

Development of a Stack-based Central Processing Unit Using TTL Logic

by

Aaron Tang Shen Lee

Final Report submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronic Engineering)

DECEMBER 2006

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Development of a Stack-based Central Processing Unit Using TTL Logic

by

Aaron Tang Shen Lee

A project dissertation submitted to the
Electrical and Electronic Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL AND ELECTRONIC ENGINEERING)

Approved by,



(Dr. Yap Vooi Voon)

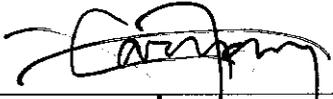
UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

December 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



AARON TANG SHEN LEE

ABSTRACT

The objective of this project is to build a stack-based central processing unit (CPU) using discrete Transistor-Transistor Logic (TTL). FORTH is the software that is to be implemented on the computer system.

The scope of this project is limited to developing and building a stack-based FORTH computer system. Wire wrapping was used to construct the CPU and the computer system. The final computer system consists of 9 wire-wrapped Eurocards connected together through a backplane.

The project is successful. A fully functional stack-based FORTH computer system has been successfully developed.

ACKNOWLEDGEMENTS

This project would never have been possible without the contribution of some very special people. I would like to thank them from the bottom of my heart.

To Mr. Andrew Holme, architect and designer of the Mark 1 FORTH Computer. For supporting me during the duration of the project. And also for his generosity in sharing his experience and knowledge.

To Dr. Yap Vooi Voon, my supervisor. For his constant guidance in every step of this project. And also for his wisdom and constant encouragement, which particularly helped me through all the rough times in this project.

To Mr. Patrick Sebastian, my microprocessor lecturer. For all his advice and support throughout this project. I would never have been able to complete the project without his support.

To Mr. Muhamad Aidil b. Jazmi, my classmate and hardware guru. For his willingness to help me anytime I was stumped. His amazing electronic skills really helped me complete this project on time.

To my friends Chen, Chok, Tong, Sim, Yaw, Sanjeev, Fong and Hikki. For keeping me sane throughout this crazy final year.

To Shirley. For being my inspiration.

TABLE OF CONTENTS

| | | | | | | | | |
|-------------------------|-----------------------------------|--|---|---|---|---|---|-----|
| CERTIFICATION | . | . | . | . | . | . | . | i |
| ABSTRACT | . | . | . | . | . | . | . | iii |
| ACKNOWLEDGEMENTS | . | . | . | . | . | . | . | iv |
| CHAPTER 1: | INTRODUCTION | . | . | . | . | . | . | 1 |
| | 1.1 | Background of Study | . | . | . | . | . | 1 |
| | 1.2 | Problem Statement | . | . | . | . | . | 2 |
| | 1.3 | Objectives and Scope of Study | . | . | . | . | . | 2 |
| | 1.4 | Overview of Report | . | . | . | . | . | 3 |
| CHAPTER 2: | LITERATURE REVIEW | . | . | . | . | . | . | 4 |
| | 2.1 | Register-based Computer Architecture | . | . | . | . | . | 4 |
| | 2.2 | Introduction to Stack Computers and FORTH | . | . | . | . | . | 11 |
| | 2.3 | Advantages of Stack Computing | . | . | . | . | . | 11 |
| | 2.4 | Stack-based Computer Architecture | . | . | . | . | . | 12 |
| | 2.5 | Stack Computing versus Register-based Computing | . | . | . | . | . | 15 |
| CHAPTER 3: | METHODOLOGY / PROJECT WORK | . | . | . | . | . | . | 17 |
| | 3.1 | Procedure Identification | . | . | . | . | . | 17 |
| | 3.2 | Equipment Required | . | . | . | . | . | 18 |
| | 3.3 | The Mark 1 FORTH Computer Design | . | . | . | . | . | 18 |
| | 3.4 | Software | . | . | . | . | . | 21 |
| | 3.5 | Testing and Troubleshooting | . | . | . | . | . | 24 |

| | | | |
|-------------------|--|-----------|----|
| CHAPTER 4: | RESULTS AND DISCUSSION | | 35 |
| | 4.1 Results. | | 35 |
| | 4.2 Changes to the Mark 1 FORTH Computer | | 42 |
| | 4.3 Performance | | 43 |
| CHAPTER 5: | CONCLUSION AND RECOMMENDATIONS | | 45 |
| | 5.1 Recommendations for Future Work | | 45 |
| REFERENCES | | | 47 |
| APPENDICES | | | 48 |

LIST OF FIGURES

Figure 2.1: Block Diagram of a Generic Datapath

Figure 2.2: Simplified Block Diagram of a Generic Datapath

Figure 2.3: Example of a typical 32-bit Instruction Format

Figure 2.4: Architecture of a Simple Computer

Figure 2.5: Architecture for a Generic Stack Computer

Figure 3.1: Mark 1 FORTH Computer Architecture

Figure 3.2: Procedures for Software

Figure 3.3: Schematic for Logical Testing Circuit

LIST OF TABLES

Table 2.1: An Example of Instruction Specifications

Table 2.2: Example of a Simple Operation

Table 2.3: Summary of Differences between Stack and Register Based Computing

Table 3.1: Mark 1 FORTH Computer Specifications

Table 3.2: Description of Mark 1 FORTH Computer Components

Table 3.3: Location of Memory Chips

Table 3.4: Results from ADD Operation on ALU Card

Table 3.5: Test Results from Microcode Sequencer

Table 3.6: ALU Functions

Table 3.7: Memory Card Test Results

Table 4.1: Performance Comparison between Mark 1 and Mark 2 Computers

LIST OF PHOTOGRAPHS

Photograph 3.1: Logical Testing Circuit

Photograph 3.2: Underside of ALU

Photograph 3.3: Underside of System Clock and Instruction Decoder

Photograph 3.4: Underside of Diode ROM

Photograph 3.5: Underside of IP Index Register

Photograph 3.6: Underside of W Index Register

Photograph 3.7: Underside of Input/Output Card

Photograph 3.8: Underside of Memory Card

Photograph 3.9: Underside of Microcode Sequencer

Photograph 3.10: Underside of Stacks Card

Photograph 4.1: Angled View of the Mark 1 FORTH Computer

Photograph 4.2: Top View of the Mark 1 FORTH Computer

Photograph 4.3: Side View of the Mark 1 FORTH Computer

LIST OF SCREENSHOTS

Screenshot 4.1: Initial Screen

Screenshot 4.2: Simple FORTH

Screenshot 4.3: Looping the Alphabet

Screenshot 4.4: Displaying “F” Part 1

Screenshot 4.5: Displaying “F” Part 2

CHAPTER 1

INTRODUCTION

1.1 Background of Study

Stack data structures (based on the Last In First Out (LIFO) principle) have been in use in computers since the 1950s. Originally, they were used to increase the execution efficiency of high-level programming languages. Today, stacks are mostly used as secondary data handling structures. For example, the program counter (PC) in typical CPU designs emulates a stack structure. However, computers that use hardware stacks as their primary data handling mechanism (stack computers) never found widespread usage and acceptance. On the other hand, register-based machines became more and more popular over the years.

A reason for this trend is suggested: In the past, stack computers used stacks that were stored in program memory. This was both slow and expensive. Therefore, much more research and development work went into register-based machines, which were perceived as being superior in architecture. However, recent developments have now made it possible for large, high speed dedicated stack memories to be cost effective. Stack computers now show a good combination of simplicity, speed and flexibility. The time has now come again for the stack computer to be considered as an alternative design to the dominant CISC and RISC designs. [1]

1.2 Problem Statement

Advances in integrated circuit technology have made it possible for many components of computer systems to be put on a single chip. Hence, more compact and sophisticated computer systems are possible. However, many electronics engineers have lost sight of what actually goes on inside a central processing unit (CPU). Many approaches have been used in attempts to resolve this problem. Unfortunately though, the internal workings of a CPU are still not clearly understood by many engineers.

To address this problem, engineers need to acquire the knowledge to be able to design and build their own CPUs. This project provides a hands-on experience in understanding the workings of a CPU and a computer system.

1.3 Objective and Scope of Study

The objective of this project is to build a central processing unit using discrete Transistor-Transistor Logic (TTL). The CPU will use a stack-based architecture. FORTH is the language that will be used to program the CPU. The final product should be a completely working stack-based FORTH computer system.

This project will be limited to developing and building a FORTH-based computer system using the previously mentioned CPU. CPU design will not be attempted in order to simplify matters. An available CPU design will be used to develop and build the system. Given the long time frame to develop the system, it is envisaged that the project is feasible.

1.4 Overview of Report

Chapter 2: Literature Review

- Register-based Computer Architecture
- Introduction to Stack Computers and FORTH
- Advantages of Stack Computing
- Stack-based Computer Architecture
- Stack Computing versus Register-based Computing

Chapter 3: Methodology

- Procedure Identification
- Equipment Required
- The Mark 1 FORTH Computer Design
- Software
- Testing and Troubleshooting

Chapter 4: Results and Discussion

- Changes to the Mark 1 FORTH Computer
- Performance

Chapter 5: Conclusion and Recommendations

- Recommendations for Future Work

CHAPTER 2

LITERATURE REVIEW

2.1 Register-based Computer Architecture

Computer architecture refers to a high-level description of the hardware required to implement a computer. [2] Typically, the architecture for a computer can be divided into a datapath and a control. The datapath will be discussed in the next section. The section following that will discuss the instruction set architecture (a set of instructions that are used to control the computer) for a conventional computer.

2.1.1 Datapaths

The datapath is defined by three basic components:

- a set of registers
- the microoperations that are performed on data stored in the registers
- the control interface

The registers are used to provide temporary, high-speed storage of data. This data will be used during microoperations that performed.

In order to perform a microoperation, the contents of specified source registers are sent to the ALU. The ALU is a shared unit; since a large number of registers have access to it. The ALU performs the operation, and the result of this operation is transferred to a destination register. Since the ALU is used in most operations, it is an integral part of the datapath. An example of another component required to perform a microoperation is a shift register. These shift registers are also found in datapaths.

In addition to the units mentioned on the previous page, the datapath also contains the digital logic that implements various microoperations. This is the control interface. This digital logic generally consists of a variety of buses, multiplexers, decoders and processing circuits. A block diagram of a simple, generic datapath is shown in Figure 2.1 on the next page. This datapath consists of four registers, an ALU, a shifter and the control interface. [2]

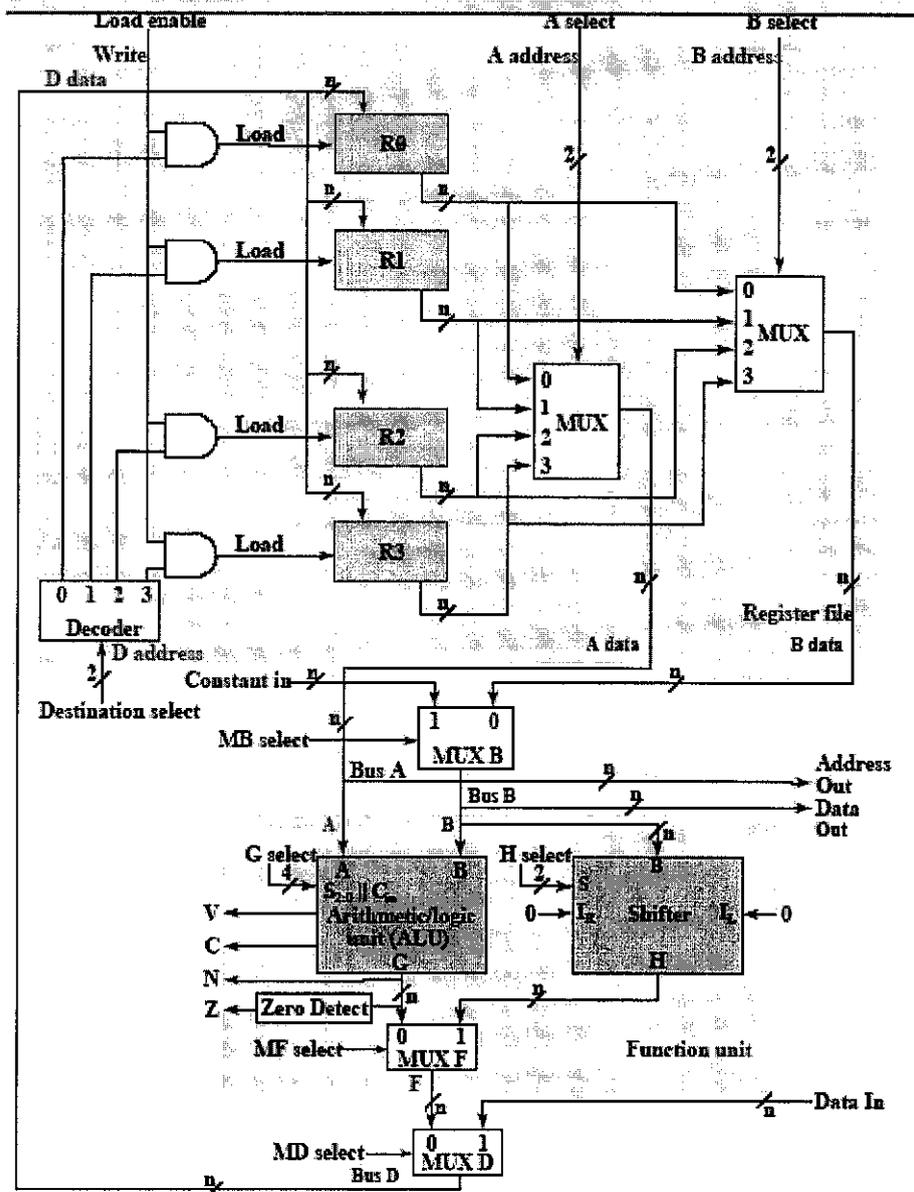


Figure 2.1: Block Diagram of a Generic Datapath

(Source: M. Mano, C. Kime; Logic and Computer Design Fundamentals; pg 432)

A simplified block diagram for a datapath is shown in the Figure 2.2 on the next page. In this datapath, the ALU and shifter are combined as a single function unit. The registers are organized into a single register file. A register file is basically a set of registers which have common microoperations. This gives the block diagram as shown in Figure 2.2 on the next page.

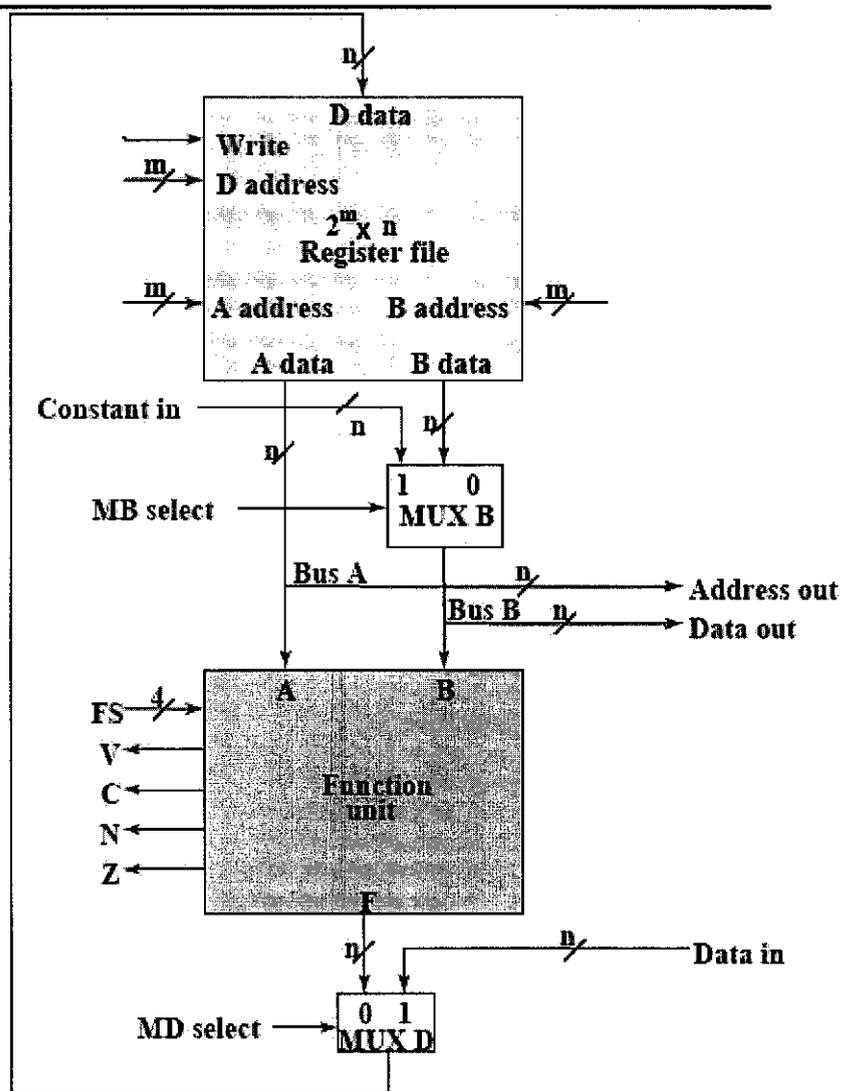


Figure 2.2: Simplified Block Diagram of a Generic Datapath

(Source: M. Mano, C. Kime; Logic and Computer Design Fundamentals; pg 442)

2.1.2 Instruction Set Architecture (ISA)

A computer can only perform a task when it is given an instruction. The instructions that may be issued by a user are stored in the instruction memory of a computer. When an instruction is issued, the control unit reads the issued instruction from memory and decodes and executes the instruction. The way it executes the instruction is by issuing a sequence of one or more of the previously discussed microoperations. Put simply, this means that an instruction consists of a combination of microoperations.

An instruction can be more specifically defined as a collection of bits that instructs the computer to perform a specific operation. The set of instructions that a computer can handle is called its instruction set. The thorough description of this instruction set is the Instruction Set Architecture. For a simple register-based computer, the ISA has three major components: the storage resources, the instruction formats and the instruction specifications.

Put simply, the storage resources component of the ISA tells the programmer (or user) which and how much resources are available for storing information. For example, a programmer might have eight 16-bit registers, a 16-bit program counter, 64 KB of Instruction Memory, and 64 KB of Data Memory.

The instruction format tells the programmer how the bits of the instruction are arranged. The instruction bits are typically divided into groups called fields. Each field is assigned a specific item. For example, one field is for just the operation code, while another field is just for the destination register address. The operation code or opcode is always at the most significant bit part of an instruction. The opcode contains control signals that prepare the datapath (e.g. set the control logic) for a specific operation. Other fields can contain different items, depending on the operation. An example of a 32-bit instruction format is shown on the next page.

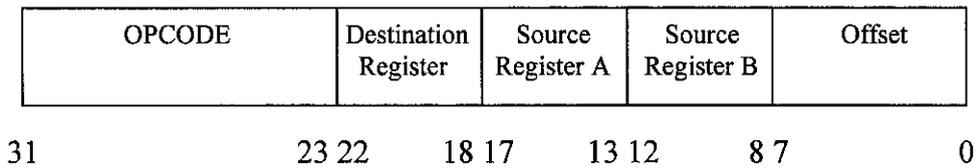


Figure 2.3: Example of a typical 32-bit Instruction Format

The final component of the ISA is the instruction specifications. These specifications basically describe each of the instructions that can be executed by the computer. The instruction specifications are usually given in a table as shown in Table 2.1 below. Therefore, the instruction specifications can be also seen as a detailed list of instructions that can be executed by the computer. [3]

| Instruction | Opcode | Mnemonic | Format | Description | Status Bits |
|--------------|---------|----------|----------|---------------------------------------|-------------|
| Move A | 0000000 | MOVA | RD,RA | $R[DR] \leftarrow R[SA]$ | N, Z |
| Increment | 0000001 | INC | RD,RA | $R[DR] \leftarrow R[SA] + 1$ | N, Z |
| Add | 0000010 | ADD | RD,RA,RB | $R[DR] \leftarrow R[SA] + R[SB]$ | N, Z |
| Subtract | 0000101 | SUB | RD,RA,RB | $R[DR] \leftarrow R[SA] - R[SB]$ | N, Z |
| Decrement | 0000110 | DEC | RD,RA | $R[DR] \leftarrow R[SA] - 1$ | N, Z |
| AND | 0001000 | AND | RD,RA,RB | $R[DR] \leftarrow R[SA] \wedge R[SB]$ | N, Z |
| OR | 0001001 | OR | RD,RA,RB | $R[DR] \leftarrow R[SA] \vee R[SB]$ | N, Z |
| Exclusive OR | 0001010 | XOR | RD,RA,RB | $R[DR] \leftarrow R[SA] \oplus R[SB]$ | N, Z |
| NOT | 0001011 | NOT | RD,RA | $R[DR] \leftarrow \overline{R[SA]}$ | N, Z |

Table 2.1: An Example of Instruction Specifications

(Source: M. Mano, C. Kime; Logic and Computer Design Fundamentals; pg 454)

As previously mentioned, the architecture of a computer consists of datapath and control. The control of the computer is done via the ISA, as explained in the previous section. A generic block diagram combining datapath and control for a simple computer architecture is shown on the next page.

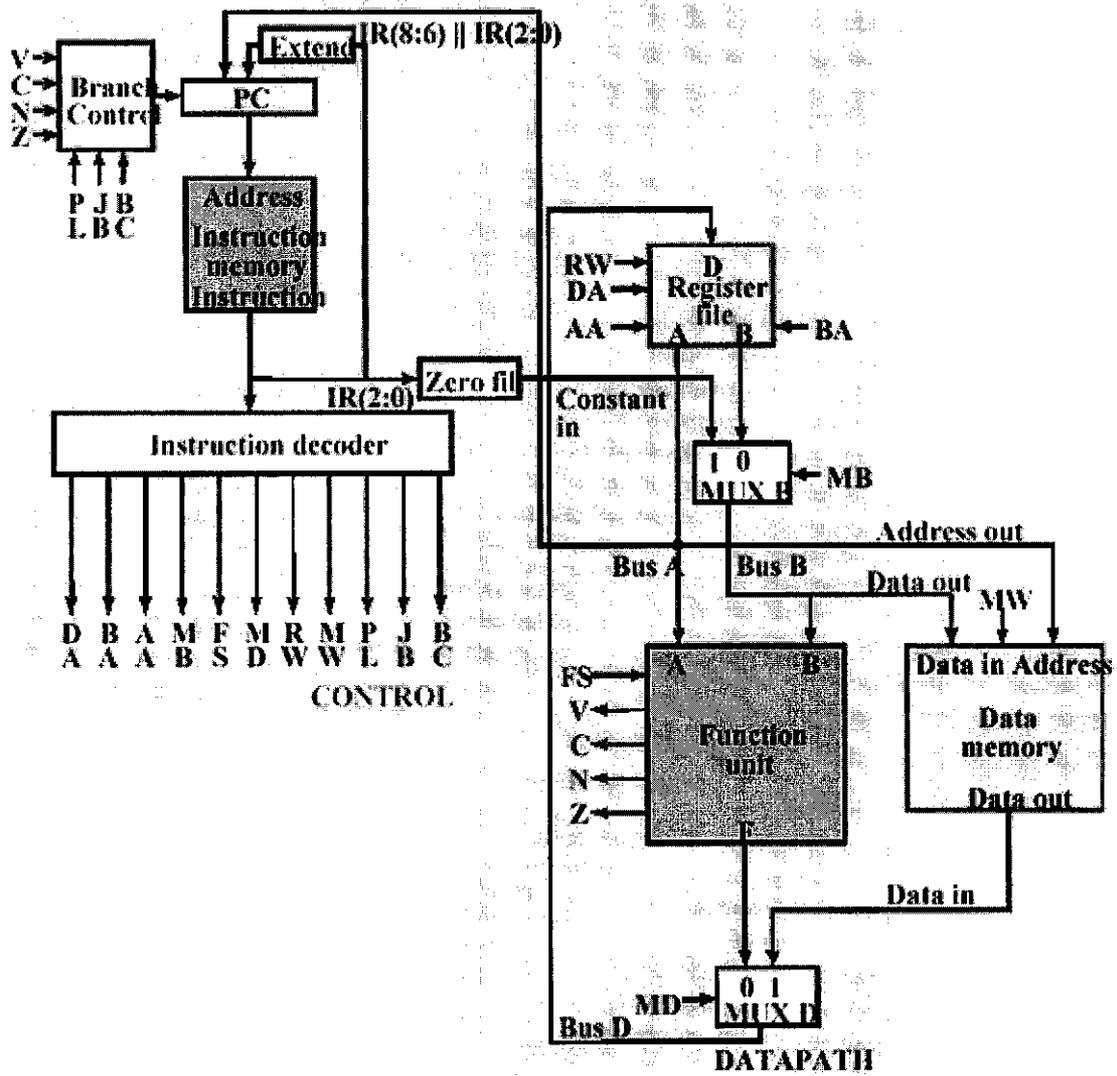


Figure 2.4: Architecture of a Simple Computer

(Source: M. Mano, C. Kime; Logic and Computer Design Fundamentals; pg 457)

2.2 Introduction to Stack Computers and FORTH

A stack computer is a system that is based on the use of stacks, rather than being register based. A stack is defined as a data structure that is based on the LIFO (Last In First Out) principle. Stacks are the simplest way of saving information for common operations, such as evaluating math equations, and calling subroutines. Stack computers are particularly useful in real time control applications.

FORTH is a programming language that is very often used in stack computers. The reason for this is that FORTH itself is based on a set of primitives that execute on a virtual stack machine. Using FORTH, a lot of processing power can be obtained from small hardware.

FORTH is a procedural and reflective programming language. It does not make use of type checking. Reflection refers to the ability a program has to modify or improve itself. FORTH features both interactive execution of commands (making it possible for FORTH to be used as a shell, in the absence of a formal operating system) and the ability to compile sequences of commands for later execution. Early FORTH versions generally compiled threaded code, but many modern versions generate optimized machine code like other language compilers. [4]

2.3 Advantages of Stack Computing

Stack computers have the following advantages:

- Less processor complexity compared to CISC designs
- Less system complexity compared to RISC and CISC designs
- Easier to write programs and compilers
- More reliable programs
- More efficient than register based machines in running well-modularized programs
- Provide more processing power with very little hardware [1]

2.4 Stack-based Computer Architecture

Stack computers and conventional computers have different architectures. A block diagram for the architecture of a generic stack computer is shown below.

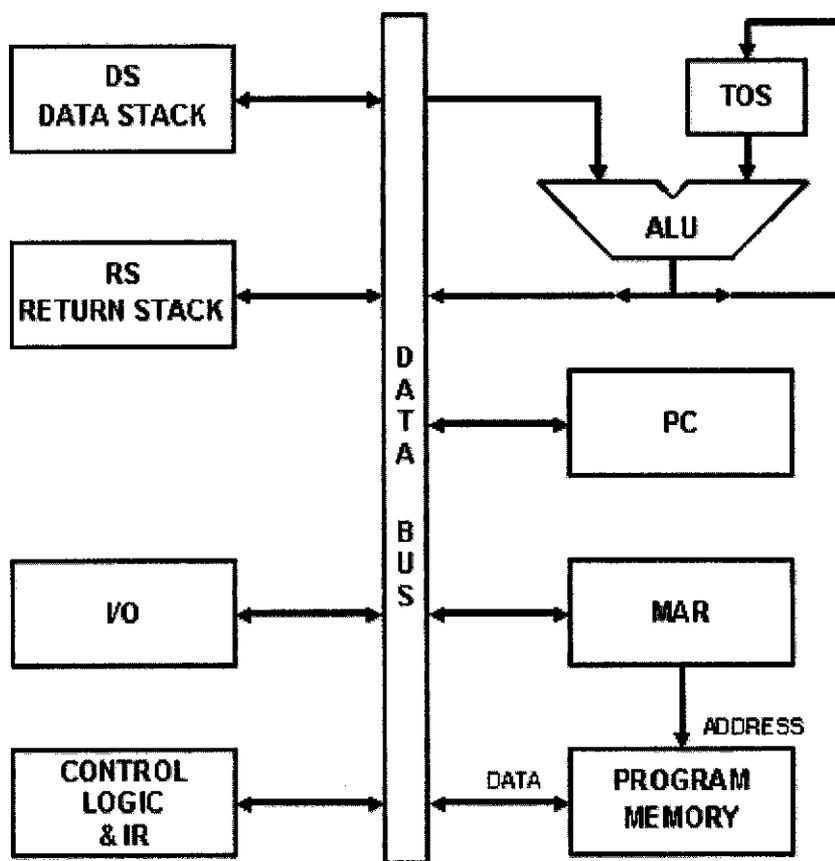


Figure 2.5: Architecture for a Generic Stack Computer

(Source: P. Koopman; Stack Computers: The New Wave, Internet Book:

http://www.cs.cmu.edu/~koopman/stack_computers/; 1989)

The stack computer shown on the previous page is a type of stack computer with multiple stacks, a large stack memory and 0-operand stack addressing. Each box in the diagram represents a logical resource for the machine. These resources are the data bus, the data stack (DS), the return stack (RS), the arithmetic/logic unit (ALU) with its top of stack register (TOS), the program counter (PC), program memory with a memory address register (MAR), control logic with an instruction register (IR), and an input/output component (I/O).

The following sections describe each component in more detail.

2.4.2 Data Bus

The stack computer shown has a single bus connecting the system resources. This is for the sake of simplicity. Commercial stack computers may have more than one data path to allow both instruction fetching and calculations to be done at the same time. In the generic stack computer, the data bus allows a single transmitting functional block and a single receiving functional block during any single operation cycle.

2.4.3 Data Stack

The data stack is memory with an internal mechanism to implement a LIFO stack. A common way to do this might be a conventional RAM with an up/down counter used for address generation. The data stack allows the two original stack operations: push and pop. The push operation extracts the value on the data bus and writes it onto the top of the stack. The pop operation extracts the value on the top of the stack and writes it onto the data bus. It then removes the value at the top of the stack, exposing the second-topmost value on the stack. This value will be used for the next processor operation.

2.4.4 Return Stack

The return stack is similar to the Data Stack in operation. The only difference is that it is used to store subroutine return addresses, instead of instruction operands.

2.4.5 ALU and Top-of-stack Register

The ALU functional block performs arithmetic and logical operations on pairs of data elements. One of these data element pairs is the top-of-stack (TOS) register, which holds the topmost element of the data stack. Thus, to the programmer, the perceived top data stack element is the data item kept in the TOS register buffer (at the ALU). The perceived second topmost element is actually the topmost item on the real data stack. This scheme allows using a single ported data stack memory while allowing operations, such as addition, on the top two stack elements. The ALU supports the standard primitive operations for any computer. This includes addition, subtraction, logical functions (AND, OR, XOR), and test for zero.

2.4.6 Program Counter

The program counter holds the address of the next instruction that is to be executed. The PC may be incremented (e.g. $PC \leftarrow PC + 4$) to fetch the next sequential instruction from program memory, or may be loaded directly from the bus to implement branches.

2.4.7 Program Memory

The program memory block has a Memory Address Register (MAR) and a reasonable amount of RAM. To access the memory, first the MAR is written with the address to be read or written. Then, on the next system cycle, the program memory is either read onto or written from the system data bus.

2.4.8 Input/Output

This component handles communications to/from the world outside the CPU.

[1]

2.5 Stack Computing versus Register-based Computing

The basic difference between stack computing and register based computing is that pure stack machines make use of 0-operand stack addressing, while register machines use register based addressing. For example, suppose a stack computer and a register-based computer need to perform the following operation:

$$X = (A + B) (C + D)$$

| Stack Computer | Register-based Computer |
|----------------|-------------------------|
| PUSH A | LD R1,A |
| PUSH B | LD R2,B |
| ADD | ADD R3,R1,R2 |
| PUSH C | LD R1,C |
| PUSH D | LD R2,D |
| ADD | ADD R1,R1,R2 |
| MUL | MUL R1,R1,R3 |
| POP X | ST X,R1 |

Table 2.2: Example of a Simple Operation

(Source: M. Mano, C. Kime; Logic and Computer Design Fundamentals; pg 488)

The two computers above perform the same task. However, the way each computer approaches the operation is clearly different. The stack computer makes full use of a stack, performing operations on items only at the top of the stack. This is why there is no need to specify any address (hence the name 0-operand stack addressing). On the other hand, the register-based computer moves data into registers first. Operations are performed on this data, and then the results are stored into registers. This method makes full use of registers, hence the name: Register-based computing. Other significant differences between the two approaches can be summarized in the following table:

| Stack Based | Register Based |
|--|--|
| Smaller programs: <ul style="list-style-type: none"> • Requires less memory | Larger programs: <ul style="list-style-type: none"> • Uses more memory |
| Less processor and system complexity <ul style="list-style-type: none"> • Tries to strike a balance between RISC and CISC • Limits data on which operations are performed on to the top of the stack | CISC <ul style="list-style-type: none"> • Increase in processor complexity for low system complexity • Goal of a consistent and simple interface between hardware and software RISC <ul style="list-style-type: none"> • Increase in system complexity for low processor complexity • Goal of making processor faster by only reducing the number of instructions Modern processors have features from both CISC and RISC machines |
| Compatible with Reverse Polish Notation (e.g. $A B + C \times D E \times +$) | Uses the more traditional infix notation (e.g. $(A+B) * C + (D * E)$) |
| Not nearly as much research and development over the years | Popular due to: <ul style="list-style-type: none"> • Cheap cost of memory • Traditional way of doing computing • Lots of research and development over the years |

Table 2.3: Summary of Differences between Stack and Register Based Computing

CHAPTER 3

METHODOLOGY / PROJECT WORK

The objective of this project (to build a central processing unit using discrete TTL) will be achieved by following the steps listed in section 3.1. Equipment required for this project is listed in section 3.2 and the appendices. Section 3.3 describes the computer system that will be built in more detail. The next section, Section 3.4 explains the methods that were used to link the computer's software to its hardware. Finally, Section 3.5 explains the methods that were used to test and troubleshoot the system.

3.1 Procedure Identification

The planned procedure can be summarized as follows:

1. Obtain components
2. Plan layout of components on system cards
3. Wire wrap components on system cards
4. Test system cards separately
5. Connect system cards to each other via a backplane
6. Download the software into the system
7. Test and troubleshoot the complete system

3.2 Equipment Required

The equipment required for this project can be summarized as follows:

1. Wire wrapping tool and wires
2. Eurocards, backplane and card connectors
3. Electronic components (e.g. Logic ICs, resistors, ROM and RAM chips)

Note: For the sake of brevity, the complete list of components required is not listed here. A list of components for all system cards can be found in the appendices section.

3.3 The Mark 1 FORTH Computer Design

The computer system that will be built in this project is named the Mark 1 FORTH Computer. It was designed by Andrew Holme, an electronic engineer. The Mark 1 FORTH Computer has no microprocessor. It uses discrete TTL logic chips for its CPU. Other components of the computer system are also implemented using mainly TTL chips. Table 3.1 below lists the specifications of the Mark 1 FORTH Computer. [5]

| | |
|---------------------|---------------|
| Technology | TTL and HCMOS |
| Clock Speed | 1 MHz |
| Data Bus | 8-bit |
| Address Bus | 16-bit |
| Software | FIG-FORTH |
| ROM | 8 KB |
| RAM | 24 KB |
| Input/Output | RS-232 |

Table 3.1: Mark 1 FORTH Computer Specifications

The architecture of the Mark 1 FORTH Computer is shown in Figure 3.1 below.

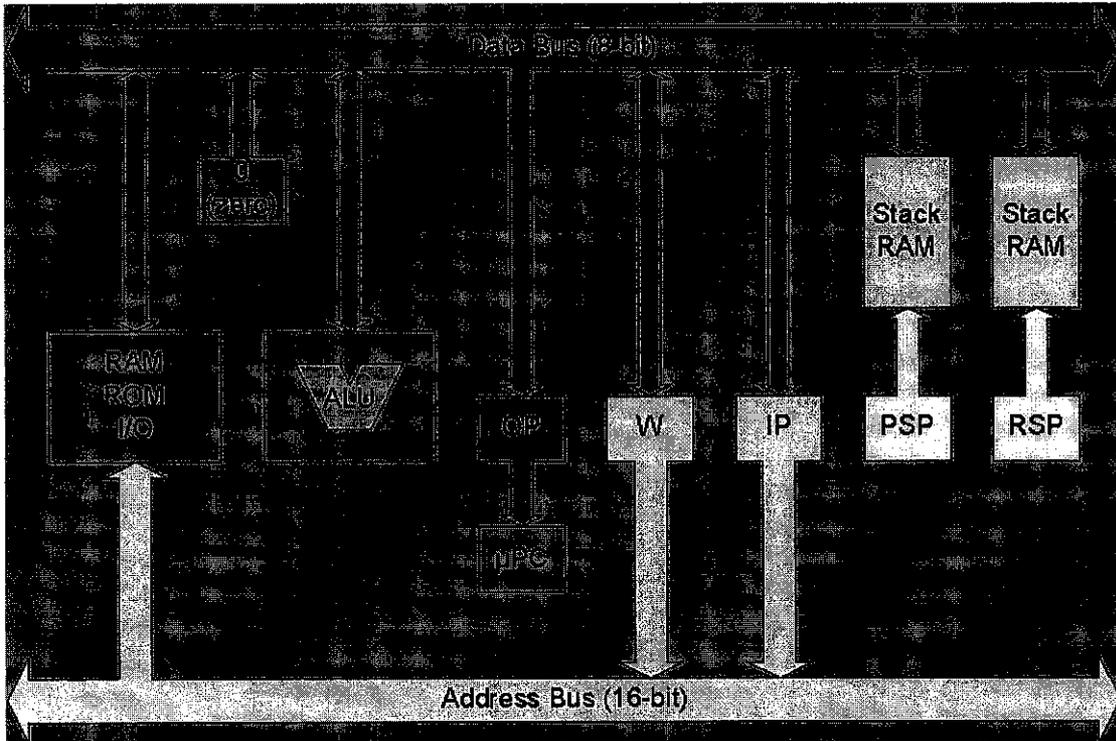


Figure 3.1: Mark 1 FORTH Computer Architecture

(Source: A. Holme; Mark 1 FORTH Computer; Internet:
<http://www.holmea.demon.co.uk/Mk1/Architecture.htm>; 2003)

Table 3.2 on the next page describes each of the components shown in the figure above in more detail.

| Module | Width (bits) | Description | Comments |
|-----------|--------------|-------------------------------|---|
| ALU | 8 | Arithmetic and logic unit | The ALU data path is a bottleneck. It takes four clock cycles to load the inputs, set the ALU function, and read the result. This is the least satisfactory aspect of the whole design. |
| OP | 8 | Operand register | OP is loaded into the uppermost 8 bits of μ PC. The lower 4 bits are reset to zero. |
| μ PC | 12 | Microcode program counter | |
| W | 16 | FORTH Working register | The 16-bit index registers, IP and W, support increment, decrement, and can address memory. |
| IP | 16 | FORTH Instruction Pointer | |
| PSP | 8 | FORTH Parameter stack pointer | The stack pointers, RSP and PSP, are 8-bit up/down counters feeding the A1-A9 address inputs of the stack RAMs. The least significant address input (A0) selects the upper or lower byte. Logically, the stacks are 16-bits wide by 256 words deep. The FORTH word length is 16 bits. |
| RSP | 8 | FORTH Return stack pointer | |
| Stack RAM | 16 | Dedicated stack RAM | |
| 0 | 8 | Force 00H on data bus | |

Table 3.2: Description of Mark 1 FORTH Computer Components

(Source: A. Holme; Mark 1 FORTH Computer; Internet: <http://www.holmea.demon.co.uk/Mk1/Architecture.htm>; 2003)

The Mark 1 FORTH Computer has a very elegant design. It resembles the generic stack computer that was discussed in Section 2.4 in some ways. However, in other ways, it improves on the generic stack computer's design.

The Mark 1 FORTH computer is built using nine system cards connected together via a DIN41612 64-way backplane. Together, these nine cards implement the architecture previously shown in Figure 3.1. The system cards are:

- System Clock and Instruction Decoder
- Microcode Sequencer
- Working (W) Index Register
- Instruction Pointer (IP) Index Register
- Stacks
- Memory
- Diode ROM
- Input/Output
- Arithmetic Logic Unit (ALU)

The complete schematics for the computer can be found in the appendices.

3.4 Software

The Mark 1 FORTH computer has both RAM and EPROM chips for its memory. The RAM chips are only used during normal operation of the computer. The EPROM chips however, are used not only during normal operation, but during the boot sequence, interrupts and resets. Therefore they need to be programmed before being placed on their respective cards. The table below shows the location of each memory chip:

| System Card | Memory Chip | Quantity |
|---------------------|---------------|----------|
| Memory | 6264 8KB RAM | 3 |
| | M2764 8KB ROM | 1 |
| Stacks | 6116 2KB RAM | 2 |
| Microcode Sequencer | M2764 8KB ROM | 1 |

Table 3.3: Location of Memory Chips

The first EPROM chip is the M2764 chip on the Microcode Sequencer card. This chip contains microcode routines for the computer and is referred to as the μ ROM. Andrew Holme provided complete software for programming this chip: source code for an assembler (written in C++), and the assembly code for the μ ROM. First of all, the assembler was compiled using Microsoft Visual C++. Using this assembler, the assembly code was then assembled: resulting in an Intel Hex image and a list of opcodes (these opcodes, which are formatted as MASM EQU statements are used later in the High-Level ROM programming). The Intel Hex image for the μ ROM was then used to program the M2764 ROM chip. A Topmax Universal Programmer was used for the programming process.

The second EPROM chip (another M2764 chip) is found on the Memory card. This chip is referred (by Andrew Holme) to as the High-Level ROM. The High-Level ROM contains the boot sequence for the computer. Microsoft Macro Assembler (MASM) and Microsoft Incremental Linker (LINK) were used to generate the Intel Hex image for this chip. The Intel Hex was then burned into the ROM using the same Topmax Universal Programmer as above. Assembly code was again provided by Andrew.

Another file of assembly code that was assembled for the Mark 1 FORTH computer is the FIG-FORTH software. This version of FORTH was originally for the 6502 computer. However, Andrew translated this version of FORTH into a version that works with his (Mark 1) computer. The assembly code for this translated version of FORTH is available on his website. MASM and LINK were once again used to generate the Intel Hex image of the assembly code. However, this Intel Hex image was not burned directly into a ROM chip. As in Andrew Holme's Mark 1, The FORTH will be downloaded via serial port into the computer's RAM during normal operation.

The flowchart on the next page graphically shows the procedures as explained above.

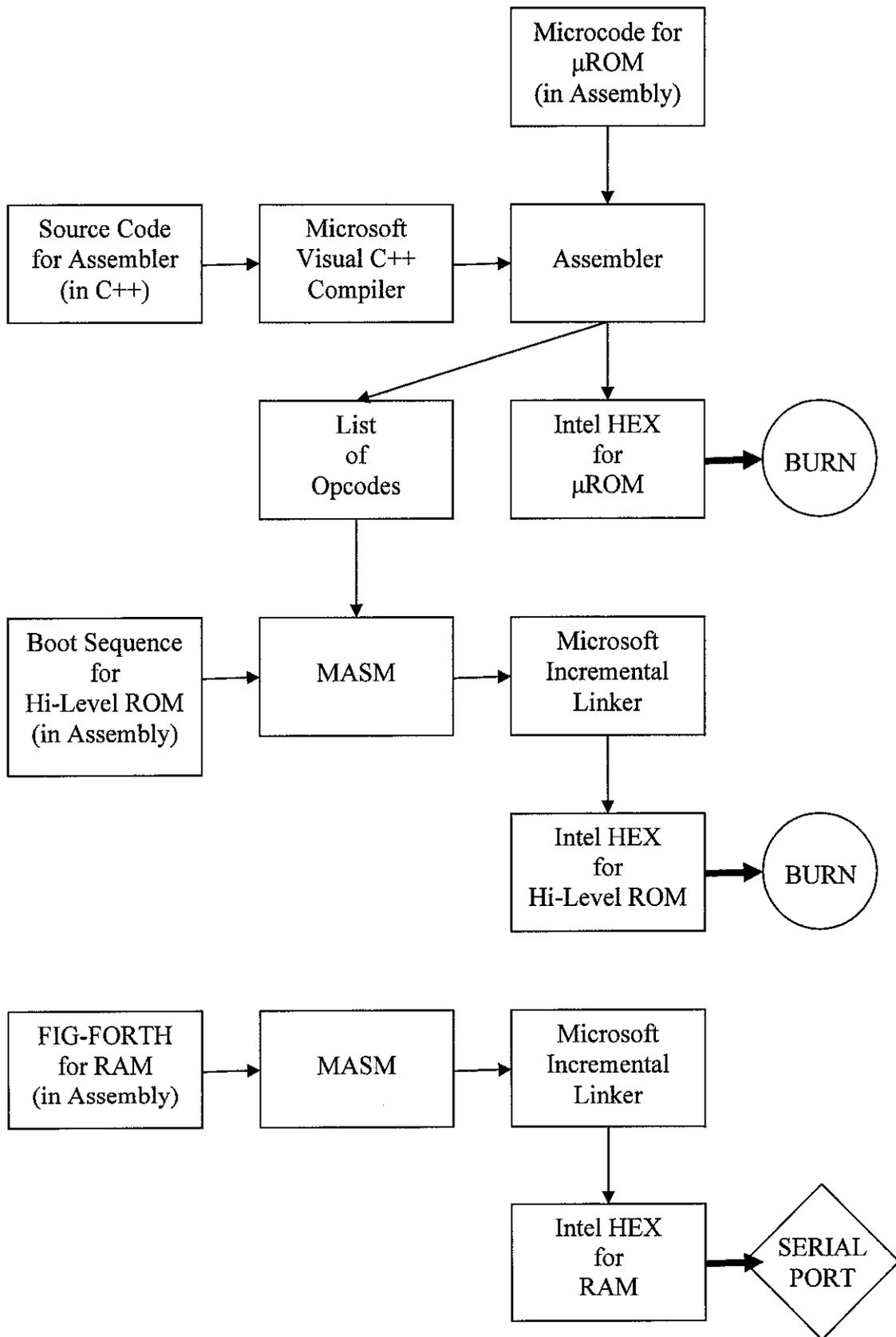


Figure 3.2: Procedures for Software

3.5 Testing and Troubleshooting

Wire wrapping of each system card was time consuming and tedious, but it was easy. The testing and troubleshooting process was the hardest part of completing the project. This section describes the process of testing and troubleshooting the computer.

The system cards were first tested individually. Each card was first subjected to continuity testing. Then each card was checked for short circuits. After these two tests were completed, a test circuit was used to logically test each card. This circuit was used to input control bits to each system card using DIP switches. A very low frequency clock signal (about 1 Hz) was then sent to the card being tested (except the System Clock and Instruction Decoder card). Results were observed either via the LEDs on the test circuit, an oscilloscope, or a logic probe. As each system card is different from the other cards, the control bits for each card are also different. However, the same circuit can be used. The most important thing is to ensure that the correct control lines are connected to the DIP switches. The picture below shows the logical test circuit. Figure 3.3 on the next page shows the schematic for the logical testing circuit.



Photograph 3.1: Logical Testing Circuit

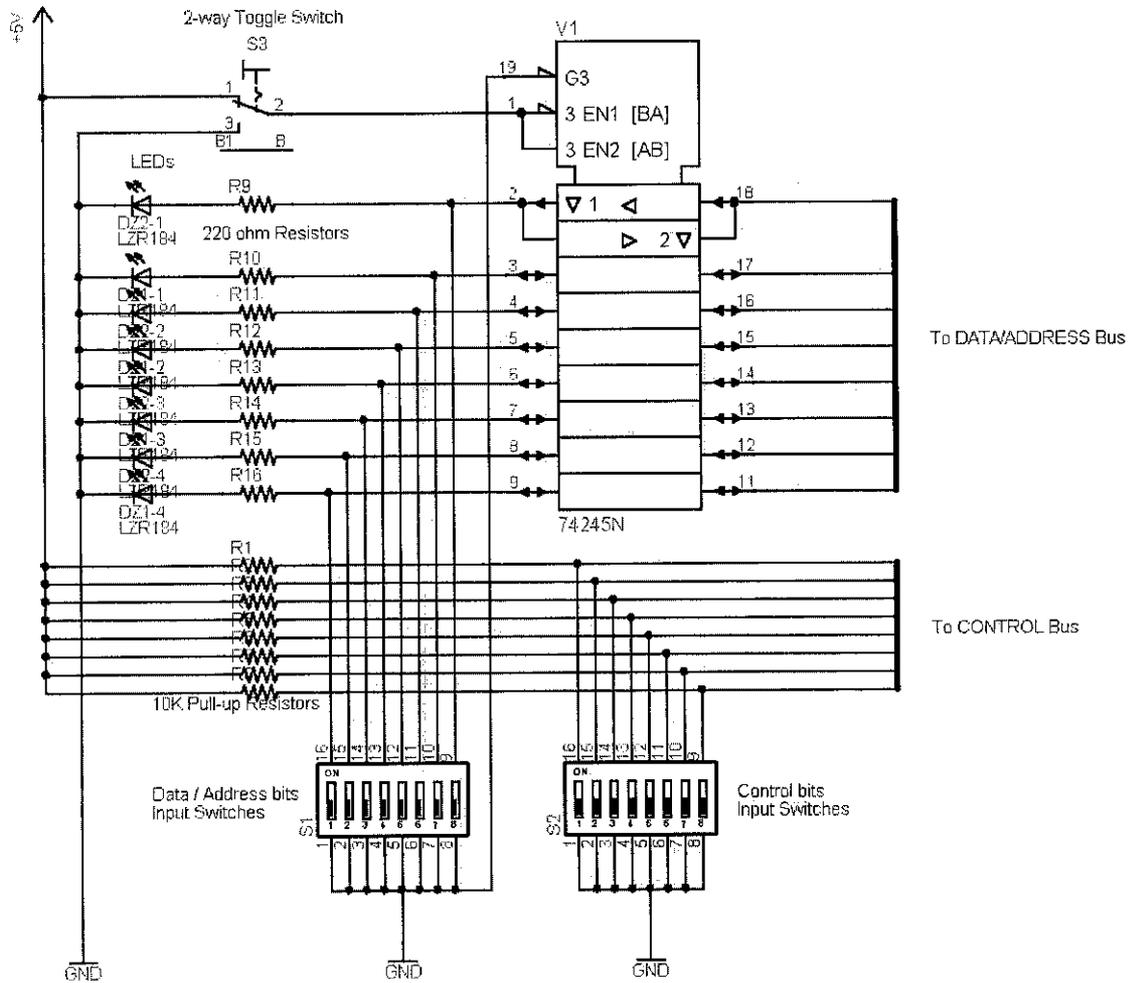


Figure 3.3: Schematic for Logical Testing Circuit

The sections below describe the individual test results from each system card:

3.5.1 ALU

The ALU is capable of handling 8-bit arithmetic and logical operations. It was tested by performing arithmetic calculations step by step. Test results have verified that the ALU is working properly. It transmits the correct results to the data bus. The ALU also has conditional flags that have been verified to be working properly. Table 3.4 below shows test results of an ADD operation done on the ALU card:

| Input A | Input B | Output F | Flags | | | |
|------------|------------|-------------|-------|------|-----|-----|
| | | | Sign | Cout | A=B | IRQ |
| 0000 0001 | 0000 0010 | 0000 0011 | 0 | 1 | 0 | 0 |
| 0101 0101 | 0010 1010 | 0111 1111 | 0 | 1 | 0 | 0 |
| 0111 0000 | 0000 1111 | 0111 1111 | 0 | 1 | 0 | 0 |
| 0111 1111 | 0111 1111 | 1111 1111 | 1 | 0 | 1 | 0 |

Table 3.4: Results from ADD Operation on ALU Card

3.5.2 System Clock and Instruction Decoder

This card has two functions. The first is to provide accurate clock signals to the clock buses. The second is to accurately decode control signals (u0-u7) that are input to this card. The card then transmits the decoded control signals to the control bus of the computer. Testing of this board has shown accurate CLK1 and CLK2 clock signals that are in quadrature with each other. All control signals have also been accurately decoded and transmitted to the control bus.

3.5.3 Microcode Sequencer

The microcode sequencer contains a microcode counter as well as the instruction memory stored on a ROM chip. The instruction memory stores 43 routines that will be used by the Mark 1 FORTH computer. However, a total of 256 routines can actually be stored. The instruction memory has been verified to accurately output the correct control bits according to instruction address. The microcode counter has also been verified to be working correctly. Table 3.5 below shows some test instruction addresses (inputs) and the respective control bits (outputs).

| Input | Output | |
|----------------------------|---------------------------|------------------------------|
| Instruction Address | Control Bits (HEX) | Control Bits (Binary) |
| 0000 | 30 | 0011 0000 |
| 0001 | 70 | 0111 000 |
| 0002 | 88 | 1000 1000 |
| 0003 | 88 | 1000 1000 |
| 0004 | 84 | 1000 0100 |
| 0005 | 21 | 0010 0001 |
| 0006 | 88 | 1000 1000 |
| 0007 | 61 | 0110 0001 |
| 0008 | 28 | 0010 1000 |
| 0009 | 89 | 1000 1001 |
| 000A | 68 | 0110 1000 |
| 000B | 89 | 1000 1001 |
| 000C | F3 | 1111 0011 |
| 000D | 25 | 0010 0101 |
| 000E | 88 | 1000 1000 |
| 000F | B0 | 1011 0000 |

Table 3.5: Test Results from Microcode Sequencer

3.5.4 Diode ROM

The Diode ROM's purpose is to accurately decode control signals for the ALU card. An input of 4 bits (u0-u3) is decoded on this card to produce a possible of 16 functions for the ALU card. Each function consists of a series of 8 bits, which will be used to set the function (e.g. ADD, SUB, AND) for the ALU. The table below shows the functions for the ALU.

| | OPP | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
|-------|-----|----|----|----|----|----|----|----|----|
| 0 | ADD | 1 | 0 | 0 | 1 | L | x | H | H |
| 1 | ADC | 1 | 0 | 0 | 1 | L | x | x | L |
| 2 | SUB | 0 | 1 | 1 | 0 | L | x | L | H |
| 3 | SBB | 0 | 1 | 1 | 0 | L | H | x | L |
| 4 | ASL | 0 | 0 | 1 | 1 | L | x | H | H |
| 5 | ROL | 0 | 0 | 1 | 1 | L | x | x | L |
| 6 | | | | | | | | | |
| 7 | | | | | | | | | |
| 8 (a) | 0< | 1 | 1 | 1 | 1 | H | L | x | x |
| 8 (b) | A | 1 | 1 | 1 | 1 | H | x | x | x |
| 9 | B | 0 | 1 | 0 | 1 | H | x | x | x |
| 10 | AND | 1 | 1 | 0 | 1 | H | x | x | x |
| 11 | OR | 0 | 1 | 1 | 1 | H | x | x | x |
| 12 | NOT | 0 | 0 | 0 | 0 | H | L | x | x |
| 13 | XOR | 0 | 1 | 1 | 0 | H | x | x | x |
| 14 | A=B | 1 | 0 | 0 | 1 | H | x | x | x |
| 15 | | | | | | | | | |

Table 3.6: ALU Functions [5]

The Diode ROM card has been verified to accurately set the function for all 16 possible functions.

3.5.5 W and IP Index Registers

The 2 index register cards are exactly the same, consisting of counters that are cascaded to be 16-bit. They have been tested to count in both directions (increment/decrement), and also to address memory accurately.

3.5.6 Stacks

This card contains the RAM chips that will be used as stack memory. This card also contains counters that are used as stack pointers (either parameter stack pointer or return stack pointer). It has been verified that data can be accurately pushed and popped from each of the stacks. Both stack pointers have also been shown to be able to accurately count in both directions (increment/decrement).

3.5.7 Memory

The memory card was tested by reading and writing sample data to the memory. In other words, this card has been successfully tested to be able to handle read and write operations. The tables below show sample test data that was written to and read from the memory card:

| Address (Memory Location) (in HEX) | Data Bits | | |
|---------------------------------------|-----------|----------|-----------|
| | Write | Read | Verified? |
| 2000 | 11110000 | 11110000 | Correct |
| 4000 | 10101010 | 10101010 | Correct |
| 6000 | 11111111 | 11111111 | Correct |
| 2001 | 00001111 | 00001111 | Correct |
| 4001 | 10101010 | 10101010 | Correct |
| 6001 | 01010101 | 01010101 | Correct |

Table 3.7: Memory Card Test Results

3.5.8 Input / Output

The Input / Output card was logically tested only up to a certain extent. This is because the 82C51A chip on this card requires initialization codes that would have been quite tedious to input manually. Therefore, this chip was not logically tested. However, the other parts on the board were verified to be working properly. This included the logic gates for decoding signals, the hex buffer (74LS368), the 14-stage clock divider (HEF4060BP) and the data transceiver (74LS245). During testing for the 14-stage clock divider, an extra component was added to the I/O board. This is the 74HCT04 hex inverter chip. The chip was added to generate a better clock signal for the 82C51A USART. Before this chip was added, the clock signal was not a clean one and was of low magnitude. Putting the clock signal through two hex inverters ensured that it was clean and about 5V (peak to peak) in magnitude. Testing for the 82C51A USART was planned for later, after the system cards had been connected together.

3.5.9 The Complete System

With all the system cards verified to be working properly (at that current point in time), all the cards were connected together using the backplane. The serial cable was connected to a personal computer and Hyperterminal was started (Hyperterminal is the program that is used to interface the Mark 1 and the personal computer). The Mark 1 was then switched on. However, no results were detected on Hyperterminal. The computer system was not properly working yet. The system now required troubleshooting as a whole, instead of individual card testing.

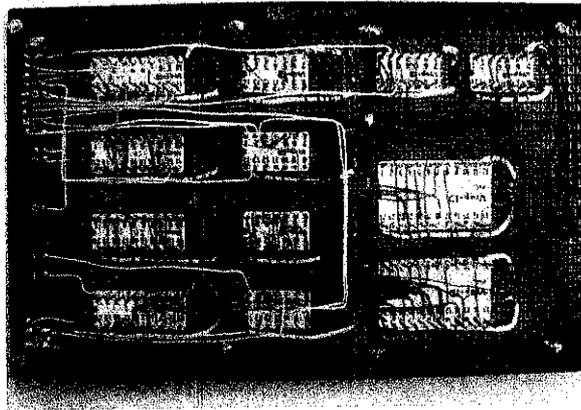
Troubleshooting the system as a whole was a problem. Rightfully, since every system card worked individually the system should work when connected together. However, this was not so. Using a combination of an oscilloscope, a logic probe and a list of what could possibly have gone wrong; the system was checked for errors. After weeks of searching, two errors were detected:

- The cable connecting the IO card to the PC was not properly wired
- The Octal buffer on the Diode ROM card (74HCT244) was not suitable for the Diode ROM card.

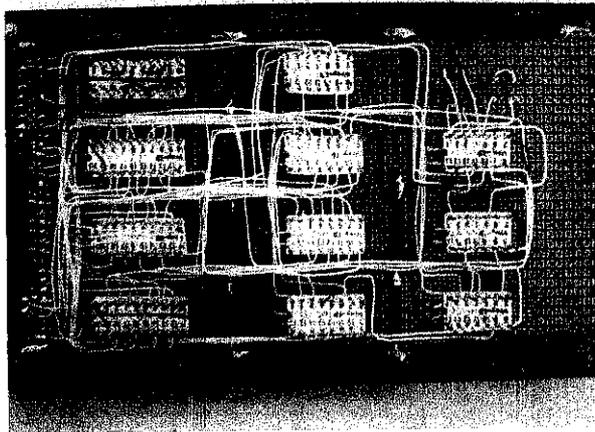
Regarding the cable, Andrew Holme's schematics show a DB25 socket at the IO card. This socket was to be used with a null modem DB25 – DE 9 cable (and connected to the serial port on a PC). However, the assumption was made that the schematic was showing a direct connection to the PC. Therefore the cable was wrongly wired. The problem has now been rectified, and the IO card schematic currently shows a direct connection to the serial port on any normal computer.

Regarding the octal buffer on the Diode ROM card, a 74HCT244 chip had previously been used. This was to ensure compatibility with the other TTL chips. However, there was a problem with this configuration. The VOL(max) from the 74HCT138 decoders added with the voltage drop across the diodes exceeds the VIL(max) of either the 74LS244 or 74HCT244 chips. Therefore, in order for the Diode ROM to work, a 74HC244 chip must be used. This problem had not been detected during individual testing of cards. However, when the cards were assembled together on the backplane, this problem caused the system to produce unexpected results. Once the 74HCT244 chip was replaced with a 74HC244 chip, the system worked fine.

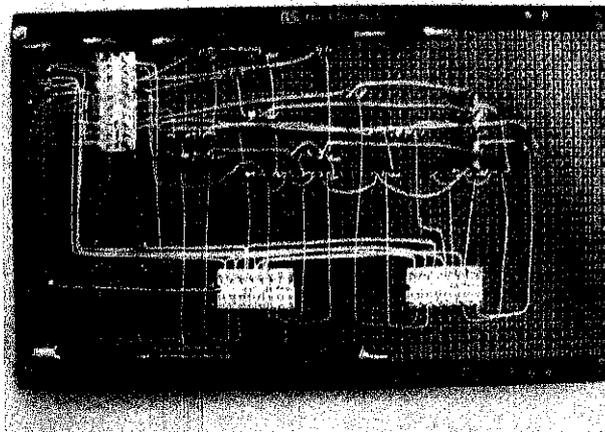
The following pictures show the completely wire-wrapped cards:



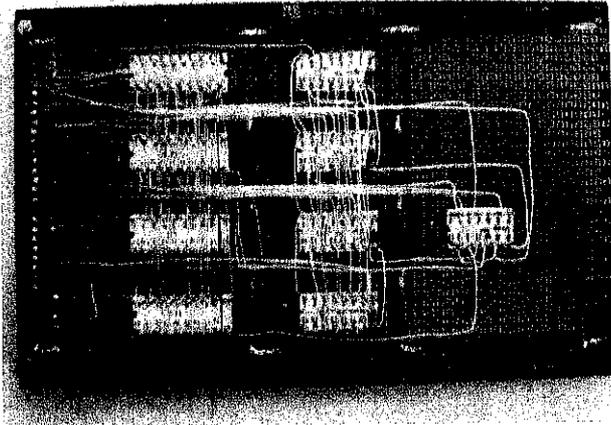
Photograph 3.2: Underside of ALU



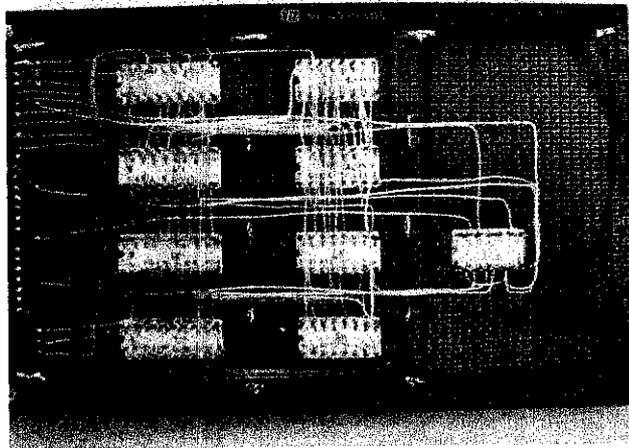
Photograph 3.3: Underside of System Clock and Instruction Decoder



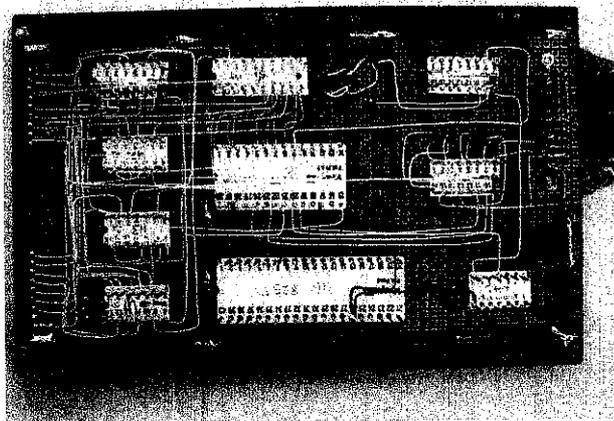
Photograph 3.4: Underside of Diode ROM



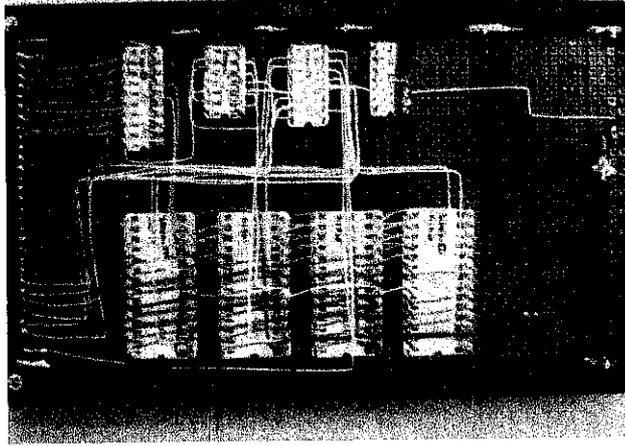
Photograph 3.5: Underside of IP Index Register



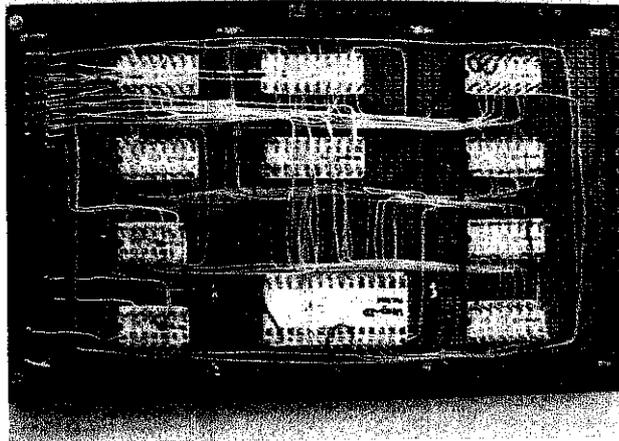
Photograph 3.6: Underside of W Index Register



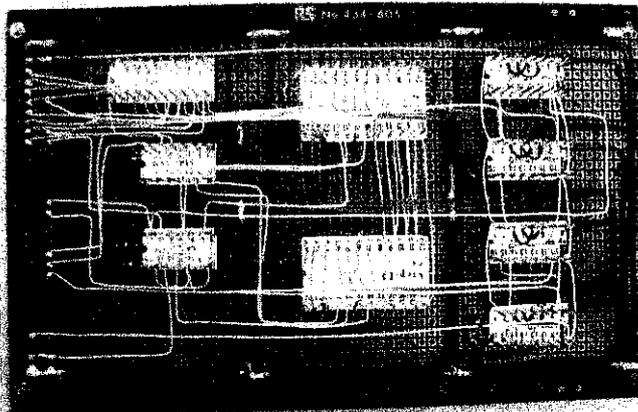
Photograph 3.7: Underside of Input/Output Card



Photograph 3.8: Underside of Memory Card



Photograph 3.9: Underside of Microcode Sequencer



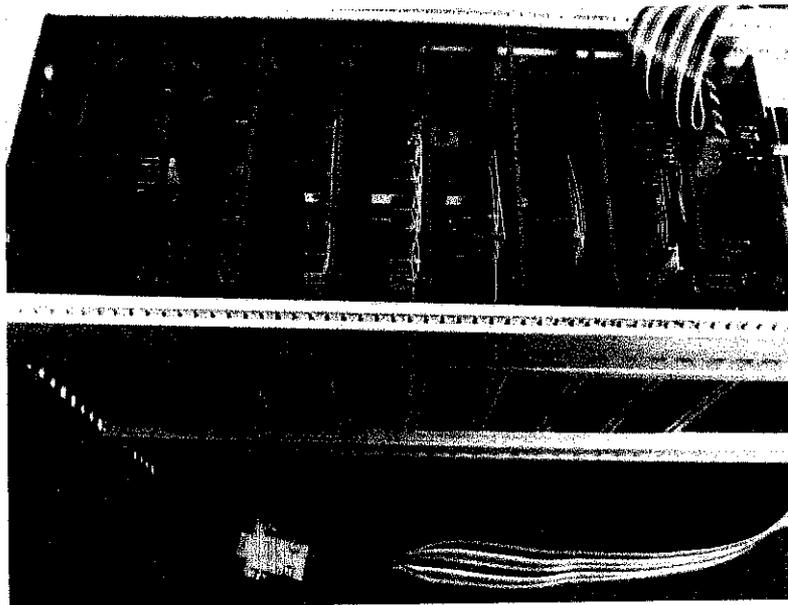
Photograph 3.10: Underside of Stacks Card

CHAPTER 4

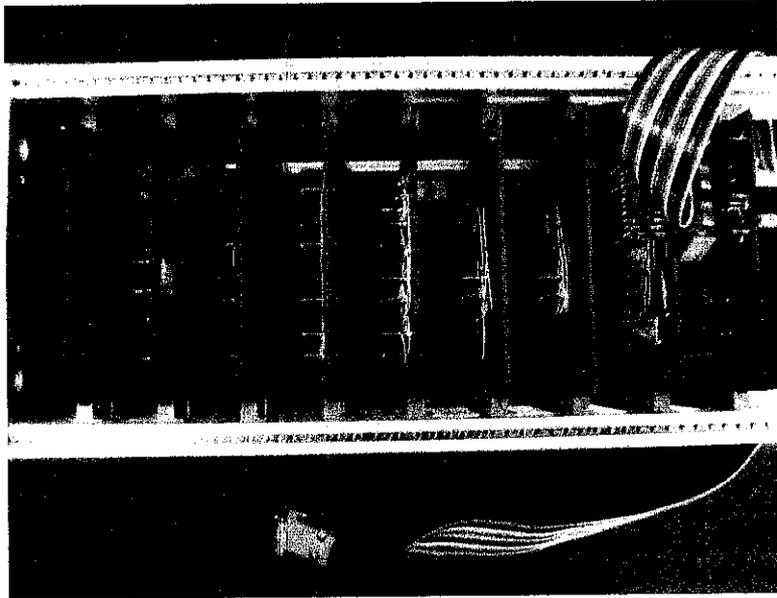
RESULTS AND DISCUSSION

4.1 Results

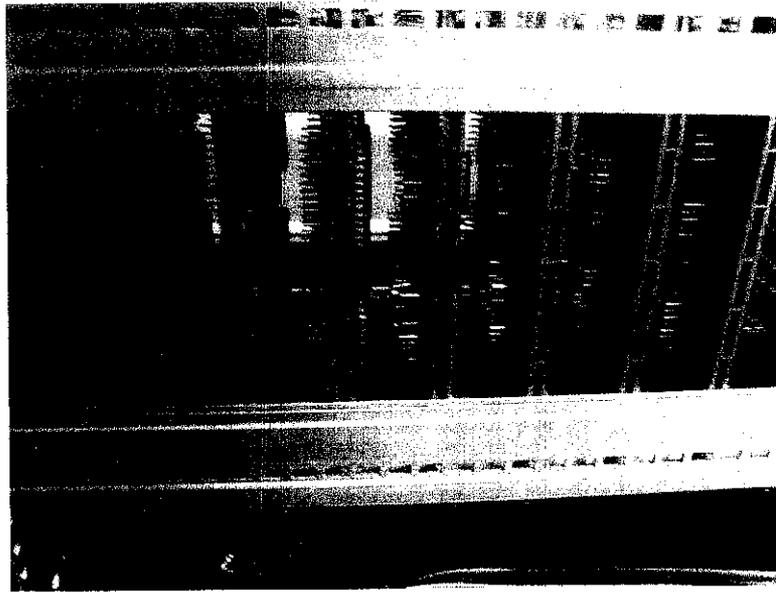
The Mark 1 FORTH computer is now fully operational. The computer system has been tested for about 2 weeks now, with no errors. Being a FORTH based computer system that can be programmed in anyway that the programmer wants, it is obviously impossible to fully show the capabilities of the computer in a report. Therefore, this section will focus on the results of some simple instructions that can be performed on the Mark 1. The following pictures show the complete Mark 1 FORTH computer:



Photograph 4.1: Angled View of the Mark 1 FORTH Computer



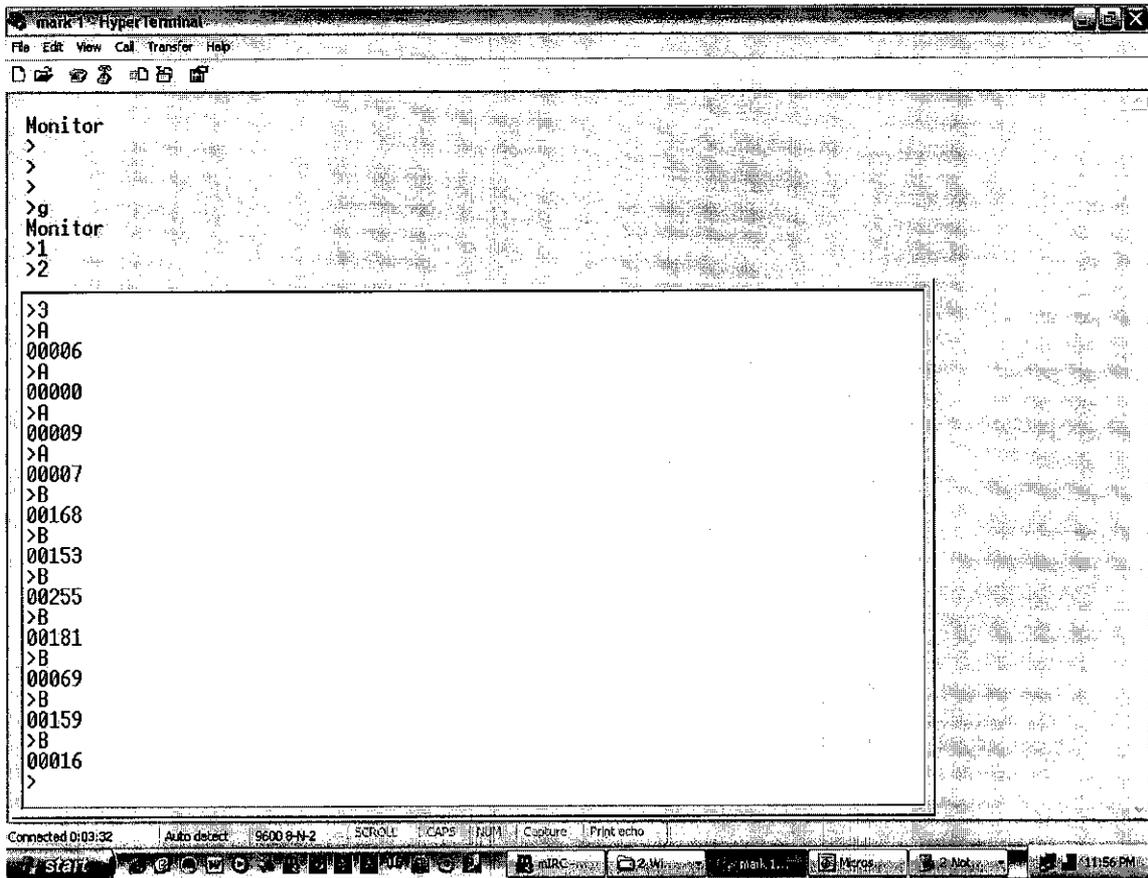
Photograph 4.2: Top View of the Mark 1 FORTH Computer



Photograph 4.3: Side View of the Mark 1 FORTH Computer

The computer draws between 0.9 to 1.1 amperes of current, operating at 5V. There are no heat problems, as all the chips remain cool during operation.

The screenshot below shows the results at Hyperterminal when the computer is booted up:



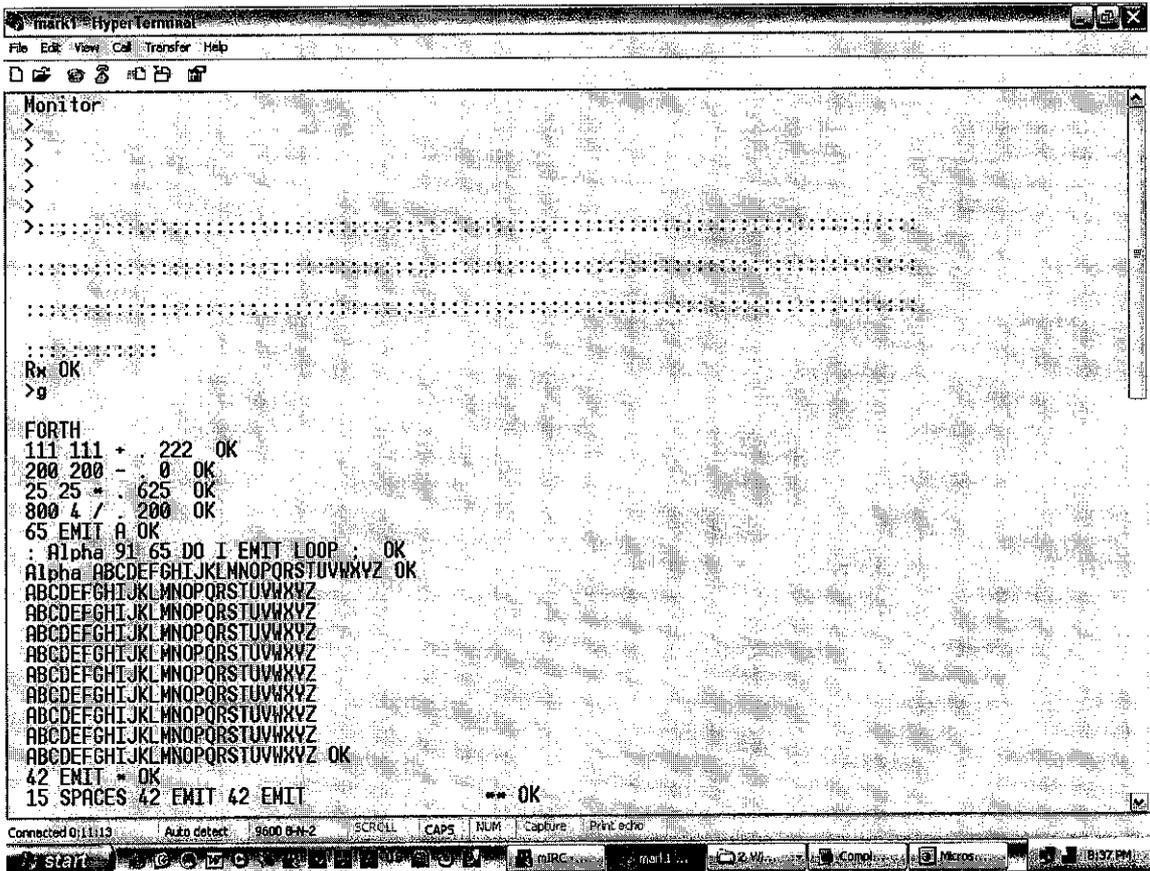
Screenshot 4.1: Initial Screen

As seen above, the first words to greet the user are:

Monitor

>

If “Enter” is pressed, the computer will continue to display the “>” symbol, as seen above. Entering the character “A” followed by a hexadecimal character results in the computer displaying the decimal value of the character entered. Entering “B” followed by a byte of hexadecimal characters (e.g. F3, 07, A5) results in the computer displaying the decimal value of the byte entered. This can also be seen in the screenshot above.

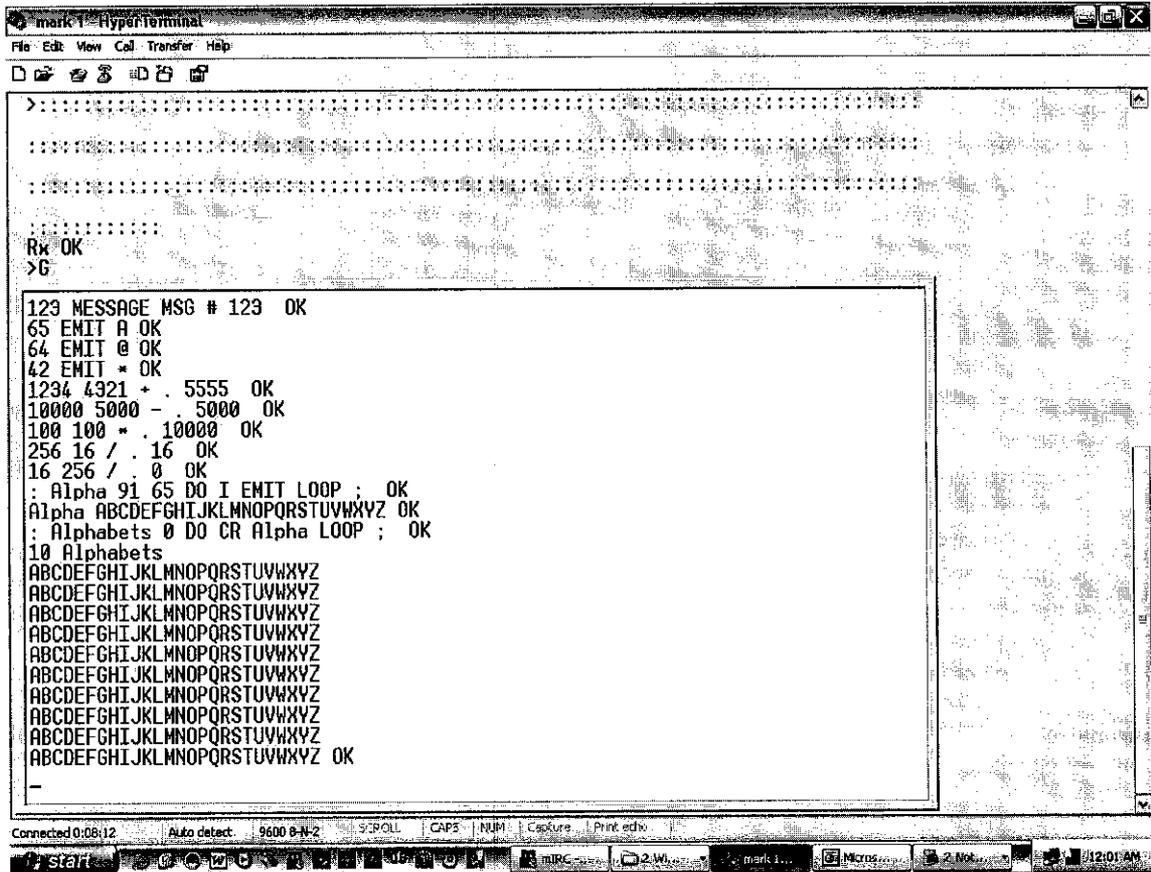


Screenshot 4.2: Simple FORTH

The screenshot above shows the results of loading the FIG-FORTH software into the Mark 1. When the FIG-FORTH is being downloaded into the system, it sends out a stream of “.....” characters, followed by “Rx OK” if the download was successful. Hitting the “G” character starts the FORTH software. The screenshot shows sample arithmetic operations being done on the computer. The results are correct:

| | | | | |
|-----|---|-----|---|-----|
| 111 | + | 111 | = | 222 |
| 200 | - | 200 | = | 0 |
| 25 | * | 25 | = | 625 |
| 800 | / | 4 | = | 200 |

The command “ : Alpha 91 65 DO I EMIT LOOP ; ” seen in the Screenshot 4.2 basically tells the system to save a routine displaying all the characters in the alphabet (A to Z) as a word called “Alpha”. A word in FORTH refers to a subroutine that can be called by the system at a later time. This is demonstrated in the next line. When “Alpha” is entered to the system, it outputs “ABCDEFGHIJKLMNOPQRSTUVWXYZ”.

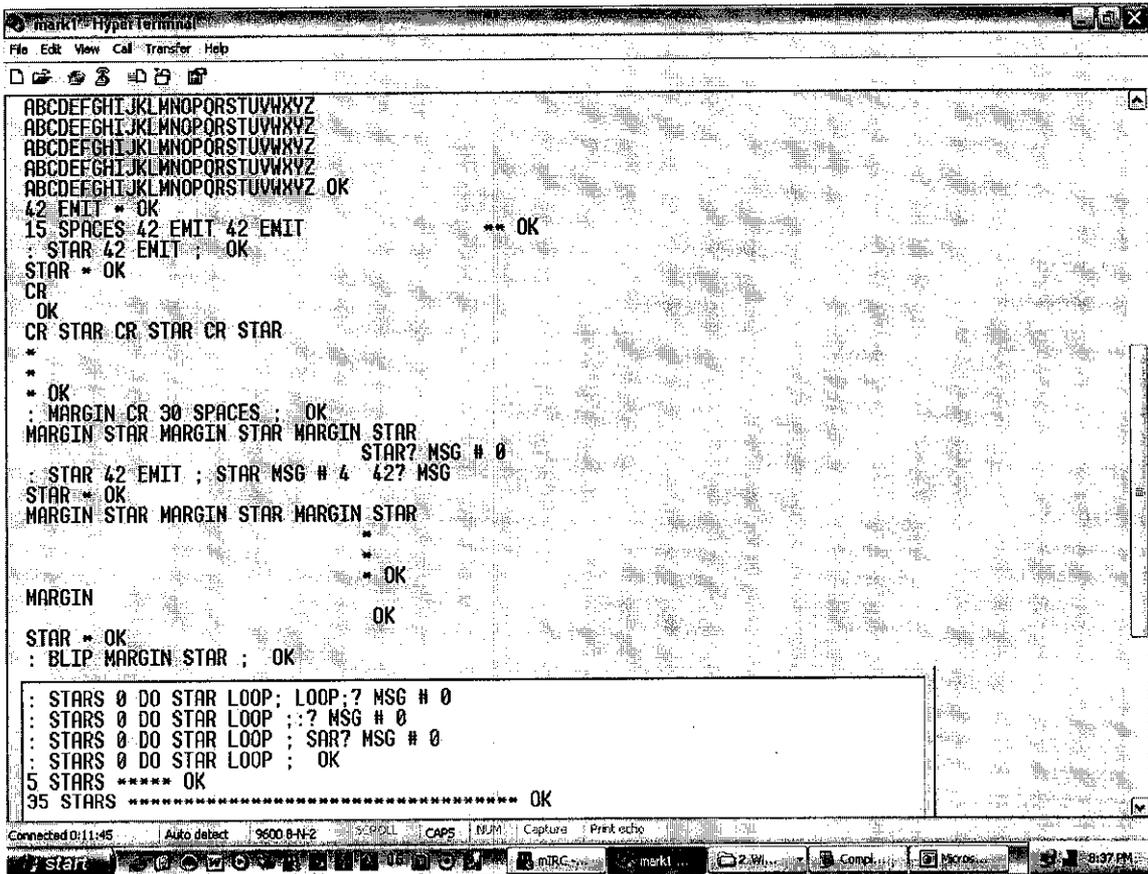


Screenshot 4.3: Looping the Alphabet

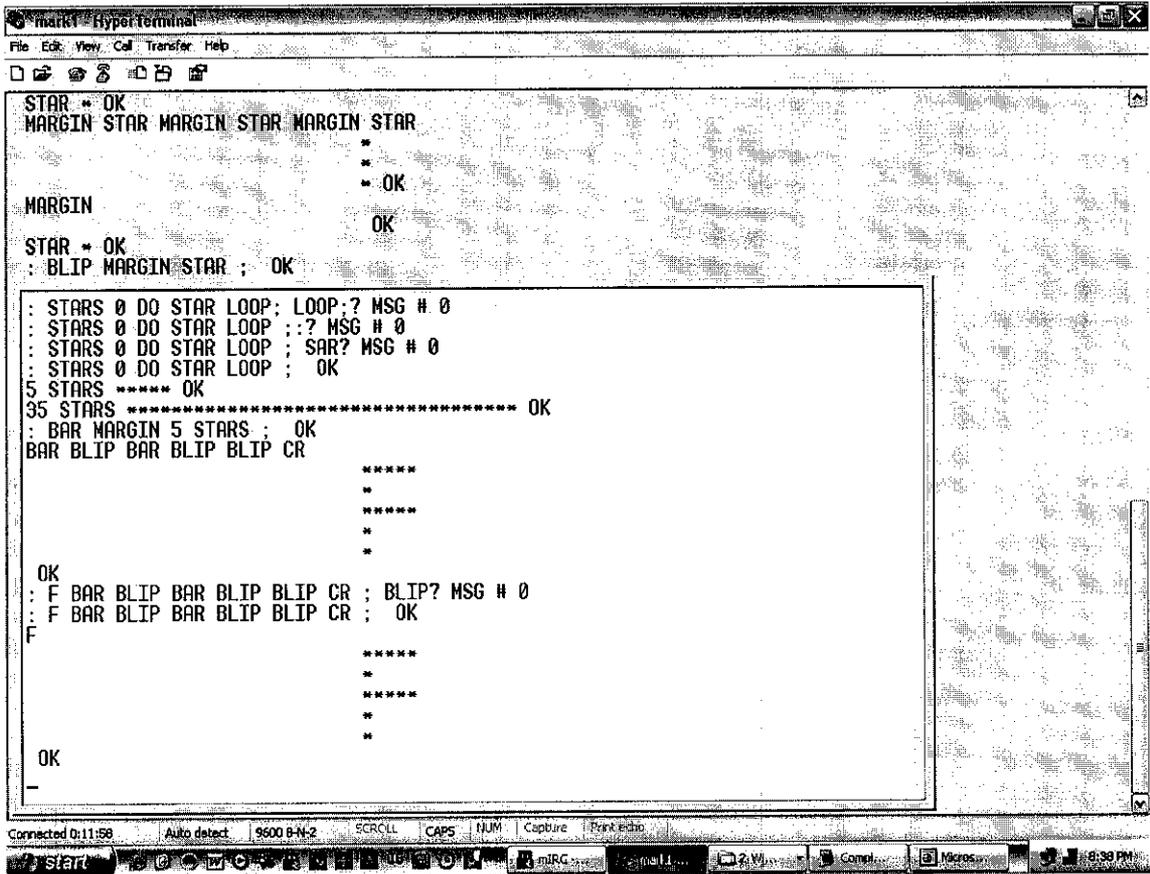
Screenshot 4.3 shows some new commands with their results. The command “65 EMIT” asks the computer to display the character represented in ASCII by the decimal number 65, which is “A”. Similarly, any other character represented in ASCII can be displayed by the computer. In the screenshot above, “@” and “*” are displayed.

The command “ : Alphabets 0 DO CR Alpha LOOP ; ” makes use of the previously defined “Alpha” word. This command creates a new routine called “Alphabets” that displays “ABCDEFGHIJKLMNOPQRSTUVWXYZ” on new lines for a specified number of times. In the Screenshot 4.3, “10 Alphabets” is entered. The system duly outputs the alphabet 10 times, each time on a new line.

Finally, the next two screenshots show the use of words in displaying the letter “F” using “*” characters:



Screenshot 4.4: Displaying “F” Part 1



Screenshot 4.5: Displaying “F” Part 2

The previous 2 screenshots show the following words being defined:

STAR

STARS

MARGIN

BLIP

BAR

F

[6]

Each of these words makes use of words that have been previously been defined. The final word “F” displays the letter “F” using “*” characters, as seen above. Programming in FORTH is usually done this way: Creating simple words, then expanding on those words to create more and more complex words and finally applications. In this way, FORTH is a simple and small language, yet amazingly powerful.

4.2 Changes to the Mark 1 FORTH Computer

In the course of doing this project, the original design by Andrew Holme was adhered to as much as possible. However, several changes were made due to a variety of reasons. These changes are documented in this section.

4.2.1 Primary Usage of LS TTL Chips

The original Mark 1 design primarily utilizes HC chips, with only a few TTL chips. However in this version of the Mark 1 computer system, LS TTL chips are primarily used. HCT chips are used wherever the schematics state that a HC chip should be used. The only 2 differences are the Microcode Sequencer (where a 74HC107 chip is used) and the Diode ROM (where a 74HC244 chip is used). Although the original Mark 1 used mostly HC chips, this version (using mostly TTL chips) also works. This shows that the design is not component dependent, a hallmark of a robust design.

4.2.2 Addition of 74HCT04 Hex Inverter to Input / Output Card

The original Input / Output card connects an output (pin 9) from the HEF4060BP clock divider directly into the clock input (pin 20) of the 82C51A USART. In our IO card however, the clock signal obtained from the 4060 clock divider was not a clean square wave. It was therefore deemed necessary for the output from the clock divider (pin 9) to be passed through two inverters first, before sending it to the clock input (pin 20) of the USART. The signal obtained after the inverters is a much better square wave.

4.2.3 RS-232 Null Modem Cable

The original Input / Output card has a DB25 socket for connecting the Mark 1 to a DE9 serial port on a personal computer. This requires the use of a DB25 to DE9 null modem cable. However, the input / output card that was built here has a “built-in” cable directly to a DE9 female connector. This connector just needs to be plugged in directly to a DE9 serial port and the two computers can start communicating. In short, there is no need for a null modem cable anymore. It is already “built-in”.

4.3 Performance

It is difficult to ascertain the performance of the Mark 1 computer, because there is no FORTH-based computer readily available to carry out a comparative study. However, Andrew Holme’s website compares the performance of the Mark 1 to another FORTH computer that he built: the Mark 2. The table on the next is reproduced from his website. It compares the number of CPU cycles taken by each computer to perform a FORTH primitive. Although the Mark 2 is a more advanced computer, the Mark 1 performs reasonably well, beating the Mark 2 in some categories.

| FORTH Primitive | Number of Cycles | | |
|--------------------|------------------|-----------|------------|
| | Mark 1 | Mark 2 | Difference |
| Enter | 14 | 16 | -2 |
| ;S | 12 | 12 | 0 |
| LIT | 14 | 16 | -2 |
| EXECUTE | 7 | 6 | 1 |
| (DOES) | 19 | 22 | -3 |
| BRANCH | 14 | 10 | 4 |
| 0BRANCH | 21 | 16 | 5 |
| (LOOP) | * 29/32 | * 24/26 | |
| (DO) | 19 | 24 | -5 |
| LEAVE | 15 | 16 | -1 |
| R> | 13 | 16 | -3 |
| >R | 13 | 16 | -3 |
| R | 12 | 14 | -2 |
| AND | 19 | 18 | 1 |
| OR | 19 | 18 | 1 |
| XOR | 19 | 18 | 1 |
| + | 20 | 18 | 2 |
| - | | 18 | |
| 0= | 19 | 16 | 3 |
| 0< | 17 | 16 | 1 |
| DUP | 14 | 14 | 0 |
| SWAP | 21 | 20 | 1 |
| DROP | 10 | 10 | 0 |
| OVER | 16 | 16 | 0 |
| @ | 14 | 14 | 0 |
| ! | 16 | 18 | -2 |
| C@ | 13 | 14 | -1 |
| C! | 14 | 18 | -4 |
| D+ | 37 | 32 | 5 |
| NEGATE | 17 | 16 | 1 |
| DNEGATE | 25 | 26 | -1 |
| U* | * 325/613 | * 118/182 | |
| U/ | 531 | * 190/222 | |

* Min/Max

Table 4.1: Performance Comparison between Mark 1 and Mark 2 Computers [7]

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

The project has been fully completed and the objectives fully achieved. A fully operational stack-based computer has been built. The computer runs at 1 MHz, has 8 kilobytes of ROM and 24 kilobytes of RAM. FIG-FORTH is the software used on the system. FORTH is used as an assembler, a programming language and as the operating system on the computer.

5.1 Recommendations for Future Work

The next step for this project will be to use the 8255 chip (currently unused) to display further Input / Output capabilities. The Input / Output card has a socket and the architecture for the 8255 chip. However, neither software nor hardware is configured to use the chip. The 8255 could be used to further show the Mark 1's capabilities. For example, the 8255 could be used to light up LEDs, or perhaps control an electrical device (e.g. a motor). FORTH is after all, a language originally written for control purposes.

Another recommendation would be to build a front panel for the computer. This front panel would have a series of LEDs to show the status of all the lines on the backplane. For example, 8 LEDs to display the status of the Data Bus, 16 LEDs for the Address Bus, and other LEDs for decoded signals and the Control Bus. This would be particularly good for demonstration purposes.

A standalone power supply could also be built for the Mark 1. Currently, it is drawing power from a laboratory power supply using crocodile clips and wires. Since the computer draws only about 1 ampere of current, a standard ATX Power Supply Unit (normally found in Pentium 4 computers) could be modified to constantly supply 5V to it. The Mark 1 could then be plugged in directly to a typical electrical power socket and operate. This would greatly increase its portability.

A final enhancement would be to build a case to store the Mark 1 computer in. The currently used card cage offers very little protection against dust, insects and particularly water. A large computer casing could be modified to fit the Mark 1 computer. Furthermore, the modified ATX Power Supply Unit (as above) would also fit perfectly into the computer casing. With the added protection and cooling (from the computer casing's fans), the Mark 1 would be a very reliable computer for many years to come.

REFERENCES

- [1] P. Koopman, Stack Computers: The New Wave, Internet Book:
http://www.cs.cmu.edu/~koopman/stack_computers/, 1989
- [2] M. Mano, C. Kime, Logic and Computer Design Fundamentals, Pearson
Prentice Hall, 2004, pgs 430-444
- [3] M. Mano, C. Kime, Logic and Computer Design Fundamentals, Pearson
Prentice Hall, 2004, pgs 450-455
- [4] Wikipedia, FORTH, Internet Web Page:
<http://en.wikipedia.org/wiki/FORTH>, 2006
- [5] A. Holme, Mark 1 FORTH Computer, Internet Web Page:
<http://www.holmea.demon.co.uk/Mk1/Architecture.htm>, 2003
- [6] L. Brodie, Starting FORTH, FORTH, Inc., 1981, pgs 10-13
- [7] A. Holme, Mark 2 FORTH Computer, Internet Web Page:
<http://www.holmea.demon.co.uk/Mk2/Architecture.htm>, 2003

APPENDICES

APPENDIX A: LIST OF COMPONENTS

Card 1: Arithmetic Logic Unit (ALU)

| Quantity | Part Number | Description |
|----------|-------------|----------------------|
| 3 | 74LS377 | 8-bit register |
| 2 | 74LS00 | Quad NAND Gates |
| 2 | 74LS181 | 4-bit ALU |
| 1 | 74LS86 | Quad XOR Gates |
| 1 | 74LS32 | Quad OR Gates |
| 1 | 74LS242 | Quad Bus Transceiver |
| 1 | 74LS74 | Dual D Flip Flops |
| 1 | 74LS244 | Octal Buffer |
| 1 | 1K Resistor | Pull-up Resistor |

Card 2: Diode ROM

| Quantity | Part Number | Description |
|----------|-------------|--|
| 2 | 74HCT138 | 3 to 8 Inverting Decoder/Demultiplexer |
| 1 | 74HC244 | Octal Buffer |
| 30 | 1N4001 | Diode |

Card 3: System Clock and Instruction Decoder

| Quantity | Part Number | Description |
|-----------------|-----------------------|---|
| 4 | 74LS244 | Octal Buffer |
| 2 | 74LS155 | Dual 1 of 4 Decoder/Demultiplexer |
| 1 | 74LS138 | 3 to 8 Inverting Decoder/Demultiplexer |
| 1 | 74HC107 | Dual JK Negative Edge Triggered Flip Flop |
| 1 | 74LS04 | Hex Inverter |
| 1 | 74LS32 | Quad OR Gates |
| 1 | 74LS08 | Quad AND Gates |
| 1 | 2 MHz XTAL | 2 MHz Crystal Oscillator |
| 2 | 470 Ω Resistor | Resistor |
| 1 | 22 pF Capacitor | Ceramic Capacitor |

Card 4: Stacks

| Quantity | Part Number | Description |
|-----------------|--------------------|--|
| 1 | 74LS245 | Octal Transceiver |
| 4 | 74LS169 | Modulo Binary (16) Synchronous Counter |
| 2 | HM6116ALP | 2KB SRAM |
| 1 | 74LS32 | Quad OR Gates |
| 1 | 74LS08 | Quad AND Gates |

Card 5: Memory and Power On Reset Circuit

| Quantity | Part Number | Description |
|-----------------|------------------------|----------------------------|
| 3 | UT6264CPC | 8KB SRAM |
| 1 | M2764 | 8KB EPROM |
| 1 | 74LS155 | Dual 1 of 4 Demultiplexers |
| 1 | 74LS245 | Octal Transceiver |
| 1 | 74LS00 | Quad NAND Gates |
| 1 | 74LS132 | Quad NAND Schmitt Triggers |
| 1 | 10 Ω Resistor | Resistor |
| 1 | 27 k Ω Resistor | Resistor |
| 1 | 10 μ F Capacitor | Ceramic Capacitor |
| 4 | 1 k Ω Resistor | Pull-up Resistor |
| 1 | Push Button | RESET Push Button |

Card 6: Microcode Sequencer

| Quantity | Part Number | Description |
|-----------------|--------------------|--|
| 3 | 74LS163 | 4-bit synchronous counter |
| 1 | 74LS377 | 8-bit register |
| 1 | M2764 | 8KB EPROM |
| 1 | 74LS173 | 4-bit register |
| 1 | 74LS169 | Modulo Binary (16) Synchronous Counter |
| 1 | 74LS02 | Quad NOR Gates |
| 1 | 74LS08 | Quad AND Gates |
| 1 | 74LS244 | Octal Buffer |
| 1 | 74LS151 | 1 of 8 Multiplexer |

Card 7: Working (W) Index Register

| Quantity | Part Number | Description |
|-----------------|--------------------|--|
| 4 | 74LS244 | Octal Buffer |
| 4 | 74LS169 | Modulo Binary (16) Synchronous Counter |
| 1 | 74LS32 | Quad AND Gates |

Card 8: Instruction Pointer (IP) Index Register

| Quantity | Part Number | Description |
|-----------------|--------------------|--|
| 4 | 74LS244 | Octal Buffer |
| 4 | 74LS169 | Modulo Binary (16) Synchronous Counter |
| 1 | 74LS32 | Quad AND Gates |

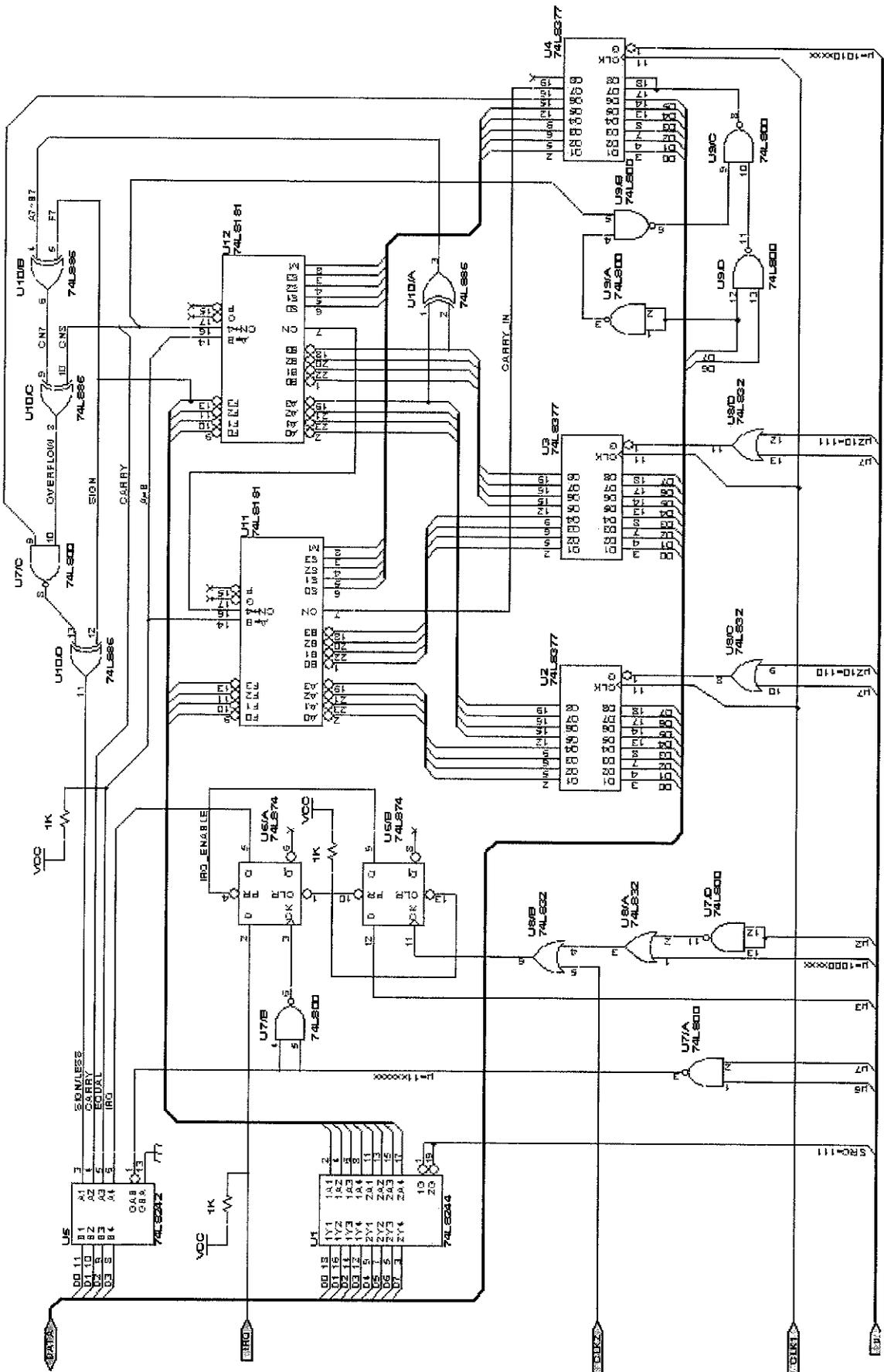
Card 9: Input/Output

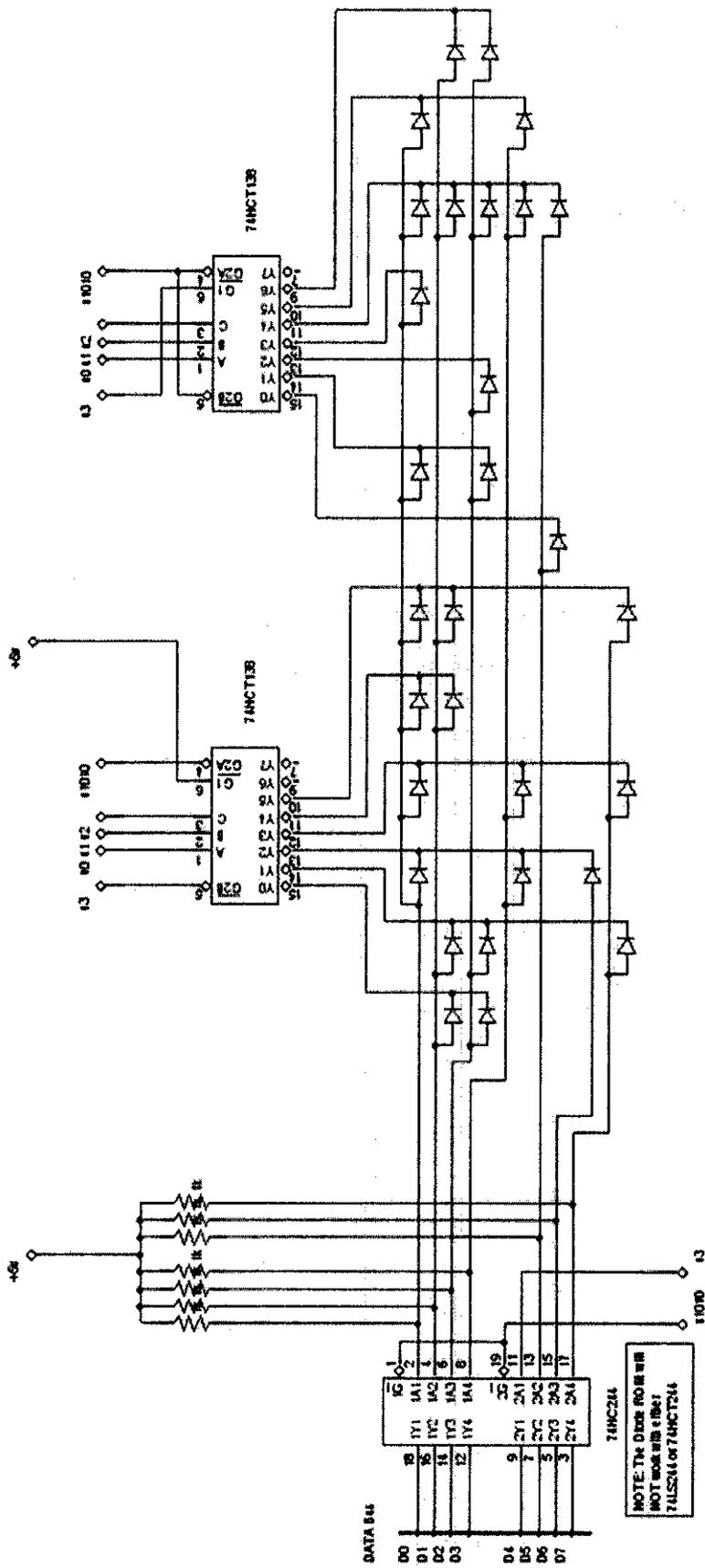
| Quantity | Part Number | Description |
|-----------------|-------------------------|-------------------------------|
| 1 | 74LS245 | Octal Transceiver |
| 1 | HEF4060BP | 14-Stage Binary Counter |
| 1 | 74LS368 | Hex Buffer |
| 1 | 82C51A | USART |
| 1 | MAX232 | RS-232 Driver |
| 2 | 74HCT32 | Quad OR Gates |
| 1 | 74HCT00 | Quad NAND Gates |
| 1 | 74HCT04 | Hex Inverters |
| 1 | DE9 Female Socket | DE9 Female Connector (Socket) |
| 1 | 330 k Ω Resistor | Resistor |
| 1 | 2.2 k Ω Resistor | Resistor |
| 1 | 2.4576 MHz XTAL | Crystal Oscillator |
| 1 | 100 pF Capacitor | Ceramic Capacitor |
| 1 | 10 pF Capacitor | Ceramic Capacitor |
| 5 | 1 μ F Capacitor | Electrolytic Capacitor |

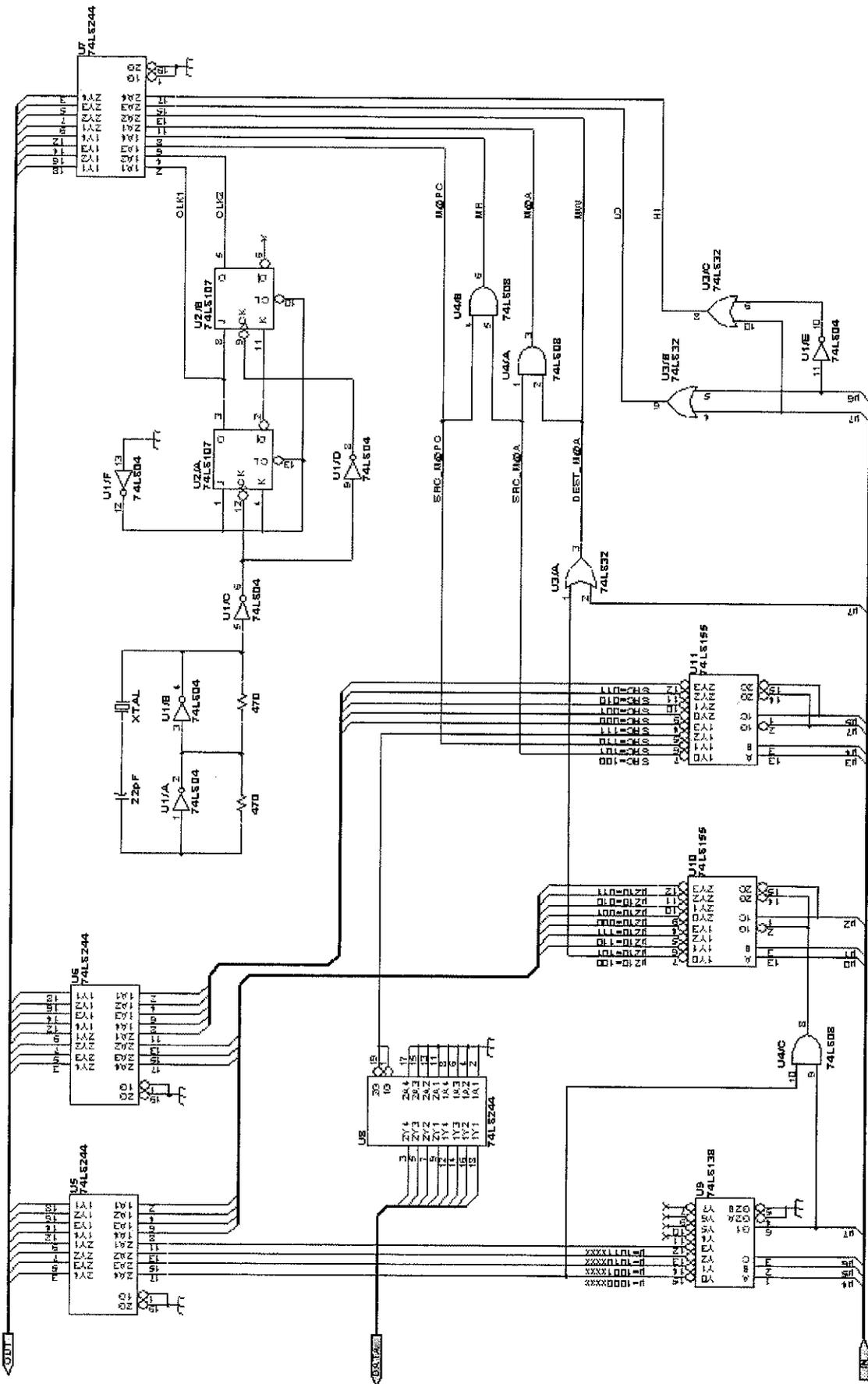
APPENDIX B: CIRCUIT DIAGRAMS

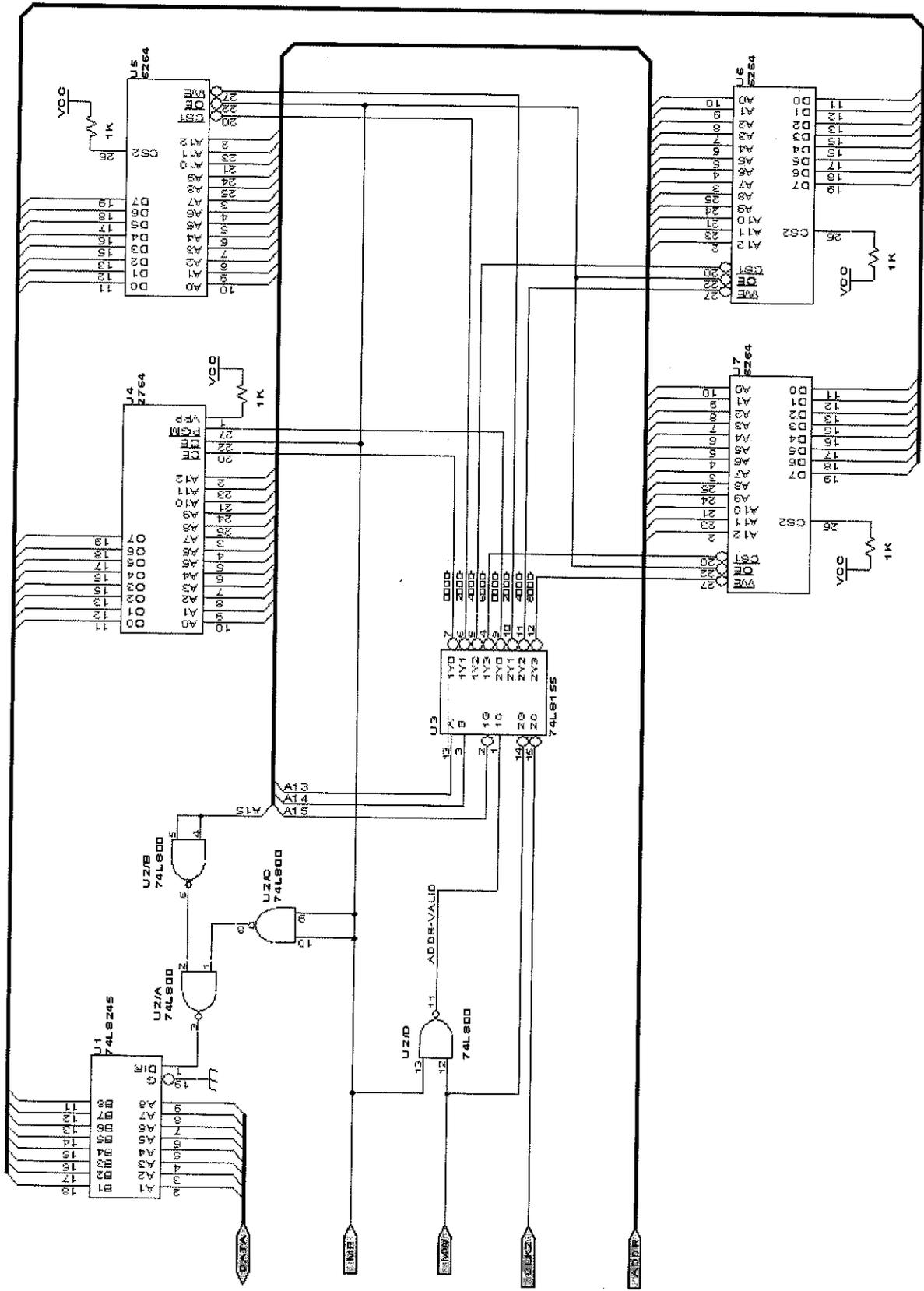
(source: A. Holme, Mark 1 FORTH Computer, Internet Web Page:
<http://www.holmea.demon.co.uk/Mk1/Architecture.htm>, 2003)

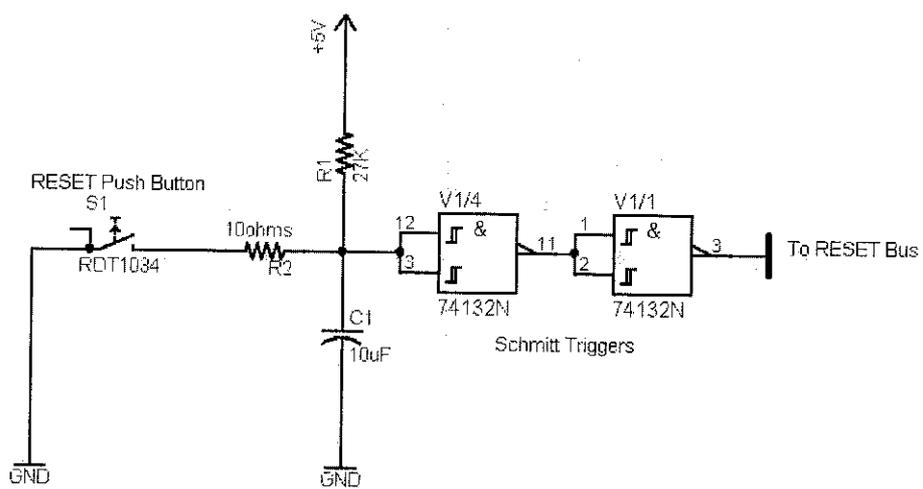
| | |
|-----------|--|
| Page vii | : ALU |
| Page viii | : Diode ROM |
| Page ix | : System Clock and Instruction Decoder |
| Page x | : Stacks |
| Page xi | : Memory |
| Page xii | : Power On Reset Circuit |
| Page xiii | : Microcode Sequencer |
| Page xiv | : Index Registers |
| Page xv | : Input/Output |

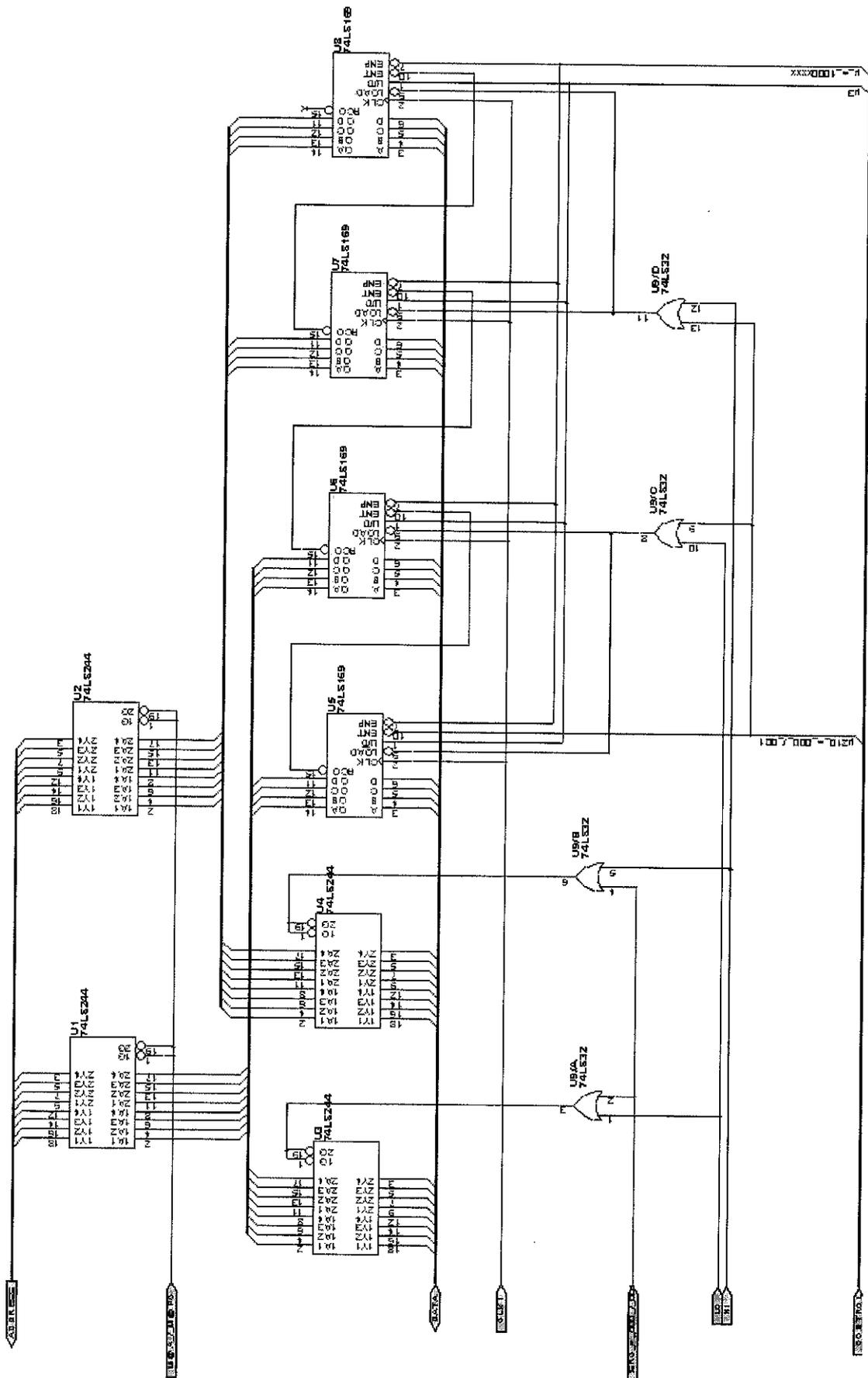












APPENDIX C: BACKPLANE SPECIFICATIONS

| BUS A | | | BUS C | | | | |
|-------|-------|--------------|----------------|-----------------|-------------------------|---------------|------------|
| 1 | +5V | | | | | | |
| 2 | CLK1 | | | | | | |
| 3 | D0 | DATA | 3 | μ 0 | μ | | |
| 4 | D1 | | 4 | μ 1 | | | |
| 5 | D2 | | 5 | μ 2 | | | |
| 6 | D3 | | 6 | μ 3 | | | |
| 7 | D4 | | 7 | μ 4 | | | |
| 8 | D5 | | 8 | μ 5 | | | |
| 9 | D6 | | 9 | μ 6 | | | |
| 10 | D7 | | 10 | μ 7 | | | |
| 11 | A0 | | ADDRESS | 11 | | μ 210=101 | Dest=OP |
| 12 | A1 | | | 12 | | μ 210=110 | Dest=ALU A |
| 13 | A2 | 13 | | μ 210=111 | Dest=ALU B | | |
| 14 | A3 | 14 | | μ 210=000 | W | | |
| 15 | A4 | 15 | | μ 210=001 | IP | | |
| 16 | A5 | 16 | | μ 210=010 | Dest=TOS | PSP | |
| 17 | A6 | 17 | | μ 210=011 | Dest=R | RSP | |
| 18 | A7 | 18 | | SRC=111 | ALU | | |
| 19 | A8 | 19 | | SRC=000 | W | | |
| 20 | A9 | 20 | | SRC=001 | IP | | |
| 21 | A10 | 21 | | SRC=010 | TOS | | |
| 22 | A11 | 22 | | SRC=011 | R | | |
| 23 | A12 | 23 | | μ =1000xxxx | INC / DEC | | |
| 24 | A13 | 24 | | μ =1001xxxx | JUMP # | | |
| 25 | A14 | 25 | | μ =1010xxxx | ALU Function | | |
| 26 | A15 | 26 | | μ =1011xxxx | JUMP OP | | |
| 27 | MR | Memory Read | 27 | M@IP | Address = IP | | |
| 28 | MW | Memory Write | 28 | M@W | Address = W | | |
| 29 | RESET | | 29 | LO | LO-byte | | |
| 30 | IRQ | | 30 | HI | HI-byte | | |
| 31 | CLK2 | | | | | | |
| 32 | 0V | | | | | | |