         ;enter 2 hex characters
          jmp     again2
passed_12:
          or      dl,al
          shl     dx,cl           ;shift the byte by one
                                  ;nibble
          mov     al,[di+1]       ;get second byte
          xlat
```

```
            cmp     al,0ffh
            jne     passed_13
            display enter_2
            jmp     again2
passed_13:
            or      dl,al           ;shift the byte by one
                                    ;nibble

            mov     [si],dl         ;store the hex data

            pop     dx
            pop     cx
            pop     bx
            pop     di

            ret
input_data endp


;*************************************************************
;
; MODULE NAME:    HEX_TO_ASCII
; MODULE NUMBER:
; FUNCTION:       This procedure converts the data in the
;                 BX, register from hexidecimal to ascii
;                 and stores  it in the location pointed
;                 to by the DI register.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*************************************************************

hex_to_ascii proc near

            push    cx
            push    bx
            push    ax

            mov     ax,bx           ;load the data in ax
            lea     bx,ascii_table
            push    ax              ;save the data
            cmp     cx,2            ;test for size of the data
            je      byte_only
            mov     al,ah
            and     al,0fh          ;convert low nibble of
                                    ;high byte
            xlat
            mov     [di],al         ;save fisrt character
            inc     di
byte_only:
            pop     ax              ;retrieve second byte
            mov     ah,al           ;hold low byte
```

```
            and     al,0f0h         ;select high nibble
            xlat
            mov     [di],al         ;save the character
            inc     di
            mov     al,ah           ;store low byte
            and     al,0fh          ;select low nibble
            xlat
            mov     [di],al         ;save the character

            pop     ax
            pop     bx
            pop     cx

            ret

hex_to_ascii endp


;*****************************************************************
;
; MODULE NAME:      GET_ADDR
; MODULE NUMBER:
; FUNCTION:         This procedure retrieves a three byte hex
;                   address from the key board and converts it
;                   to its binary equivalent. The results are
;                   returned in the addr_hold buffer.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************

get_addr    proc    near

            push    di
            push    bx
            push    cx
            push    dx

again1:     buf_size 4,temp_buf     ;accept 3 characters + cr
            mov     al,bytes_in
            cmp     al,3
            je      passed_1
            display   enter_3       ;display an error message
            jmp     again1
passed_1:
            xor     dx,dx
            mov     cl,4
            lea     di,byte_1
            lea     bx,xlat_ascii_char_2_hex_value
            mov     al,[di]         ;get the first byte
            xlat                    ;exchange a hex character
```

```
                                        ;with its binary value
            cmp     al,0ffh
            jne     passed_2
            display enter_3             ;enter 3 hex characters
            jmp     again1
passed_2:
            or      dl,al
            shl     dx,cl               ;shift the byte by one
                                        ;nibble
            mov     al,[di+1]           ;get second byte
            xlat
            cmp     al,0ffh
            jne     passed_3
            display enter_3
            jmp     again1
passed_3:
            or      dl,al
            shl     dx,cl               ;shift the byte by one
                                        ;nibble
            mov     al,[di+2]           ;get third byte
            xlat
            cmp     al,0ffh
            jne     passed_4
            display enter_3
passed_4:   or      dl,al
            mov     addr_hold,dx        ;save the converted
                                        ;address
            pop     dx
            pop     cx
            pop     bx
            pop     di

            ret
get_addr    endp


;*****************************************************************
;
; MODULE NAME:      USER_PRO
; MODULE NUMBER:
; FUNCTION:         This procedure handles the user requests
;                   from the Z-100 keyboard. And carries out
;                   the required functions with the emulator.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************

user_pro    proc    near

user_rep:   call    clr
            display user_menu           ;prompt the user for
```

```
            display select          ;a selection
            call    read_key        ;get the option number
            cmp     al,'a'
            je      over_t
            jmp     opt_user_b
            call    clr
over_t:     display file            ;request filename
            buf_size 15,user_code   ;get the string
            mov     bx,offset user_file ;may include drive id
            add     bl,char_ct
            mov     al,0
            mov     [bx],al         ;add zero for dos call
            call    load_buf        ;move file from disk to
                                    ;the buffer area
            mov     bx,offset user
            add     bx,2049
            mov     al,eot
            mov     [bx],al         ;flag end of user file
            mov     xmt,0           ;clear counter
            mov     u_start,0
            mov     byte_num,2048
used:       call    send_buf        ;download the file
            call    get_flg         ;see if transfer was
                                    ;successful
            compare msg_buf,ack_stg
            cmp     ax,0
            jne     retry
            display data_out        ;advise user of the
                                    ;download success
            display continue
not_yet:    call    read_key
            cmp     al,cr
            jne     not_yet
            jmp     exit_user_pro
retry:      add     xmt,1           ;increment counter
            cmp     xmt,3
            jne     skip_a
            jmp     err_user_pro
skip_a:     jmp     used
opt_user_b:
            cmp     al,'b'
            je      skip_b
            jmp     opt_user_c
skip_b:     call    clr
            display start_addr      ;prompt for view address
            call    get_addr        ;get the address
            mov     ax,addr_hold
            mov     u_start,ax      ;save the address
            mov     si,offset user
            add     si,ax
            call    clr
            display view_nr         ;get last address to view
            call    get_addr
```

```
            mov     ax,addr_hold
            mov     u_stop,ax       ;save the converted
                                    ;stop address
            mov     ax,u_start
            mov     bx,u_stop
            sub     bx,ax           ;calculate byte count
            mov     byte_num,bx     ;store byte count
            mov     cx,bx           ;load byte counter
            cmp     cx,20           ;see if bytes > 20
            jle     skip_c
            jmp     no_no           ;if so display error msg
skip_c:     call    clr
            display view_head
see_it:     push    cx
            convert 3,view_addr,u_start
                                    ;convert address to
                                    ;display format
            convert 2,view_data,[si]
                                    ;get desired byte
                                    ;convert to display format
            display view_buf        ;display the location
                                    ;address and contents
            inc     si              ;increment data pointer
            add     u_start,1       ;increment address pointer
            pop     cx
            loop    see_it
            display continue        ;wait for continue
say_1:      call    read_key
            cmp     al,cr
            jne     say_1
            jmp     exit_user_pro

opt_user_c:
            cmp     al,'c'
            jne     opt_user_d
            call    clr

            display start_addr      ;prompt for edit address
            call    get_addr        ;get the address
            mov     ax,addr_hold
            mov     u_start,ax      ;save the address
            mov     u_start1,ax
            mov     si,offset user
            add     si,ax
            call    clr
            display edit_nr         ;get last address to edit
            call    get_addr
            mov     ax,addr_hold
            mov     u_stop,ax       ;save the converted
                                    ;stop address
            mov     ax,u_start
            mov     bx,u_stop
            sub     bx,ax           ;calculate byte count
```

```
            mov     byte_num,bx     ;store byte count
            mov     cx,bx           ;load byte counter
            cmp     cx,10           ;see if bytes > 10
            jg      no_no           ;if so display error msg
            call    clr

edit_in:    push    cx
            convert 3,location,u_start ;setup address display
            call    clr
            display edit_data       ;prompt for the data
            call    input_data      ;read the data and store
                                    ;convert data from ascii
            inc     si              ;to hexidecimal format
            pop     cx
            add     u_start,1
            loop    edit_in
            mov     xmt,0           ;send user file
            mov     ax,u_start1     ;reset start address
            mov     u_start,ax
            jmp     used            ;download new data
opt_user_d:
            cmp     al,'d'
            je      exit_user_pro

no_no:      display bad_sel         ;display invalid input
            call    read_key
            cmp     al,cr
            jne     no_no
            jmp     user_rep

err_user_pro:
  no_way:   call    clr
            display dw_err
            display continue
            call    read_key
            cmp     al,cr
            jne     no_way

exit_user_pro:  ret
user_pro    endp



;*************************************************************
;
; MODULE NAME:      BREAK_PRO
; MODULE NUMBER:
; FUNCTION:         This procedure serves as the interface
;                   between the user and break options. The
;                   user may enable, disable, view or change
;                   the break detectors.
; INPUTS:
; OUTPUTS:
```

```
; MODULES CALLED:
;
;*****************************************************************

break_pro proc    near

          push    di
          push    dx
          push    cx

brk_top:  call    clr
          call    disp_stat
          display break_menu
          display select
          call    read_key          ;get the option number
          cmp     al,'a'
          jne     opt_brk_b
          cmp     break_flg,on      ;see if already enabled
          jne     skip_d
          jmp     no_go
skip_d:   send    break_stg         ;send string to emulator
          call    get_flg
          compare msg_buf,rdy_stg
          cmp     ax,0
          je      skip_e
          jmp     brk_error
skip_e:   mov     al,on             ;send on flag
          call    send_b
          mov     break_flg,al      ;update the break flag
          call    get_flg
          compare msg_buf,ack_stg ;check for acknowledge
          cmp     ax,0
          je      skip_f
          jmp     brk_error
skip_f:   jmp     brk_top
opt_brk_b:
          cmp     al,'b'
          je      skip_g
          jmp     opt_brk_c
skip_g:   cmp     break_flg,off     ;see if already off
          jne     skip_h
          jmp     no_go
skip_h:   send    break_stg         ;send string to emulator
          call    get_flg
          compare msg_buf,rdy_stg
          cmp     ax,0
          je      skip_i
          jmp     brk_error
skip_i:   mov     al,off            ;send on flag
          call    send_b
          mov     break_flg,al      ;update the break flag
          call    get_flg
          compare msg_buf,ack_stg ;check for acknowledge
```

```
            cmp     ax,0
            je      skip_j
            jmp     brk_error
skip_j:     jmp     brk_top
opt_brk_c:
            cmp     al,'c'
            jne     opt_brk_d
            call    clr
            display brk_head          ;display break header

            convert 3,brk_addr1,break_1  ;load the break
            convert 3,brk_addr2,break_2  ;display with
            convert 3,brk_addr3,break_3  ;current data
            convert 3,brk_addr4,break_4
            convert 3,brk_addr5,break_5
            display view_brk
            display continue
no_way_b:   call    read_key
            cmp     al,cr
            jne     no_way_b
            jmp     brk_top
opt_brk_d:
            cmp     al,'d'
            je      back_1
            jmp     opt_brk_e
back_1:     call    clr
            display sel_brk_reg       ;get break register
            call    read_key
            mov     brk_sel,al        ;store ascii register #
            cmp     al,'1'            ;determine break address
            jne     chk_2             ;to display
            convert 3,brk_sel_data,break_1
            mov     di,offset break_1
            jmp     put_out
chk_2:      cmp     al,'2'
            jne     chk_3
            convert 3,brk_sel_data,break_2
            mov     di,offset break_2
            jmp     put_out
chk_3:      cmp     al,'3'
            jne     chk_4
            convert 3,brk_sel_data,break_3
            mov     di,offset break_3
            jmp     put_out
chk_4:      cmp     al,'4'
            jne     chk_5
            convert 3,brk_sel_data,break_4
            mov     di,offset break_4       ;set storage addr
            jmp     put_out
chk_5:      cmp     al,'5'
            je      skip_k
            jmp     no_go
skip_k:     convert 3,brk_sel_data,break_5
```

```
            mov     di,offset break_5
put_out:    display brk_now          ;display current break
                                     ;address
            display new_brk_addr     ;prompt for new address
            call    get_addr         ;get the address
            mov     ax,addr_hold
            mov     [di],ax          ;store the new data
            mov     xmt,0            ;clear retransmit counter

            display more_brk
            call    read_key
            cmp     al,'y'
            je      skip_m
skip_l:     cmp     al,'Y'
            je      skip_m
            jmp     breaking
skip_m:     jmp     back_1           ;change another break addr

breaking:   send    break_flg        ;pass break address
            call    get_flg
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     brk_error
            mov     al,down
            call    send_b
retry_brk:  call    get_flg
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     retry_brk
            mov     cx,10
            mov     chk_sum,0
            lea     di,break_1       ;point to top of break
                                     ;string
brk_out:    mov     al,[di]          ;send the break buffer
            call    send_b
            inc     di
            add     chk_sum,al
            loop    brk_out
            mov     al,chk_sum
            call    send_b
            call    get_flg
            compare msg_buf,ack_stg ;check for acknowledge
            cmp     ax,0
            jne     brk_retry
brk_retry:
            jmp     brk_top
            add     xmt,1
            cmp     xmt,3
            je      brk_error
            jmp     retry_brk
opt_brk_e:
            cmp     al,'e'
            je      exit_brk_pro
```

```
no_go:    call    clr
          display bad_sel          ;display invalid input
          call    read_key
          cmp     al,cr
          jne     no_go
          jmp     brk_top

brk_error:
          call    clr
          display dw_brk_err       ;display error message
no_go1:   display continue
          call    read_key
          cmp     al,cr
          jne     no_go1
          jmp     brk_top          ;try again

exit_brk_pro:     ret
break_pro         endp



;*****************************************************************
;
; MODULE NAME:    STEP_PRO
; MODULE NUMBER:
; FUNCTION:       This procedure signals the emulator to
;                 enables or disables the single-step
;                 function and updates the single-step
;                 status flag. It also steps the target
;                 upon demand.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************

step_pro  proc near

step_top: call    clr
          call    disp_stat
          display step_menu
          display select
          call    read_key         ;get the option number
          cmp     al,'a'
          jne     opt_step_b
          cmp     step_flg,on      ;see if already on
          je      step_top
          send    step_stg
          call    get_flg
wait_step:
          compare msg_buf,rdy_stg ;wait for ready signal
          cmp     ax,0
          jne     wait_step
```

```
            mov     al,on           ;load on flag
            call    send_b
            call    get_flg
            compare msg_buf,ack_stg ;check for acknowledge
            cmp     ax,0
            je      skip_ac
            jmp     step_err
skip_ac:    mov     step_flg,on     ;update the step flag
            jmp     step_top
opt_step_b:
            cmp     al,'b'
            jne     opt_step_c
            mov     bl,on
            cmp     step_flg,bl     ;see if set to on
            je      skip_n
            jmp     can_not_do
skip_n:     send    step_stg        ;contact emulator
wait_step3:
            call    get_flg
            compare msg_buf,rdy_stg ;wait for ready signal
            cmp     ax,0
            jne     wait_step3
            mov     al,step
            call    send_b          ;send step flag
            call    get_flg
            compare msg_buf,ack_stg ;check for acknowledge
            cmp     ax,0
            je      skip_o
            jmp     step_err
skip_o:     display stepped          ;target was stepped
            display continue
wait_step5:
            call    read_key
            cmp     al,cr
            jne     wait_step5
            jmp     step_top

opt_step_c:
            cmp     al,'c'
            je      skip_p
            jmp     opt_step_d
skip_p:     cmp     step_flg,on     ;see if already on
            je      hump
            jmp     step_top
hump:       send    step_stg
            call    get_flg
wait_step1:
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     wait_step1
            mov     al,off          ;load off flag
            call    send_b
            call    get_flg
```

```
          compare msg_buf,ack_stg
          cmp     ax,0
          jne     step_err
          mov     step_flg,off      ;update the step flag
          jmp     step_top
can_not_do:
          display bad_step          ;stepper must be enabled
          display continue          ;try again
wait_step2:
          call    read_key
          cmp     al,cr
          jne     wait_step2
          jmp     step_top
step_err:
          display step_error        ;circuit has a problem
          display continue          ;try again
wait_step21:
          call    read_key
          cmp     al,cr
          jne     wait_step21
          jmp     step_top
opt_step_d:

          ret
step_pro  endp


;*****************************************************************
;
; MODULE NAME:     PORT_PRO
; MODULE NUMBER:
; FUNCTION:        This procedure prompts the user for an
;                  selection and then carries out the command.
;                  The user may read, write, or view the
;                  target's 8755 ports.
; INPUTS:
; OUTPUTS:
; MODULES CALLED:
;
;*****************************************************************

port_pro  proc    near

port_top: call    clr
          call    disp_stat
          display port_menu
          display select
          call    read_key          ;get the option number
          cmp     al,'a'
          je      rd_a

opt_port_b:
```

```
            cmp     al,'b'
            je      skip_q
            jmp     opt_port_c
skip_q:     mov     cl,'b'          ;flag port B
            jmp     over_rd_a
rd_a:       mov     cl,'a'          ;flag port A
over_rd_a:
            mov     xmt,0
            send    port_stg        ;call emulator board
p_wait:     call    get_flg
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     p_wait
            mov     al,up           ;send up flag
            call    send_b
            cmp     cl,'a'
            je      do_rd_a
            mov     al,portB        ;set up port B read
            jmp     do_rd_b
do_rd_a:    mov     al,portA        ;send port identification
do_rd_b:    call    send_b
p_wait1:    call    get_flg
            compare msg_buf,rdy_stg ;wait for ready signal
            cmp     ax,0
            jne     p_wait1
            call    rcv_B           ;get the data
            mov     chk_sum,al      ;save the data
            call    rcv_B
            cmp     al,eot
            je      skip_r
            jmp     exit_port_pro
skip_r:     call    rcv_B           ;get checksum
            cmp     al,chk_sum
            je      skip_s
            add     xmt,1
            cmp     xmt,3
            jne     p_wait1
            jmp     exit_port_pro   ;if no match try again
skip_s:     send    ack_stg
            mov     ax,0
            mov     al,chk_sum
            mov     work,ax
            convert 2,port_view,work
            display port_cont       ;display the port data
            display continue
p_hold:     call    read_key
            cmp     al,cr
            jne     p_hold
            jmp     port_top

opt_port_c:
            mov     xmt,0
            cmp     al,'c'
```

```
            je      wr_a
opt_port_d:
            cmp     al,'d'
            je      skip_t
            jmp     opt_port_e
skip_t:     mov     cl,'b'          ;flag port B
            jmp     over_a
wr_a:       mov     cl,'a'          ;flag port A
over_a:     display port_data        ;prompt for new data
            lea     si,port_hold    ;point to data buffer
            call    input_data      ;get the data
            send    port_stg        ;call emulator board
p_wait4:    call    get_flg
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     p_wait4
            mov     al,down         ;send down flag
            call    send_b
            mov     chk_sum,0       ;clear checksum
p_wait5:    call    get_flg
            compare msg_buf,rdy_stg ;wait for ready signal
            cmp     ax,0
            jne     p_wait5
            cmp     cl,'a'
            je      do_a
            mov     al,portB        ;set up port B write
            mov     dl,port_hold    ;store new data in the
            mov     portb_sto,dl    ;port holding buffer B
            jmp     do_b
do_a:       mov     al,portA        ;send port identification
            mov     dl,port_hold    ;store new data in the
            mov     porta_sto,dl    ;port holding buffer A
do_b:       call    send_b
            mov     chk_sum,al
            mov     al,port_hold    ;send the data
            call    send_b
            add     chk_sum,al
            mov     al,eot
            call    send_b          ;send eot
            mov     al,chk_sum
            call    send_b          ;send checksum
            call    get_flg
            compare msg_buf,ack_stg
            cmp     ax,0            ;check for acknowledge
            je      skip_u
            add     xmt,1
            cmp     xmt,3
            jne     p_wait5
            jmp     exit_port_pro
skip_u:     display port_done
            display continue
p_hold2:    call    read_key
            cmp     al,cr
```

```
            jne     p_hold2
            jmp     port_top

opt_port_e:
            cmp     al,'e'
            je      skip_ad
            jmp     opt_port_f
skip_ad:    mov     xmt,0
            mov     ax,0
            mov     al,ddra_sto
            mov     work,ax
            convert 2,port_view,work  ;convert ddr data
            display port_cont         ;display the port data
            display chg_ddr           ;do you want to change
            call    read_key          ;the ddr
            cmp     al,'y'
            je      skip_v
            jmp     port_top          ;display menu
skip_v:     display port_data          ;prompt for new data
            lea     si,ddra_sto       ;point to data buffer
            call    input_data        ;get the data
            mov     chk_sum,0
            send    port_stg          ;call emulator
p_wait6:    call    get_flg
            compare msg_buf,rdy_stg
            cmp     ax,0
            jne     p_wait6
            mov     al,down           ;send down flag
            call    send_b
p_wait7:    call    get_flg
            compare msg_buf,rdy_stg ;wait for ready signal
            cmp     ax,0
            jne     p_wait7
            mov     al,ddra           ;send port identification
            call    send_b
            mov     chk_sum,al
            mov     al,ddra_sto       ;send the data
            call    send_b
            add     chk_sum,al
            mov     al,eot
            call    send_b            ;send eot
            mov     al,chk_sum
            call    send_b            ;send checksum
            call    get_flg
            compare msg_buf,ack_stg
            cmp     ax,0              ;check for acknowledge
            je      skip_w
            add     xmt,1
            cmp     xmt,3
            jne     p_wait7
            jmp     exit_port_pro
skip_w:     display ddr_out           ;ddr downloaded message
            display continue
```

```
p_hold3:  call    read_key
          cmp     al,cr
          jne     p_hold3
          jmp     port_top

opt_port_f:
          cmp     al,'f'
          je      skip_x
          jmp     opt_port_g

skip_x:   mov     xmt,0
          mov     ax,0
          mov     al,ddrb_sto
          mov     work,ax
          convert 2,port_view,work  ;convert ddr data
          display port_cont         ;display the port data
          display chg_ddr           ;do you want to change
          call    read_key          ;the ddr
          cmp     al,'y'
          je      skip_y
          jmp     port_top          ;display menu
skip_y:   display port_data         ;prompt for new data
          lea     si,ddrb_sto       ;point to data buffer
          call    input_data        ;get the data
          mov     chk_sum,0
          send    port_stg          ;call emulator
p_wait8:  call    get_flg
          compare msg_buf,rdy_stg
          cmp     ax,0
          jne     p_wait8
          mov     al,down           ;send down flag
          call    send_b
p_wait9:  call    get_flg
          compare msg_buf,rdy_stg ;wait for ready signal
          cmp     ax,0
          jne     p_wait9
          mov     al,ddrb           ;send port identification
          call    send_b
          mov     chk_sum,al
          mov     al,ddrb_sto       ;send the data
          call    send_b
          add     chk_sum,al
          mov     al,eot
          call    send_b            ;send eot
          mov     al,chk_sum
          call    send_b            ;send checksum
          call    get_flg
          compare msg_buf,ack_stg
          cmp     ax,0              ;check for acknowledge
          jne     alright
          add     xmt,1
          cmp     xmt,3
          jne     p_wait9
```

```
            jmp     exit_port_pro
alright:    display ddr_out          ;ddr downloaded message
            display continue
p_hold4:    call    read_key
            cmp     al,cr
            jne     p_hold4
            jmp     port_top

exit_port_pro:
            display port_err         ;display error message
p_hold5:    display continue
            call    read_key
            cmp     al,cr
            jne     p_hold5
            jmp     port_top
opt_port_g:
            ret

port_pro    endp


;*****************************************************************

start:
            mov     ax,dseg
            mov     ds,ax            ;set data segment
            mov     mode_flg,on      ;initialize to emulator
            mov     break_flg,off    ;break off
            mov     step_flg,off     ;stepin step mode

            call    clr
            display standby          ;display wait message
            call    get_flg
            compare msg_buf,ack_stg ;look for first ack signal
            cmp     ax,0             ;if 0 then SRAM ok
            je      sram_ok
            display sram_err         ;if ram is bad display
                                     ;error message and goto DOS
            jmp     dos
sram_ok:    call    get_flg
            compare msg_buf,ack_stg ;look for second ack signal
            cmp     al,0             ;if 0 then user RAM ok
            je      well
            display sram_err         ;if ram is bad display
                                     ;error message and goto DOS
            jmp     dos
well:       mov     xmt,0
send_emul:
            send    down_ld          ;send download flag
            mov     chk_sum,0
humpl:      lea     dx,filnam        ;point to emulator code
            mov     ax,3d00h         ;open the file for reading
            int     21h              ;pointer at beginning
```

```
            jnc     skip_ab
            jmp     load_err
skip_ab:    mov     bx,ax               ;store file handle in BX
next_set:   lea     dx,user             ;destination buffer
            mov     cx,2048             ;load number of bytes
            mov     ah,3fh
            int     21h
            mov     cx,ax               ;set counter to actual
                                        ;number of bytes
            mov     si,offset user      ;point to buffer
emul_down:
            cmp     ax,0
            je      emul_gone
emul_d1:    mov     al,[si]
            call    send_b
            inc     si
            add     chk_sum,al          ;update checksum
            loop    emul_d1
            jmp     next_set
emul_gone:
            mov     al,ETB               ;place eot at the end
            call    send_b               ;of the stored data
            mov     al,ETB
            call    send_b
            mov     al,chk_sum          ;send checksum value
            call    send_b

            call    get_flg
            compare msg_buf,ack_stg
            cmp     ax,0
            jne     skip_z
            jmp     window_1            ;display setup menu
                                        ;else retransmit
skip_z:     add     xmt,1
            cmp     xmt,3
            jne     skip_ae
            jmp     dos
skip_ae:    jmp     send_emul

            mov     ah,3eh              ;close the file
            int     21h

window_1:   call    clr
            call    disp_stat           ;display system status
            display setup_menu          ;display setup menu
            display select
            call    read_key            ;get the option number
            cmp     al,'a'
            jne     opt_set_b
            call    user_pro            ;goto user memory
                                        ;subroutine
            jmp     window_1
```

```
opt_set_b:
          cmp     al,'b'
          jne     opt_set_c
          call    break_pro       ;goto break control
                                  ;subroutine
          jmp     window_1

opt_set_c:
          cmp     al,'c'
          jne     opt_set_d
          call    port_pro        ;goto port subroutine
          jmp     window_1

opt_set_d:
          cmp     al,'d'
          jne     opt_set_e
          jmp     window_2        ;goto main menu
opt_set_e:
          display bad_sel         ;display invalid input
opt_set_e1:
          call    read_key
          cmp     al,cr
          jne     opt_set_e1
          jmp     window_1

window_2: call    clr
          call    disp_stat       ;display system status
          display main_menu       ;display main menu
          display select
          call    read_key        ;get the option number
          cmp     al,'a'
          jne     opt_main_b
          call    step_pro        ;goto target subroutine
          jmp     window_2

opt_main_b:
          cmp     al,'b'
          jne     opt_main_c
          call    break_pro       ;toggle the break status
          jmp     window_2

opt_main_c:
          cmp     al,'c'
          jne     opt_main_d
          call    break_pro       ;goto break address
                                  ;subroutine
          jmp     window_2

opt_main_d:
          cmp     al,'d'
          jne     opt_main_e
          call    step_pro        ;goto single-step control
                                  ;subroutine
```

```
                jmp     window_2

opt_main_e:
                cmp     al,'e'
                jne     opt_main_f
                call    user_pro
                jmp     window_2

opt_main_f:
                cmp     al,'f'
                jne     opt_main_g
                call    port_pro        ;goto to port subroutine
                jmp     window_2

opt_main_g:
                cmp     al,'g'
                jne     opt_main_h
dos:            mov     ax,4c00h        ;return to DOS
                int     21h
opt_main_h:
                display bad_sel         ;display invalid input
opt_set_h:
                call    read_key
                cmp     al,cr
                jne     opt_set_h
                jmp     window_2
load_err:       call    clr
                display disk_err
                display continue
load_err1:
                call    read_key
                cmp     al,cr
                jne     load_err1
                jmp     hump1

  cseg          ends
                end     start
```

Appendix E: User Guide

This user guide contains information concerning emulator system testing which is still pending. For completeness the interrelationship of untested circuits and the rest of the system will be discussed.

The only portion of the emulator which has not been tested is the break detector circuitry. The individual components of this section have been tested, but the overall functionality of the circuit is still pending. This circuit consists of the 74F524 register comparators (U19-U28), the 8259A interrupt controller (U58), and the 74F157 multiplexers (U42-U44). These devices are completely wired and respond to their control and data signal in the proper manner.

Testing of the register/comparators consisted of preforming read and write operations to ensure proper wiring. The interrupt controller appears to function properly when excited manually, but hasn't been verified with software control. The break disable function is functional and provides a means of using the emulator in a limited capacity. The other functions of the emulator do not directly depend on the break feature and function properly when isolated from the break section.

Software to support the break circuits consists of an interface to the user via the host computer and a corresponding break module in the emulator code. The host

computer portion of the control software has been tested by performing data aquisition. The emulator portion has some additional testing to under go. The break handler routine has not been tested in response to an actual break condition. This program is currently preventing complete testing of the break detector hardware as a complete circuit. No additional hardware errors are expected from the break circuit, but this is not verifiable at this time.

Bibliography

1. Bisbee, Lt Col Charles, Associate Professor. Personal Interview. Air Force Institute of Technology, Wright-Patterson AFB, OH 21 April 1988.

2. Intel Corporation. Microsystem Handbook Set. Intel Literature Department, Intel Corporation, Santa Clara, CA 1986.

3. Booch, Grady Software Engineering with Ada Tokyo: The Benjamin/Cummings Company, 1987.

4. Dinwiddie, George "An 8031 In-Circuit Emulator", Byte, Small System Journal, 11: 181-199 (July 1986).

5. Pedicini, Chris " Engineering on a Micro", Byte, Small System Journal, 11: 145-170 (July 1987).

6. Pressman, Roger S. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill, 1987.

7. Russo, John P. "A Vic-201 Commodore 64 Terminal Emulator", Byte, Small System Journal, 9: 379-388 (April 1986).

8. Stern, Marc "All About Interfacing", Radio Electronics, 57: 87-96 (December 1986).

9. Wilcox, Alan D. Engineering Design: Project Guidelines. Englewood Cliffs NJ: Prentice-Hall, 1987.

10. Woodhull, Albert S. "An EPROM Simulator", Byte, Small System Journal, 9: 400-410 (March 1985).

VITA

Captain John L Woods was born on [redacted] He graduated from high in Deadwood, South Dakota, in 1970. He entered the USAF in 1972 and served until July 1981. Upon leaving the Air Force he attended the University of Oklahoma, form which he received the degree of the Bachelor of Science in Electrical Engineering in May 1984. Upon graduation, he received a commission in the USAF through the Officer Training School. He received the degree of Master of Science in Consumer Studies at Oklahoma State University, Stillwater Oklahoma. He served as an Elecromagnetic Hazards Engineer at the Engineering Installation Division, Tinker AFB, Oklahoma, until entering the School of Engineering, Air Force Institute of Technology, in June 1987.

[redacted]

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Unlimited Distribution
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-63	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENA	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

8755 EMULATOR DESIGN (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)

John L Woods, Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1988 December	205

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	6		Computers, Computer Logic

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Charles R. Bisbee, Lt Col, USAF
Associate Professor of Electrical Engineering

Approved for release in accordance with AFR 190-1
[signature]
12 Jan 1989

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Lt Col Charles R. Bisbee	513-255-6913	AFIT/ENG

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

Item 19.

Abstract

This paper discusses the requirements to develop and build an electronic device to emulate the 8755 microchip. The design had five basic objectives: (1) Allow the user to download 8755 emulation memory. (2) Allow control of the target program from the Z-100. (3) Provide a single step capability. (4) Provide breaking at a specified address. (5) Allow the user to set or change the emulated 8755 input/output ports.

It describes the standard memory and input/output capabilities of the 8755. It describes in detail the emulator enhancement features to the standard 8755. The hardware circuits used to implement the emulator are discussed at the block diagram, component, and signal levels. It concludes with a detail description of the emulator sotware used to control the hardware.