

To the Graduate Council:

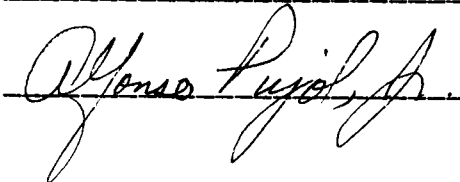
I am submitting herewith a thesis written by Larry D. Wilhite entitled "The Design and Implementation of a Programmable Auxiliary Keyboard for a Microcomputer System." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.



Ronald A. Belz, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:



The Graduate School

THE DESIGN AND IMPLEMENTATION OF A
PROGRAMMABLE AUXILIARY KEYBOARD
FOR A MICROCOMPUTER SYSTEM

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Larry D. Wilhite
March 1985

ACKNOWLEDGMENTS

The author wishes to express his appreciation to his advisor, Dr. Ronald Belz and the other members of his committee, Dr. Alfonso Pujol and Dr. Bruce Bomar for their support and assistance.

Special thanks are due my friend Mr. Sid Adams, for his help in preparing acceptable figures to include in this work. Also, to my wife, Teresa, for her patience, encouragement and assistance.

ABSTRACT

An auxiliary keyboard with 64 special function keys was designed, built and interfaced to a microcomputer system. In this case, the auxiliary keyboard was designed for an Apple //e microcomputer system.

The auxiliary keyboard, as designed, is separate from the Apple system keyboard and includes seventeen keyswitches. Each of sixteen keyswitches can be programmed with four separate key sequence programs. Selection of one of the four programs is done by the seventeenth keyswitch and indicated by LEDs on the auxiliary keyboard. Each key sequence program can contain up to 32 character codes which are sent to the computer when a key is pressed. All 128 ASCII character codes can be programmed into the auxiliary keyboard except the null character. The key sequence programs are stored in a 2K EEPROM which will retain the programs after the system power is turned off.

Interfacing to the Apple //e required that the standard Apple firmware input routines be modified. Additionally, programs were written to output characters to the screen, convert a binary format number to a binary-coded-decimal format number, take single character

input from the Apple keyboard, and transfer data or programs from the Apple's RAM to an EEPROM. Both the 40-column mode and 80-column mode of the Apple //e were implemented in the auxiliary keyboard system software. Each routine was written in the 6502 Assembly language.

Hardware includes a keyboard interface card that plugs into the Apple's expansion slot number 5, a 2764 EPROM that replaces a ROM on the Apple motherboard, and an external keyboard. All previous functions and memory in the Apple //e were retained.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
II.	DESIGN CRITERIA.	7
	Basic Design Specifications.	7
	Operational Concept.	8
III.	DESIGN APPROACH.	10
	Apple //e Interface Considerations	11
	Auxiliary Keyboard Implementation Approach	16
IV.	HARDWARE DESIGN.	19
	Peripheral-Card Memory Spaces.	22
	Program Memory	25
	Keyboard Encoding.	30
	Key Sequence Memory.	33
	Board Select Logic	39
	Keyboard	42
	Other Subsystems	42
V.	SOFTWARE DESIGN.	45
	Interface Definitions and Requirements	46
	Apple //e Character Input Routines	47
	Other Systems Character Input Handlers	52
	Apple //e Software Implementation.	52
	System Software Patches.	55

Auxiliary Keyboard Input Routine	56
Auxiliary Keyboard Programming Software.	61
EEPROM Programming Utility	68
VI. APPLICATIONS	70
VII. RESULTS AND CONCLUSIONS.	77
As-Built System Specifications	77
Hardware Compatibility	78
Software Compatibility	79
LIST OF REFERENCES	80
APPENDIXES	82
A. System Schematic	83
B. Apple Standard Input Routines.	84
C. Patched System Input Routines.	86
D. Auxiliary Keyboard Input Routines.	88
E. EEPROM Utility Program	98
F. Operational Instructions	100
VITA	105

LIST OF TABLES

TABLE	PAGE
1. Peripheral Card I/O Space Usage	48
2. Escape Codes.	54
3. Control Codes	65
4. Software Compatibility.	71
5. Apple DOS System Commands	72
6. CP/M System Commands.	73
7. Basic Language Commands	74
8. dBase II Commands	76

LIST OF FIGURES

FIGURE	PAGE
1. Keyboard Enclosure.	5
2. An Interface Consideration.	12
3. The Design Approach	14
4. Auxiliary Keyboard System	15
5. Auxiliary Keyboard Logic.	20
6. I/O Space Memory Map.	23
7. Program Memory.	26
8. Address Decode Logic.	28
9. Keyboard Encoder.	32
10. Control Logic	35
11. Key Sequence Memory	36
12. Board Select Logic.	40
13. Keyboard.	41
14. Returning to CX ROM Method 1.	59
15. Returning to CX ROM Method 2.	60
A-1. System Schematic	83

CHAPTER I

INTRODUCTION

The keyboard is one of the most used peripherals of a microcomputer system. Computers utilize the keyboard as the primary human interface for entering and editing programs and data. For example, a user of word processing software enters text through the keyboard. Commands issued to the microcomputer operating system or other user software are inherently entered through the keyboard. Systems such as the McIntosh microcomputer manufactured by Apple Computer, Inc., use a mouse as an input device, but only to augment the keyboard, not replace it. The Apple //e (i.e., Apple Model 2E) microcomputer, also manufactured by Apple Computer Inc., has a 63-key keyboard and is the input device for programs, system commands and data [1]. This keyboard, like the keyboards of other microcomputer systems, generates a predefined code when one key or a combination of two or more keys are pressed. There are a set of 128 different codes called the ASCII (American Standard Code for Information Interchange) character set that can be generated from these 63 keys. How the computer responds to these characters, especially the control

characters, depends on the program that is accepting input at that time.

Other keyboards, such as those on many intelligent CRT terminals, have the same keys as the Apple //e plus some others. For example, on the VT100, manufactured by Digital Equipment Corp., the keyboard has a separate numeric keypad that enables numbers to be entered in calculator fashion [2]. Each key of the keypad generates the same character code as the corresponding numeric key on the keyboard. When a microcomputer system is used with spreadsheet programs such as Visicalc or Multiplan, the keypad is very useful for numeric input. Also, there are four keys, PF1-PF4, that cause the VT100 to transmit special predefined codes. The program accepting input recognizes the codes produced by these keys and performs some predefined programmed sequence.

The Visual 200 CRT, manufactured by Visual Technology Inc., has a numeric keypad and 13 programmable special function keys, F0-F12 [3]. Each of these keys can be programmed to generate a user-defined sequence of character codes when pressed. This sequence can contain displayable codes, control codes or escape sequences and can be a maximum of 48 characters long. These keys can be programmed to generate the codes to insert a character, delete a line, clear the screen, etc. Also, system

commands can be programmed and sent to the active program when one of these keys is pressed. Once a key is programmed, it will remain programmed until the terminal power is switched off.

The Apple //e keyboard does not have special function keys nor a separate numeric keypad. The 63 keys that make up the Apple keyboard are used to generate only one character code at a time. The code generated depends on the key or the combination of keys being pressed.

An auxiliary keyboard of the type described would be useful for many microcomputer systems. Many programs such as dBase II and Wordstar require several codes to control program operation. In dBase II there are 18 control codes used for full screen edit, modify and append functions [4]. Control codes like Cntrl-X, are used to move the cursor down to the next field, and Cntrl-P toggles the printer ON or OFF. With programmable special function keys, each key could be programmed to generate one of these obscure control codes. Labels could be made for these keys to read "DOWN" and "PRINT", thus making the dBase II program easier to learn and use. The same could be done for word processor commands and operating system commands. Labels for a group of keys could be made into an overlay that could be easily changed to correspond to the requirements of the program being run. Also, these special function

keys could be arranged in a matrix, programmed to generate numeric codes and function as a numeric keypad for numeric input.

This work was directed toward implementation of an auxiliary keyboard for an Apple //e microcomputer system (see Figure 1). A set of design specifications were compiled and are given in Chapter II. Two interface methods between the auxiliary keyboard system and the Apple //e were considered and this analysis is presented in Chapter III, the Design Approach. The auxiliary keyboard system is broken down into five hardware subsystems and discussed in detail in Chapter IV, Hardware Design. The approach taken includes making software patches to the Apple //e firmware input routines which are located in the CX ROM on the Apple //e motherboard. This software and the programming software to input the key sequences into the auxiliary keyboard are discussed in Chapter V, Software Design. In Chapter VI, Applications, compatibility of the auxiliary keyboard with several user programs, operating systems and languages are tabulated. Also in Chapter VI, a typical setup of the auxiliary keyboard is given. In Chapter VII, Results And Conclusions, the as-built specifications are compared to the design specifications of

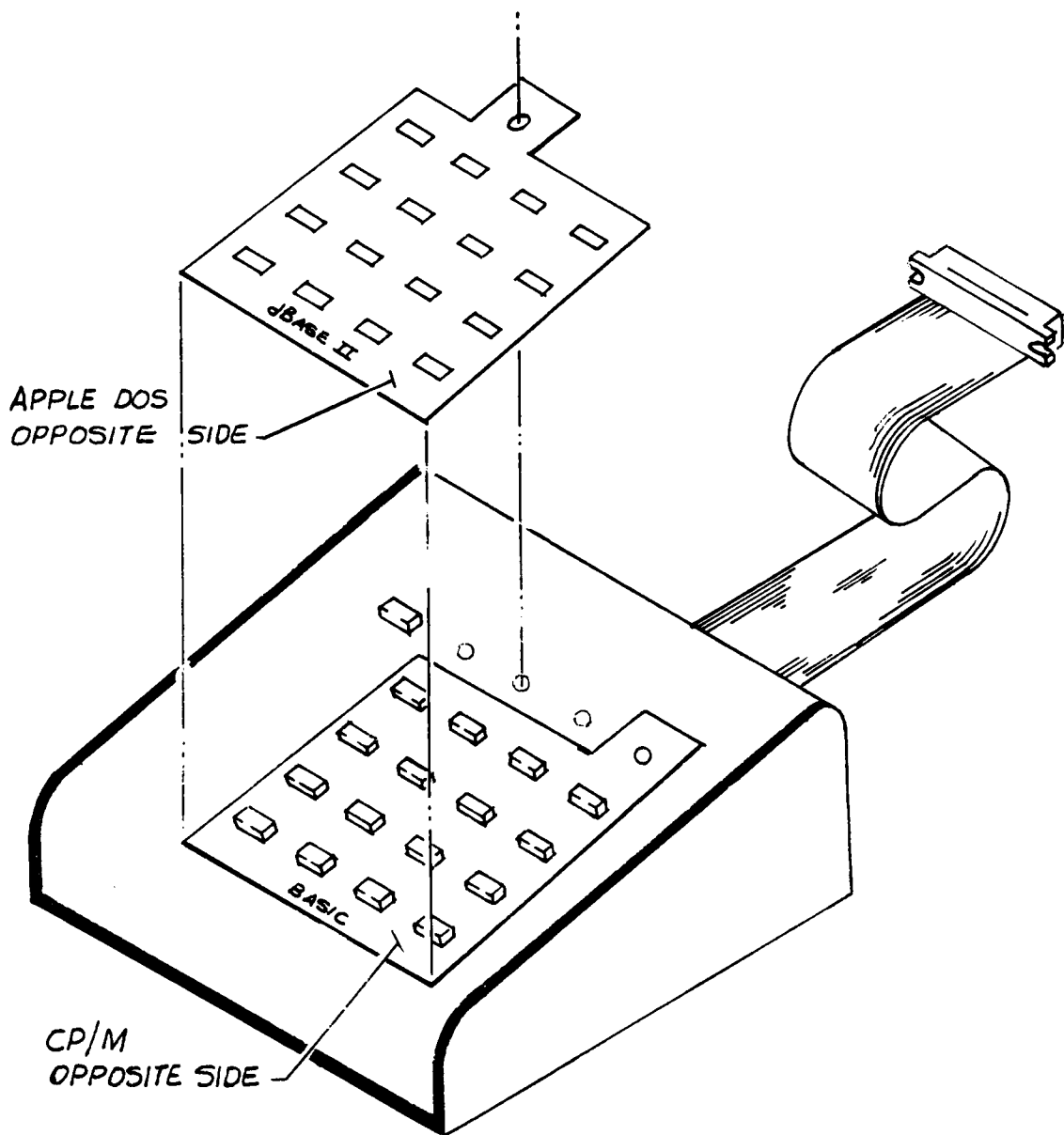


FIGURE 1. Keyboard Enclosure

Chapter II. Finally, the compatibility of the auxiliary keyboard, as implemented, is compared to other microcomputer systems in Chapter VII.

CHAPTER II

DESIGN CRITERIA

Basic Design Specifications

A set of basic design specifications or criteria was compiled before the design was begun. These specifications were used to select hardware and governed the design of the software interfaces to the user. The specifications as developed for the design of the auxiliary keyboard system are:

1. Provide 16 programmable special function keys.
2. Program special function keys from the Apple //e keyboard.
3. Retain programming of special function keys after the system power is turned off.
4. Generate and retain at least a 10-character code sequence for each special function key.
5. Provide a simple method to identify the function of each programmable special function key.

6. Prevent reduction, degradation, or inhibition of any present function of the Apple //e system.
7. Provide the above specifications in the simplest and least expensive fashion possible.
8. Use parts readily available.
9. Allow any code generated by the Apple //e keyboard to be programmed into the special function keys.

These specifications were considered to be the minimum requirements of the auxiliary keyboard. As the design progressed, enhancements to these basic requirements were identified and are described in Chapter VIII.

Operational Concept

In addition to the basic design specifications the following operational features were to be included in the auxiliary keyboard system. The auxiliary keyboard was to be separate from the main keyboard. This would allow the user to position the auxiliary keyboard in a position best

suitable for him to press the keys. The auxiliary keyboard was to become active when the Apple //e power was applied. Any required initialization was to be transparent to the user.

Programming of the keys was to be straight forward and simple. The programming software would lead the user through the programming procedure. A way to correct entry mistakes was to be included. One of the keys on the auxiliary keyboard would be capable of initiating the programming procedure.

CHAPTER III

DESIGN APPROACH

To select the best interface method for the Apple //e, an understanding of how the system keyboard is interfaced and a functional understanding of the expansion slots and the firmware input routines was required. Two different interface methods were developed and analyzed. The better method was selected based on the design criteria of Chapter II and availability of hardware.

Keyboard input is received at a very slow speed relative to the microprocessor instruction cycle. Even with the fastest typist at the keyboard there is time between each key typed for the microcomputer to perform many instructions. For example, if a typist could type 100 words per minute, where each word has an average of 5 keystrokes, typing speed is 500 keystrokes per minute or 8.33 keystrokes per second. The 6502 CPU (Central Processing Unit) of the Apple //e has a clock cycle time of 0.978 microseconds [5]. If we consider each instruction as requiring about four clock cycles, then the Apple //e can process 255,750 instructions per second. Thus, the Apple //e can process 30,702 instructions per keystroke.

Consequently, many of the required logic functions of the auxiliary keyboard were implemented in 6502 software. This is in conformance with the design criteria for simplicity and flexibility.

Apple //e Interface Considerations

The AY5-3600 PRO keyboard encoder chip, manufactured by General Instrument, is used in the Apple //e as the keyboard interface. The Apple keyboard is a matrix of keyswitches connected to the AY5-3600 keyboard encoder by way of a ribbon cable and a 26-pin connector. Outputs from the AY5-3600 include 7 bits of key code plus separate lines for shift, control, any-key-down and keyboard strobe. The key-code lines, along with control and shift, are address inputs to a separate 2316 ROM. The ROM translates these inputs to the character codes that are enabled onto the data bus by signals named KBD' and ENKBD' [1].

One method that was considered for interfacing the auxiliary keyboard to the Apple //e is shown in Figure 2. The AY5-3600 PRO would be removed from the Apple //e motherboard and installed on an auxiliary keyboard circuit card. A cable with a connector to plug into the vacated AY5-3600 socket would connect to the circuit card. On the circuit card more logic and intelligence would be located to intercept and record the key sequences, encode an

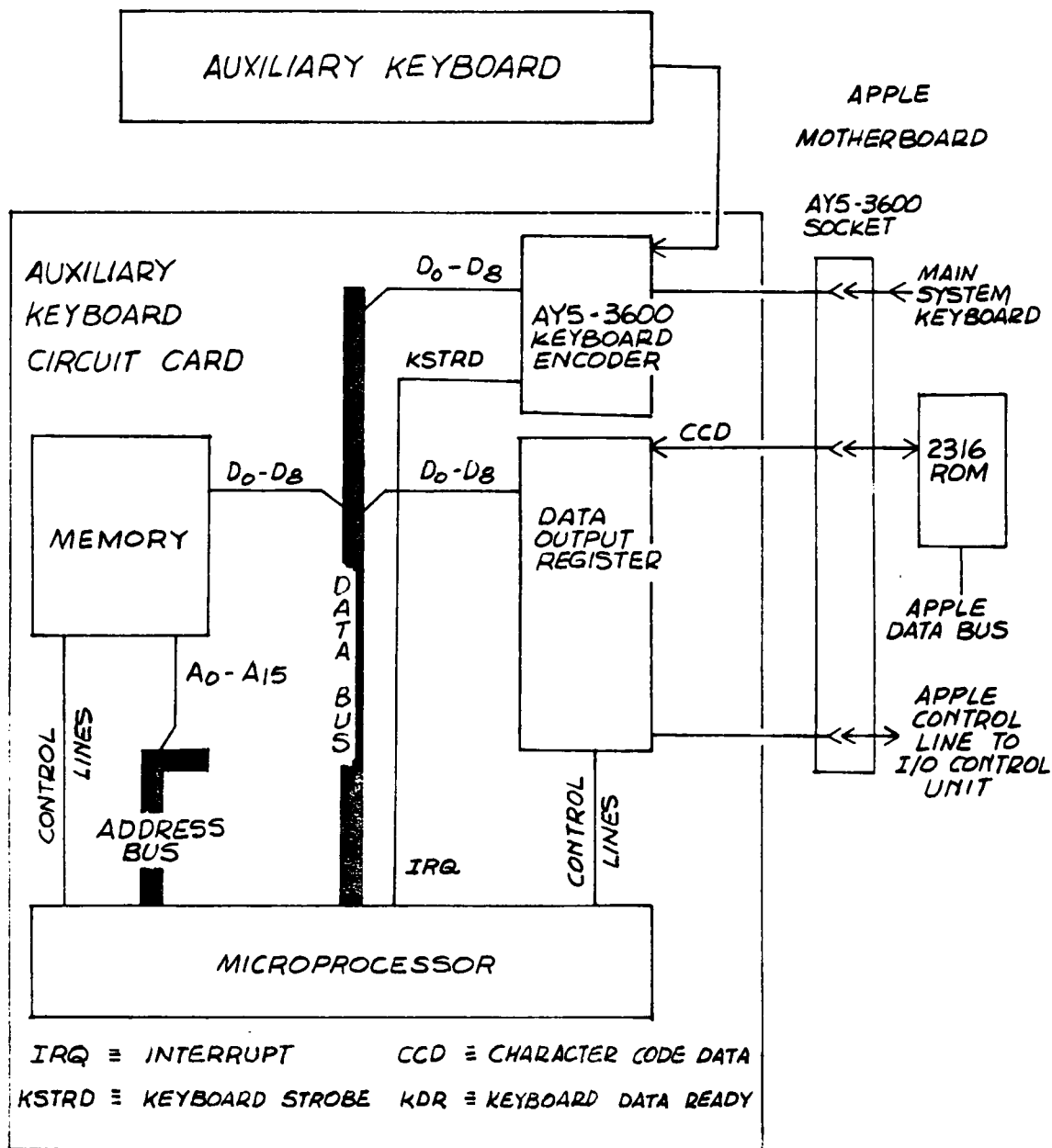


FIGURE 2. An Interface Consideration

auxiliary set of keys and generate the recorded key sequences at the proper time for the 2316 ROM on the motherboard.

The second method considered is shown in Figure 3. The auxiliary keyboard system would include keyboard encoding circuitry to decode a set of keyswitches, key sequence memory to store the programmed key sequences, program memory to store the system programming and other required interface hardware and control logic. In this method the Apple //e's firmware input routines located in the CX ROM on the Apple //e motherboard (see Figure 4) would be modified to support the auxiliary circuitry. When the Apple //e's input routines were called, the modification would cause a jump to a program memory location containing a new input routine. The new input routine would detect when an auxiliary key was pressed and output the associated, previously recorded key code sequences.

This second method is more software oriented than the first. Because this second method was considered to be the simplest and most flexible method it was chosen and implemented. More detailed descriptions are given for each part of the auxiliary keyboard system in the hardware and software design chapters.

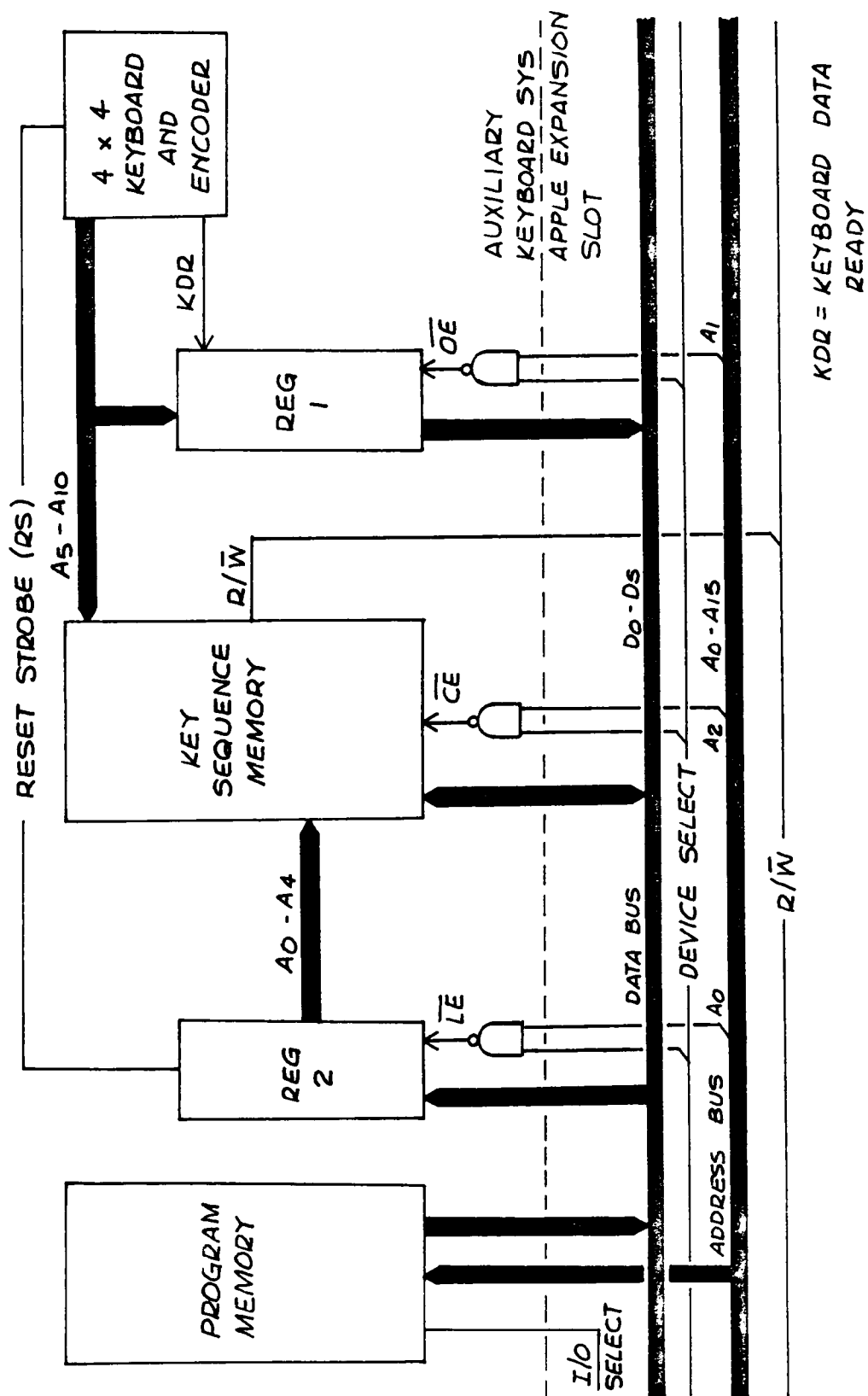


FIGURE 3. The Design Approach

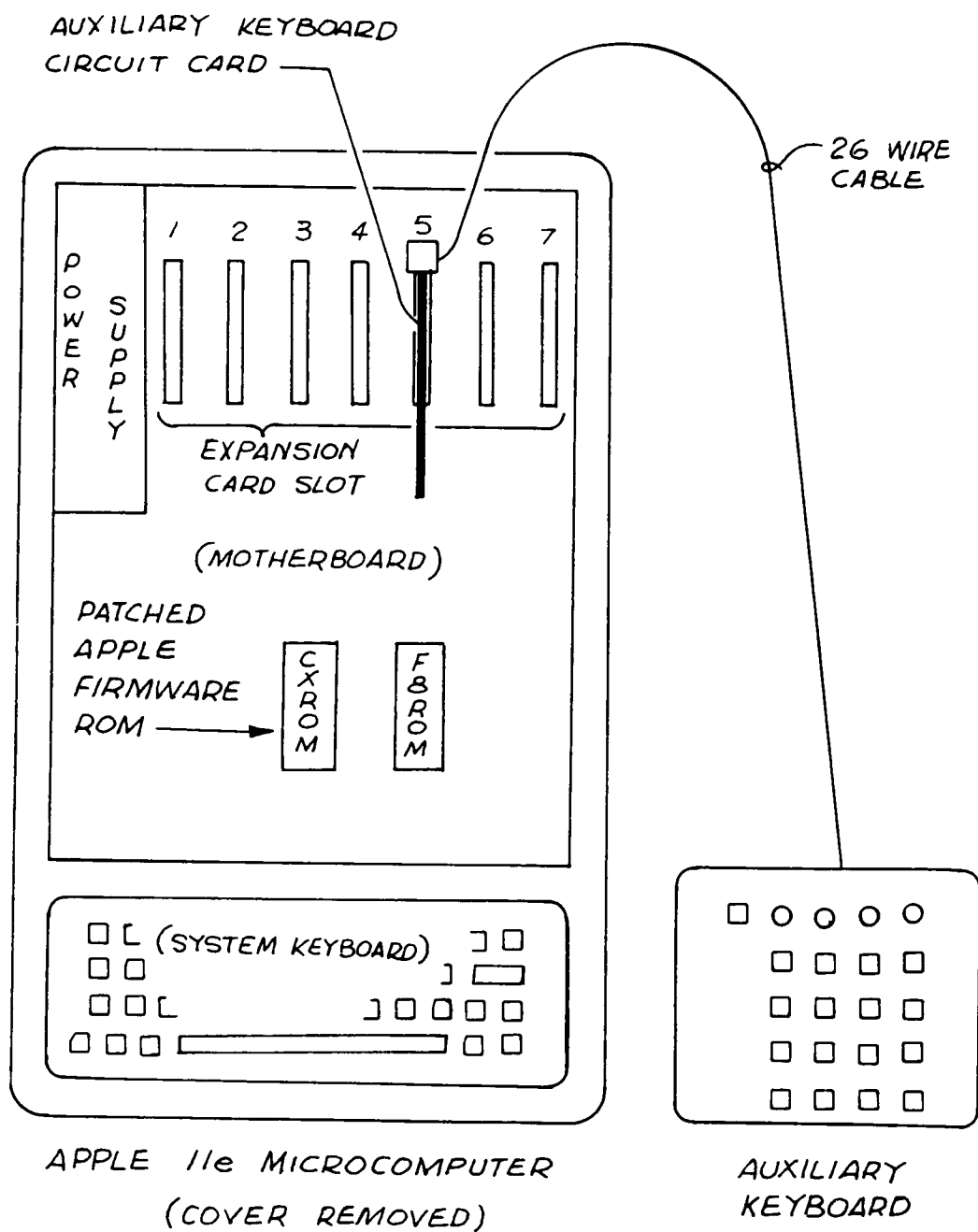


FIGURE 4. Auxiliary Keyboard System

Auxiliary Keyboard Implementation Approach

When a program calls for character input the standard Apple //e input routine is called to provide the input character. Instead of processing characters only from the main Apple keyboard the standard input routine has been modified to process characters from the main Apple keyboard and the auxiliary keyboard. The extra programming code to implement this process is located in the program memory subsystem of the auxiliary keyboard system (see Figure 3). The following paragraphs provide a general description on how this process is implemented for the auxiliary keyboard system on the Apple //e microcomputer.

The modified input routine scans the main Apple keyboard and the auxiliary keyboard until a key is pressed. A keypress is detected on the auxiliary keyboard when the Keyboard Data Ready (KDR) has been set by the auxiliary keyboard encoding logic and is read by the input routine through Reg1.

When KDR has been set, address lines A5-A10 (to the Key Sequence Memory) are also stabilized. Figure 3 schematically shows this interconnection between subsystems. Four bits of these address lines encode the 16 keys on the auxiliary keyboard. The remaining two bits encode the four 16 key sets. All six bits make up part of the address lines to the key sequence memory subsystem.

Once KDR is set and A5-A10 are stabilized the input routine zeroes a character counter, which is in the Apple //e memory. The data from the character counter is written to Reg2 which latches the data on its output lines. Five of these output lines are input to the key sequence memory as address lines A0-A4 (again refer to Figure 3). Bits A5-A10 address a section of key sequence memory containing the key code sequence associated with one of the 64 possible auxiliary keyboard inputs. Also, bits A0-A4 address each character code of the possible 32 character codes of the selected key sequence.

The next steps performed by the modified input routine are to read a character code from the key sequence memory and return it to the program requesting character input. When the next character is requested by a calling program, the character counter (described above) is incremented by one and its contents are written to Reg2, thereby addressing the next character code in the selected key sequence.

This procedure is repeated as often as the calling program requests input until either a delimiter character is detected by the input routine or 32 characters have been output. The delimiter character indicates the end of the code sequence. It should be noted, that if a carriage return is one of the output characters that is generated

before the delimiter, then subsequent inputs to the calling program will be provided by the remaining characters in the key code sequence. Therefore, several input statements can be supported through one keypress.

When the delimiter is detected or 32 character codes are output by the modified input routine, a Reset Strobe (RS) is generated to reset KDR. The RS signal is generated by the auxiliary keyboard system hardware when the modified input routine writes to memory location \$C0D8.

Another program was written in addition to the modified input routine to store the key sequence memory with the character codes. This program is also located in the program memory subsystem. It is called as a subroutine and it leads the user through the steps required to store the character codes in the key sequence memory. Any character code generated by the main Apple keyboard can be stored in the key sequence memory except the null (\$00) character. The null character is utilized by the modified input routine as the delimiter character in the key sequence.

CHAPTER IV

HARDWARE DESIGN

The hardware design can be divided into five subsystems (see Figure 5). First, the stand along Program Memory subsystem that contains the machine language code that operates the auxiliary keyboard (labeled (1) in the figure); second, the Keyboard Encoding subsystem that encodes a set of 16 keys to a 4-bit code (labeled as (2)); third, the Key Sequence Memory subsystem that retains the programmed key sequences (labeled as (3)); fourth, the Board Select Logic that selects one of four 16-key keyboards (labeled as (4)); and fifth, the Keyboard that consists of the 17 keyswitches, interconnect and housing (labeled as (5)). Before these subsystems are described the definitions of the peripheral-card memory spaces available in the Apple //e are described. The selected hardware devices and a functional description of each subsystem is presented after the peripheral-card memory space descriptions. A complete system schematic is included in Appendix A in addition to the subsystem figures referenced in the text.

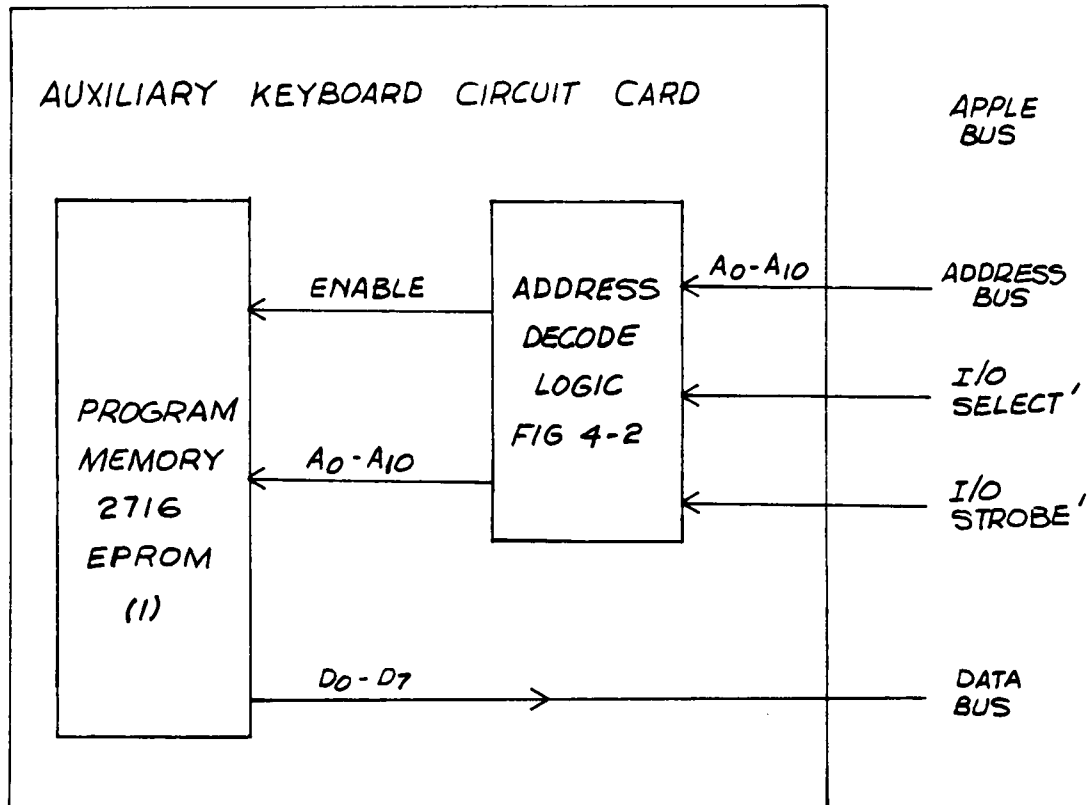
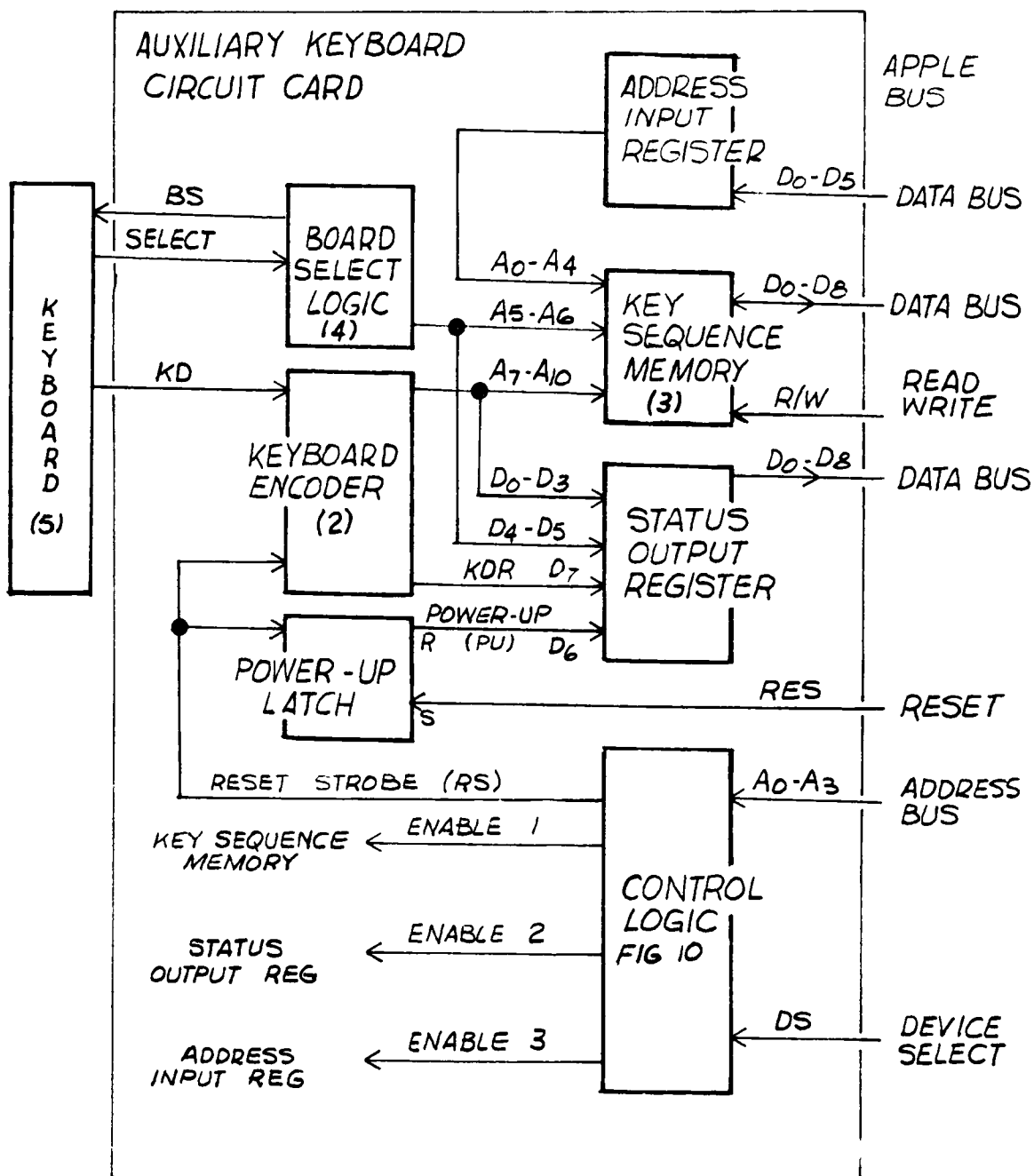


FIGURE 5. Auxiliary Keyboard Logic



KDR $\equiv$ KEYBOARD DATA READY
 BS $\equiv$ BOARD SELECTED
 KD $\equiv$ KEY DATA

FIGURE 5 (cont.)

Peripheral-Card Memory Spaces

The seven expansion slots on the Apple //e's main circuit board are used for circuit cards containing the hardware and firmware needed to interface peripheral devices to the Apple //e. These slots are not simple I/O ports; peripheral cards can access the Apple //e's data, address, and control lines via these slots.

Because the Apple //e's 6502 microprocessor does all of its own I/O through memory locations, portions of the Apple //e's memory space have been allocated for the exclusive use of the cards in the expansion slots (see Figure 6). In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or PROM) on the peripheral cards themselves.

One of these address spaces, peripheral-card I/O space, is described in the key sequence memory subsystem section. The locations in this space were used by the auxiliary keyboard hardware to interface to the auxiliary keyboard software.

Another memory space called peripheral-card ROM space is one 256-byte page of memory space that is allocated to each peripheral card. This space is normally used for read-only memory (ROM or PROM) on the card with

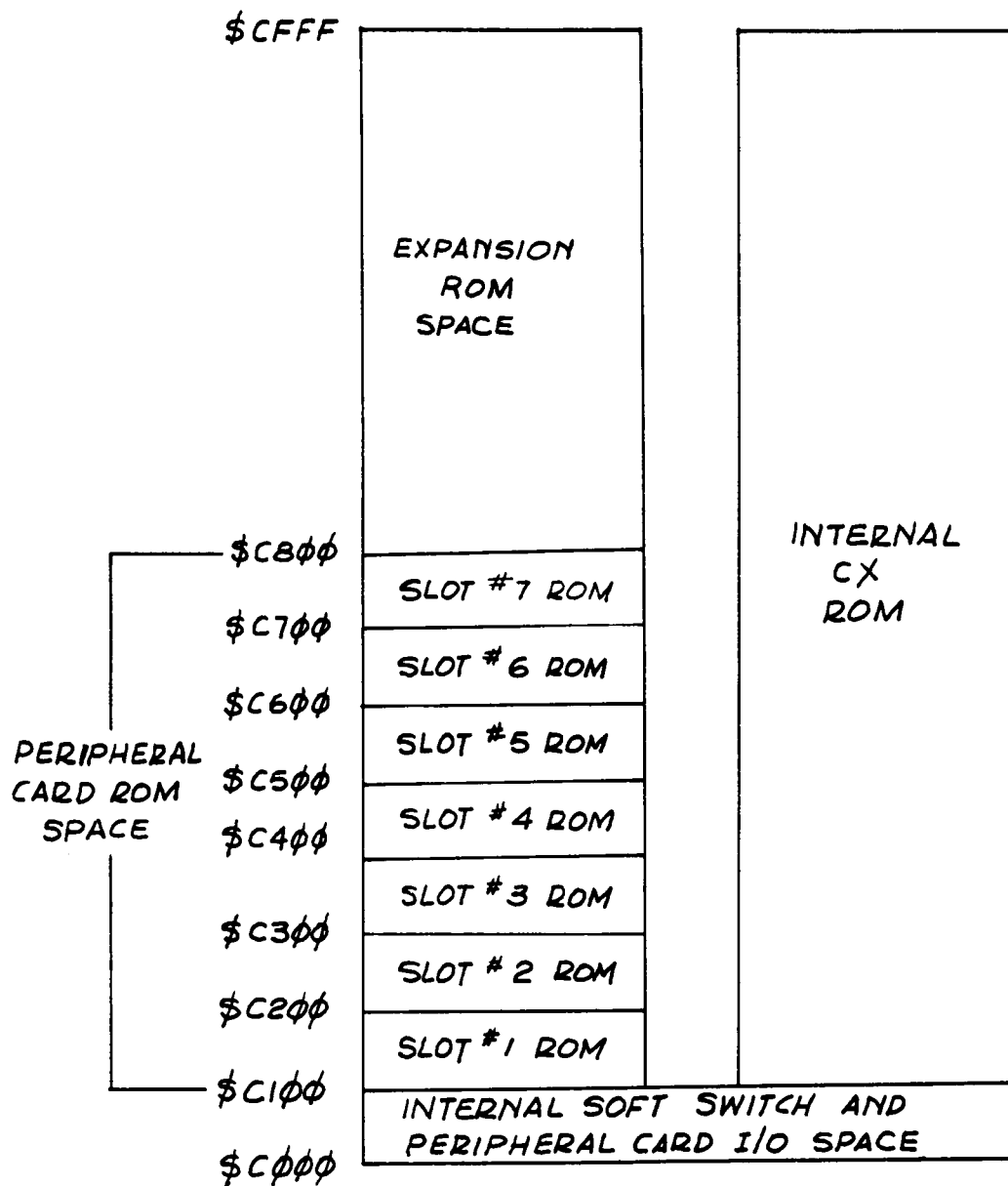


FIGURE 6. I/O Space Memory Map

driver programs that control the operation of the peripheral device connected to the card. This space was utilized to contain the auxiliary keyboard input routine, HANDLER-A. Further description of the HANDLER-A routine is given in Chapter V.

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K-byte memory space from \$C800 to \$CFFF for larger programs in ROM or PROM. This memory space is called expansion ROM space. The auxiliary keyboard programming software, HANDLER-B, was located in the expansion ROM space. Besides being larger, the expansion ROM memory space is always at the same locations regardless of which slot is occupied by the card. This memory space is available to any peripheral card that needs it. More than one peripheral card can have expansion ROM on it, but only one of them can be active at a time. To do this, the peripheral card must become inactive whenever it detects the presence of \$CFFF on the address bus. On the auxiliary keyboard circuit card the I/O STROBE' signal was used for part of the address decoding, since it is active for addresses from \$C800 through \$CFFF.

Peripheral-card RAM space is 56 bytes of main memory allocated to the peripheral cards, eight bytes per card. These 56 locations are actually in the RAM reserved for the

text and low-resolution graphics displays, but these particular locations are not displayed on the screen and their contents are not changed by the built-in output routine. Five RAM locations were required to implement the auxiliary keyboard system software. Because peripheral-card RAM was available for use by the auxiliary keyboard system software, circuitry to provide RAM for the software was not included in the hardware design.

Program Memory

The memory used to retain the auxiliary keyboard programming is a 2716 EPROM, reference Figure 7. The 2716 EPROM is a 2k by 8 bit memory chip that requires only a +5 volt power supply. It is widely available and its operation is well defined. Other factors which lead to the selection of this EPROM were the amount of program memory required and the available memory locations within the Apple //e's memory space. From a study of the Apple //e Reference Manual, it was found that expansion ROM space plus peripheral-card ROM space provides $256 + 2048 = 2304$ bytes for program storage. At the beginning of this work there was no sure way to determine how large the system programs would be. Monitor program listings of the Apple //e's input handlers, an 80-column mode and a 40-column mode input routine, were analyzed and both routines used

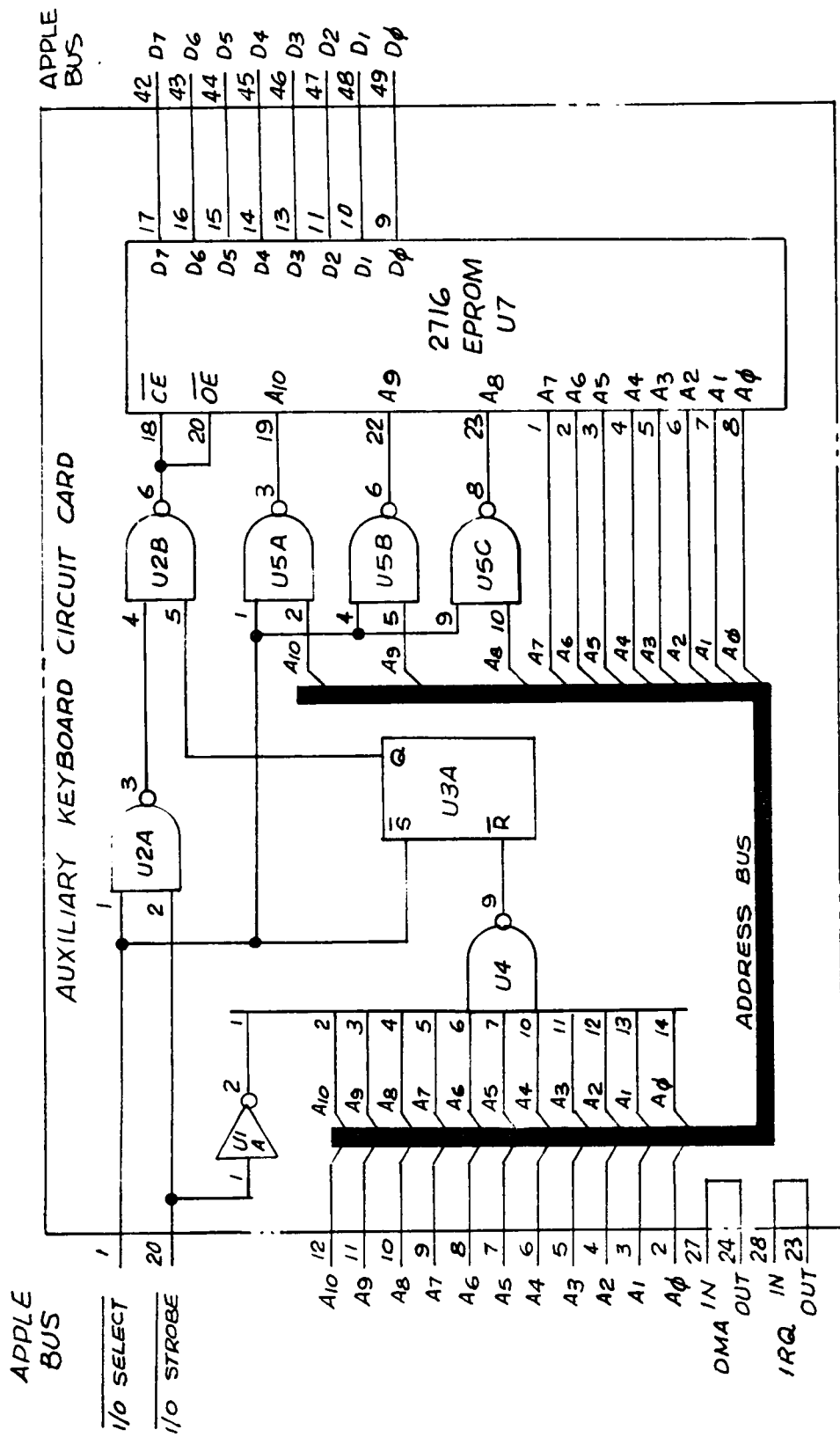


FIGURE 7. Program Memory

less than one page (256 bytes) of memory. It was therefore decided to employ a single 2k EPROM, 8 pages of memory, and design an addressing scheme to overlay the peripheral-card ROM space with the lowest 256 bytes of the expansion-card ROM space. The 2k space proved to be sufficient for all the system programs.

A unique addressing scheme is used to address the 2716 EPROM, reference Figure 8, in which the lowest page (256 bytes) of the EPROM is addressed as peripheral-card ROM space, enabled by the I/O SELECT' control line. The upper seven pages are addressed as expansion-card ROM space, enabled by the I/O SELECT' and I/O STROBE' control lines. This allows memory within the 2716 EPROM to be located in one of two separate address spaces. When the Apple 6502 CPU is accessing memory in the address range from \$C500 to \$C5FF, the I/O SELECT' of slot 5 is active and the I/O STROBE' is inactive. Through AND gates U5A, U5B and U5C, the I/O SELECT' also forces address lines A8 - A10 to logic low since only the lower 256 bytes are to be accessed when I/O SELECT' is active. Because the 2716 EPROM is a read-only device the R/W' control line is not required for memory access. A low signal on the chip enable inputs will cause the 2716 EPROM to output the addressed data byte on its data lines Q0 - Q7. The timing of I/O SELECT' or I/O STROBE' is such that the Apple's

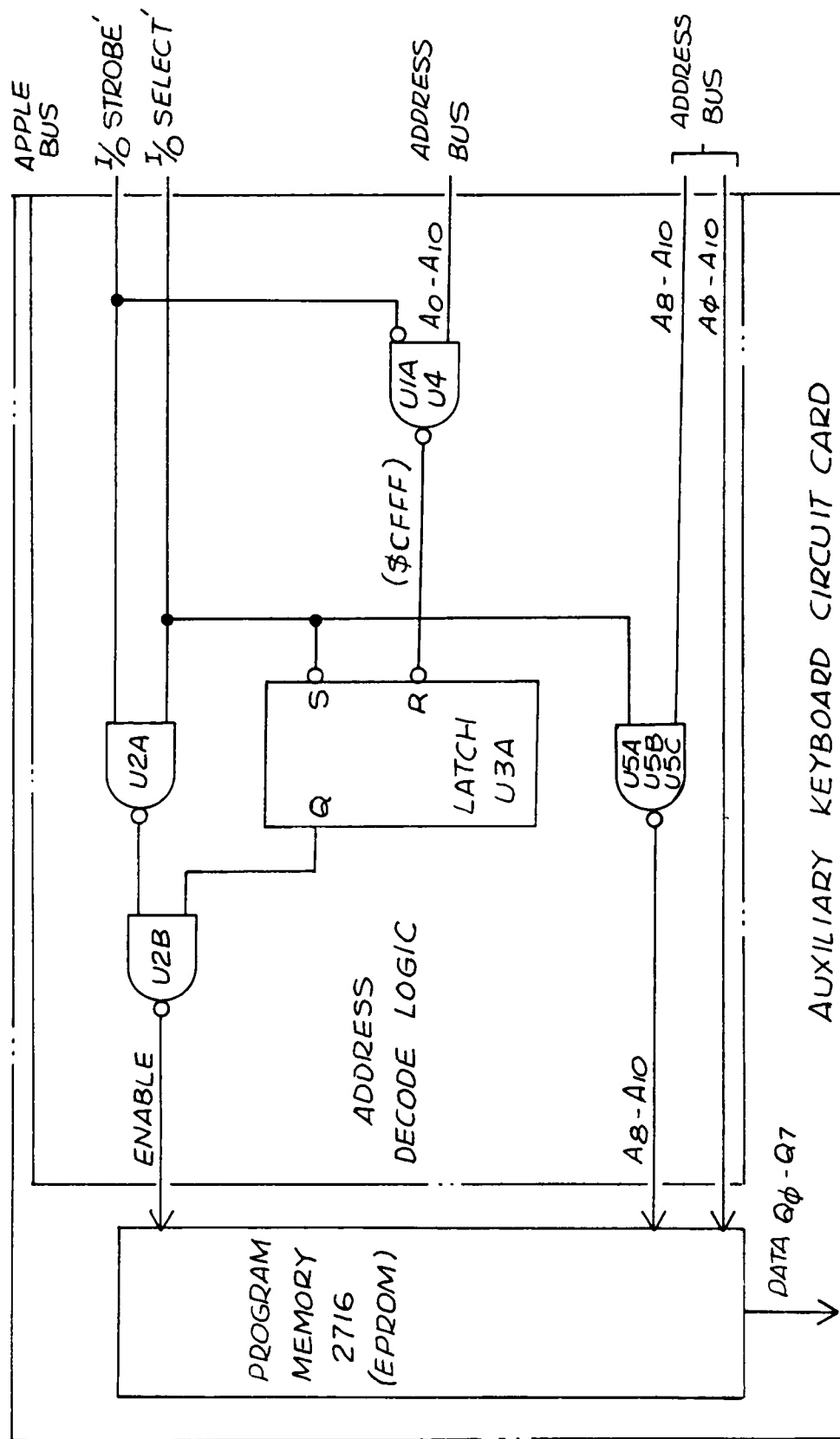


FIGURE 8. Address Decode Logic

6502 CPU is ready to accept data from the data lines when either I/O SELECT' or I/O STROBE' is active.

Because expansion-card ROM space, \$C800 through \$CFFE, is used by all six expansion slots, a two-step sequence is utilized to enable one expansion memory device at a time. When address \$CFFF is accessed by the CPU the 2716 EPROM is disabled. The same action, to disable any expansion-card ROM, is required of each card in each expansion slot of the Apple //e. To disable any active expansion-card ROM is the first step in enabling a particular expansion-card ROM on a particular peripheral card. After the access to the \$CFFF address is made the next access to slot 5 peripheral-card ROM space will enable the expansion-card ROM device in slot 5. This two-step process has set up the EPROM so that any further access to the expansion-card ROM space, I/O STROBE' active, will enable the 2716 EPROM for reading. During this time, address lines A0 - A10 are also enabled allowing access to the entire 2k address space of the 2716 EPROM. It should be noted that the lowest page of the 2716 EPROM is also accessed as peripheral-card ROM space. Use of the 2716 EPROM expansion-card ROM space can continue until another \$CFFF access by the CPU is issued.

A daisy-chain interrupt and DMA (Direct Memory Access) priority arrangement is used between each expansion slot in the Apple //e microcomputer. The continuity of this chain is continued through the auxiliary keyboard circuit card.

Keyboard Encoding

Commonly available parts such as mechanical keyswitches instead of capacitive type keyswitches or flat panel type keyswitches were utilized. Almost without exception this type of key will cause "bounce"; that is, the key makes several very short intermittent contacts as it is pressed [6]. Most digital systems are fast enough to read each bounce and record the assumed single closure event as a series of key openings and closings. To prevent entry errors, the key encoding scheme used must include the logic to mask or eliminate the bounce effect. Many methods of keyboard encoding have been devised--the best method for the auxiliary keyboard was chosen based on the design criteria outlined in Chapter II. Several special purpose chips are manufactured for keyboard encoding, such as the AY5-3600 PRO manufactured by General Instrument and used in the Apple //e, but this was considered as overkill. The auxiliary keyboard was required to have only 16 keys and

the AY5-3600 PRO will interface to 96 keys. Straight binary coding was considered but to decode 16 keys would take four 8-input gates for 16 keys (4-bit code) and this does not include the debounce logic. A simple scanning method was selected and implemented with a 74LS193 4-bit counter, a 74150 16-to-1 line multiplexer and some simple logic gates that were not utilized by the other circuits of the system, reference Figure 9. A 1-MHz clock input was available from the Apple expansion slot interface; therefore, additional logic required to generate a clock signal was not required. The debounce of the keyswitches is inherent in the design implemented and required no further circuitry. That is, the instant a key is detected a flip-flop is set. This stops the clock signal from incrementing the scanning counter. Thus, a key code is locked on the outputs of the counter. If the pressed key bounces through logic high and logic low again, the 4-bit code remains the same. This is because the flip-flop will still be set from the first transition to logic low.

When a key is pressed, the scanning counter U12 is halted and the output of the U12 counter is the key's address. That is, the four output bits from the U12 counter make up the hexadecimal key number 1 through 16. Input lines IO-I15 are maintained at +5 volts by 4.7 k Ohm pull up resistors until a key is depressed and causes

the corresponding input line to go low. The select inputs S0-S3 are connected to the scanning counter U12 and each of the 16 inputs are scanned for a low signal. When a low signal is present on the addressed input line (I0-I15) of the U13 multiplexer, output Z goes high. This is inverted by inverter U1D and sent to the set input of flip-flop U3C. Flip-flop U3C's output is set high, inverted by NAND gate U14B and input to NAND gate U14C in order to inhibit the clock input (at U14C) from incrementing scanning counter U12. This halts the scanning counter at the key's address and provides a Keyboard Data Ready signal (KDR) from the flip-flop U3C output.

Key Sequence Memory

Several memory devices involving new technologies have become available in the last few years. Among them are the electrically alterable read-only memory (EAROM), the electrically erasable programmable read-only memory (EEPROM), and the nonvolatile random access memory (NOVRAM), all of which can replace conventional memory chips in certain applications [7]. They have the attributes of the ideal memory: they are fast, random access; they do not lose information when power is lost; and they can be reprogrammed without removing them from their sockets in the computer. Since they are currently

several times more expensive than conventional memory chips, the use of these new memory devices is not widespread in current circuit designs. Of these new technology chips, the Xicor, Inc. X2816 was selected for the Auxiliary Keyboard system. The X2816 is a 2k by 8 bit EEPROM that runs on a single 5-volt power supply. This is a unique feature for this type of memory chip. The X2816 chip requires no external latches or data preconditioning (i.e. it can be used like RAM) and is designed for applications requiring up to 10,000 write cycles. Since it has the same basic pinout as the popular 2716 EPROM, the X2816 can be placed in a readily available 24 pin socket and requires almost the same interconnect wiring.

Each expansion slot in the Apple //e has the exclusive use of sixteen memory locations for data input and output in the memory space beginning at location \$C090. Slot 1 uses locations \$C090 through \$C09F, slot 2 uses locations \$C0A0 through \$C0AF, and so on through \$C0FF. These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple //e addresses one of the sixteen I/O locations allocated to a particular slot, the signal called 'DEVICE SELECT' switches to the active (low) state. Reference Figure 10, Figure 11 and the following paragraphs

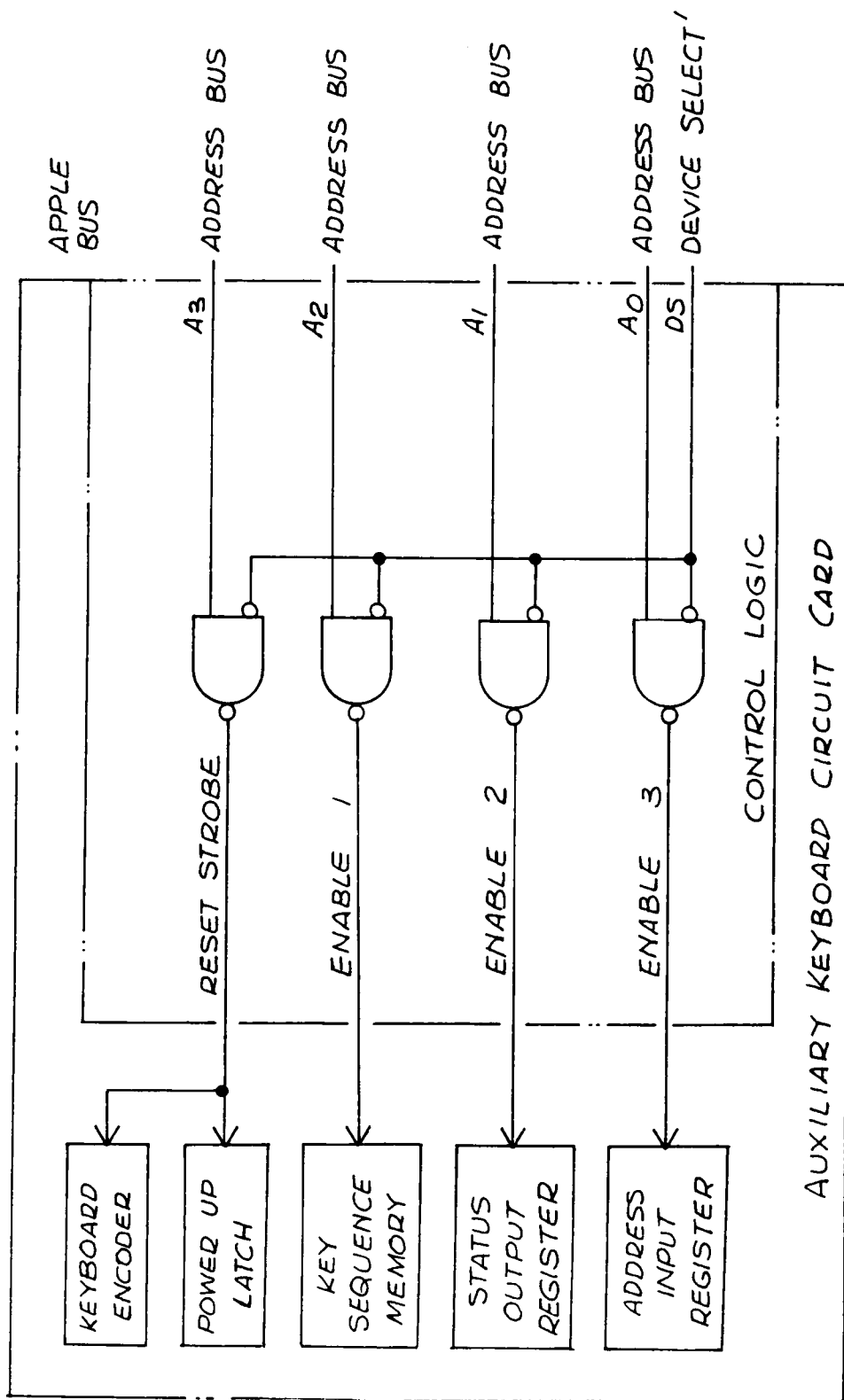


FIGURE 10. Control Logic

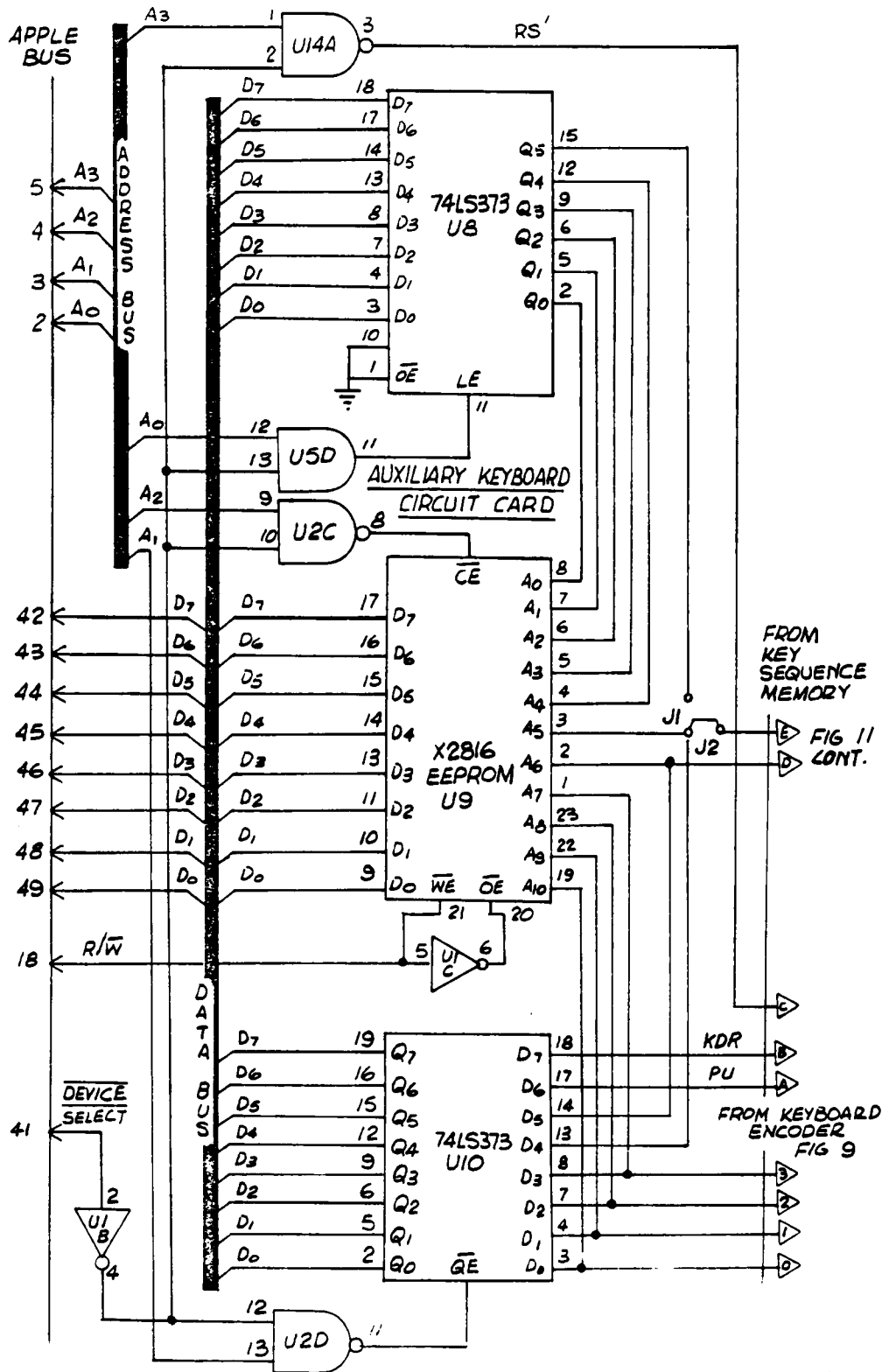


FIGURE 11. Key Sequence Memory

TO
KEY SEQUENCE
MEMORY
FIG 11

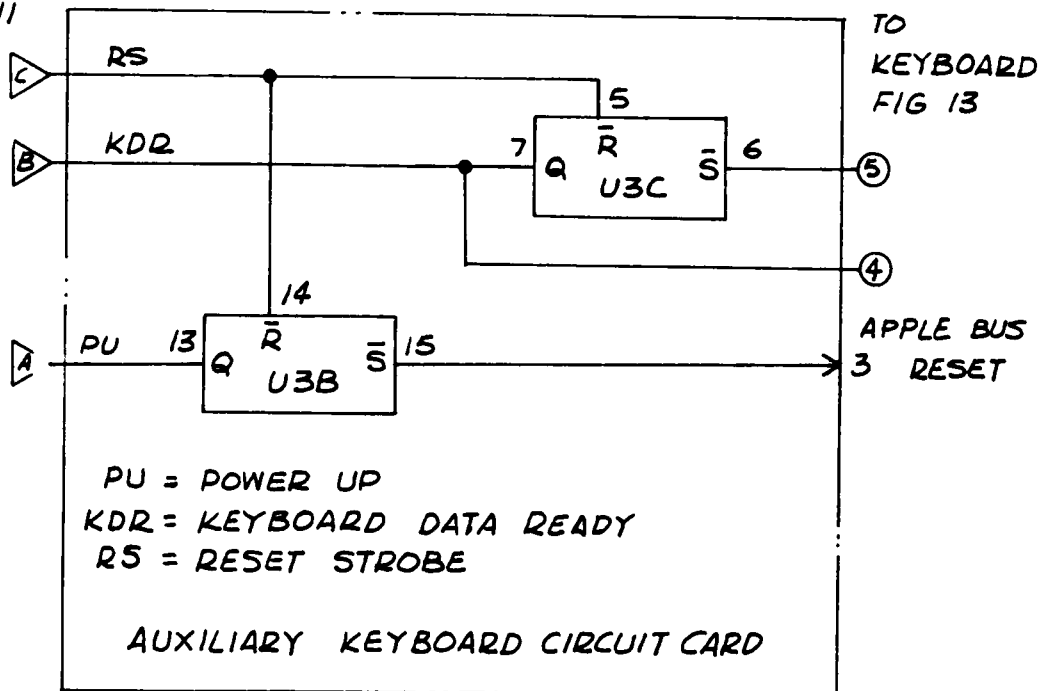


FIGURE 11 (cont.)

for an explanation of the use of the DEVICE SELECT' line and the operation of the key sequence memory subsystem.

The EEPROM selected for the auxiliary keyboard was used to retain 64 key sequences of up to 32 characters (codes) each. Two registers, U8 and U10, and the EEPROM, U9, are accessed using DEVICE SELECT' and address lines A0, A1, and A2, respectively. A fourth address line, A3, along with DEVICE SELECT' is decoded by U14A and used as a Reset Strobe (RS). Register U8 latches the address generated by an input routine for the EEPROM. Five outputs, Q0 - Q4, of register U8 are address inputs A0 - A4, of the EEPROM. Because register U8 generates an input to the auxiliary keyboard circuitry the output enable (OE) line is tied to ground making the outputs always active. The latch enable (LE) line is connected to the address decoder U5D to latch input data into register U8 when the Apple's 6502 CPU is writing to address \$C0D1. Register U10 is an output device from the auxiliary keyboard logic circuitry and is connected opposite to that of U8. The Latch Enable (LE) is tied to +5 volts which enables the input latch at all times. The output enable (OE) is connected to the address decoder U2D. Data from register U10 is output to the CPU data bus, D0-D7, when address \$C0D2 is issued and a read instruction is performed. More discussion on the use and implementation of the address input register U8 and the

system status register U10 follows in the section entitled Other Subsystems.

Output Q5 of U8 and input A5 of U9 are optionally selected by installing jumper J1. With the J1 jumper installed the EEPROM retains 32 different key sequences of up to 64 characters (codes) each. The EEPROM retains 64 different key sequences of up to 32 characters each when jumper J2 is installed. The auxiliary keyboard circuit was designed to have 64 special function keys with up to 32 character codes retained for each key. To select the option to have 32 special function keys that could retain up to 64 character codes each would require software changes.

Board Select Logic

The keyboard has 16 keys which can be utilized as four sets. The active set is selected by output lines Q0 and Q1 of counter U11 (Reference Figure 12). Counter U11 is incremented by way of RS flip-flop U3D and a separate pushbutton located on the keyboard. The single pole double throw (SPDT) pushbutton is debounced by U3D to provide a clean signal to counter U11. Four LED's, also located on the auxiliary keyboard, shown schematically in Figure 13, are driven by counter U11 through decoding logic (i.e.,

TO
KEY
SEQUENCE
MEMORY
FIG 11

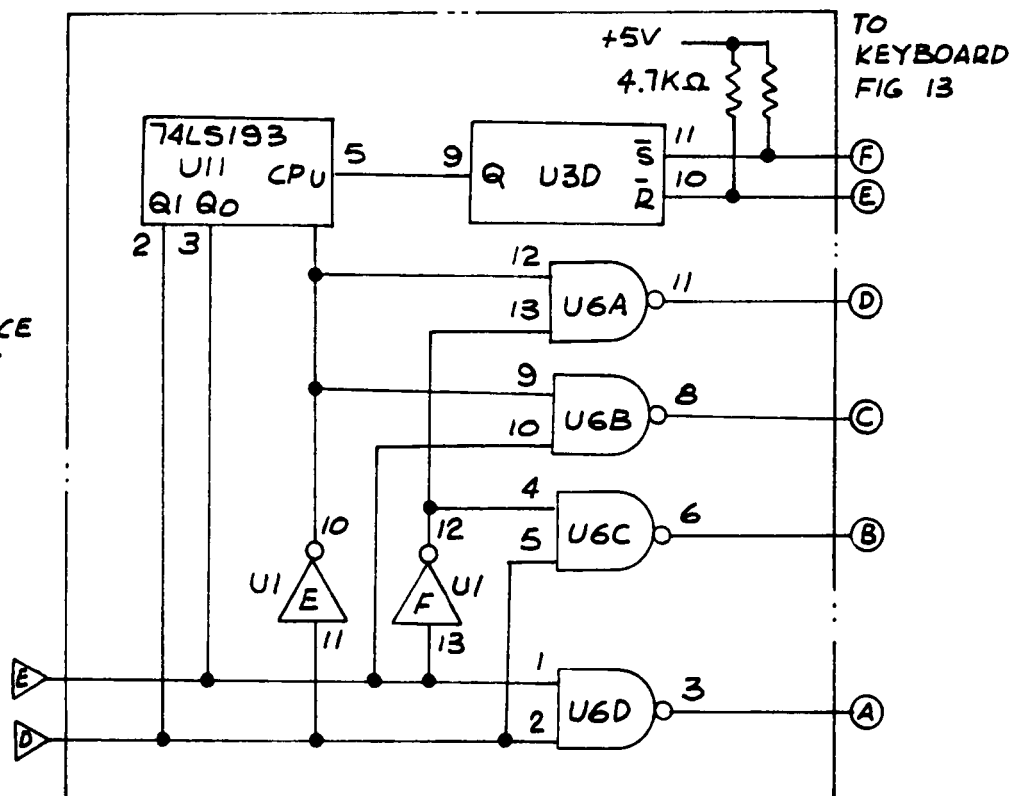


FIGURE 12. Board Select Logic

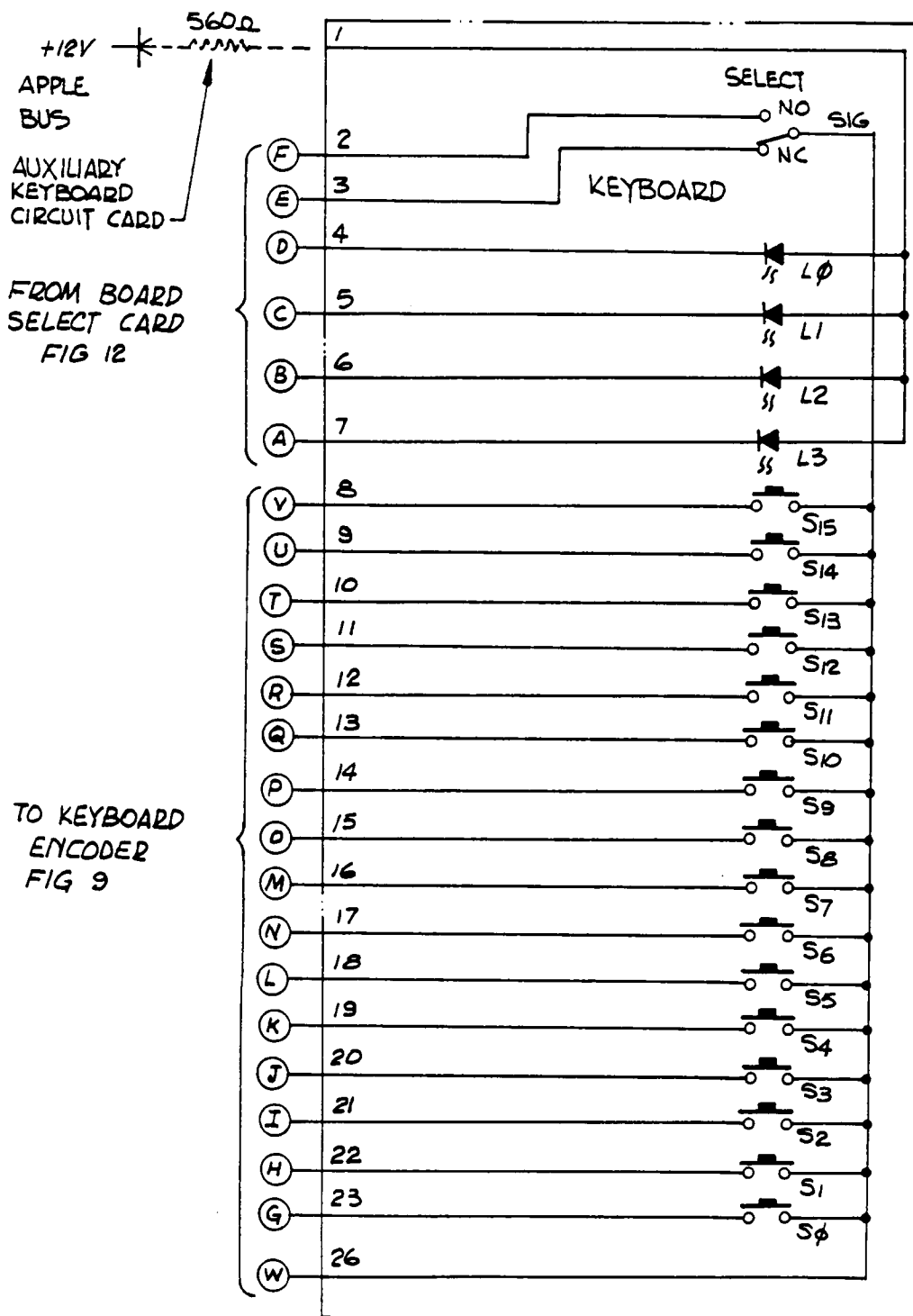


FIGURE 13. Keyboard

inverters U1E, U1F, and NAND gates U6A, U6B, U6C and U6D) for an indication of the selected set of keys.

Keyboard

The keyboard is implemented using 17 momentary pushbutton switches and four flush mounted LED indicators (see Figure 13) in a front sloped metal enclosure (see Figure 1 page 5). No electronics are included in the keyboard enclosure and all wiring to the system circuit card is through a 26-wire flat cable. Sixteen of the keys are arranged in a 4 x 4 matrix with enough space between each key to allow a two-line label of nominal height characters. All labels are integrated into a removable overlay that fits around all sixteen keys and around one of the LED indicators at the top of the keyboard.

Other Subsystems

Control signals as well as data are passed to and from the auxiliary keyboard by the two 74LS373 control registers (U8 and U10 shown on Figure 11 page 36). The 74LS373 is an 8-bit register with integral input latches. As previously mentioned, one of the control registers, U8, is used as an input register from the Apple data bus to supply address information to the EEPROM. U10, implemented as an output register, provides the key address, board

selected and system status information to the Apple data bus. System status includes power up (PU) and Keyboard Data Ready (KDR). The following paragraphs present a more detailed description on how these signals and data are used or generated.

Data input lines D0 - D3 of the system status register U10 are generated by the keyboard encoding logic and represent which one of the 16 keys on the auxiliary keyboard has been pressed.

The outputs from counter U11 of the board select logic are connected to inputs D4 and D5 of the status output register U10. These signals provide a means for the operating software to read the status output register U10 and determine which set of keys has been selected. Inputs D6 and D7 of register U10 are provided by the startup flip-flop U3B and KDR flip-flop U3C. These inputs to register U10 provide status information required by the operating software to operate the auxiliary keyboard. The startup flip-flop U3B is set by the RES' (RESET) signal from the Apple //e bus. When the operating software reads register U10 and detects Q6 set (i.e., input D6 is set), this will indicate that the Apple //e has just powered up and it can execute any initialization procedures that are required. If Q7 of the system status register is set and detected by the operating software, then it is interpreted

that a key on the auxiliary keyboard has been pressed and the key's address has been latched. Both U3B and U3C flip-flops are reset by the RS' signal generated by U14A. A read or write operation to address \$C0D8 causes U14A to go low and produce the Reset Strobe (RS').

Another important component of the hardware design was to exchange the CX ROM on the Apple //e motherboard with a 2764 Programmable ROM. The CX ROM contained the Apple //e firmware input routines that were modified to include the auxiliary keyboard system. This required that the device be programmable. The 2764 EPROM was programmed with all the CX ROM routines except for the patches to the auxiliary keyboard system programs. These patches or changes cause the input routines to jump to the auxiliary keyboard software to handle character input. More discussion follows in Chapter V on the system software patches and the CX ROM.

CHAPTER V

SOFTWARE DESIGN

In the software design a study of the Apple //e's input routines was required. These input routines (called KEYIN.B and GETKEY) were modified to cause program execution to transfer to the auxiliary keyboard input routine, HANDLER-A. HANDLER-A is located on the auxiliary keyboard circuit card in the 2716 EPROM illustrated in Figure 7 page 26. HANDLER-A is capable of handling both 40-column mode and 80-column mode input in addition to input from the auxiliary keyboard. HANDLER-B was written to take input from the Apple //e's keyboard and store it in the auxiliary keyboard program memory. HANDLER-B is located in the same 2716 EPROM as HANDLER-A. Both of these routines were developed in the Apple //e's RAM and then loaded to the 2716 EPROM. To implement this process an EEPROM utility program was written. This program reads the Apple //e's RAM and stores each byte into an EEPROM. The EEPROM was then removed from the Apple //e and installed in a PROM burner. The contents of the EEPROM were then transferred to the 2716 EPROM. The following paragraphs present a detailed description of the software design.

Because the input routines in the Apple //e were written in 6502 assembly language, all programming was done in 6502 assembly language. The Apple Editor Assembler version 1.0 was used to compile the assembly code [8].

Interface Definitions and Requirements

To interface the auxiliary keyboard, interface addresses, flags, counters, and temporary memory locations were designed into the hardware. Four addresses, \$COD8, \$COD4, \$COD2, and \$COD1 were used to perform the required hardware interface functions of the input routines. Address \$COD2, AUXFLG, contains information and status of the auxiliary keyboard. Bit 7 of this address is set when a key has been pressed on the auxiliary keyboard. Bit 6 is set when a system power up is completed. Both of these status flags are reset when an access to \$COD8, AUXSTRB, is issued. The lower 6 bits of AUXFLG contain the board selected information i.e., which set of keys (1 to 4) and the key selected information (1 to 16). Information written to POINTER (\$COD1) is used to address the key sequence memory. POINTER is incremented from 0 to 31 depending on how many character codes have been or are to be stored in the key sequence memory. Only 5 of the 11 bits that address the key sequence memory are taken from POINTER. In addition to POINTER the remaining 6 bits of

the key sequence memory address are developed from the board selected (2 bits) and the Key selected (4 bits) information. The key selected information is valid when the Keyboard Data Ready (KDR, bit 7 of AUXFLG) flag is set. The generation of the KDR signal was explained in Chapter IV under Keyboard Encoding. When KDR is set, HANDLER A writes to POINTER and reads or writes a character code to AUXCHR (\$COD4). POINTER is then incremented and another character code is read or written to AUXCHR. Writing to AUXCHR stores the character code in the key sequence memory and reading from AUXCHR recalls a previously stored character code.

Other flags, counters and temporary memory locations used by the auxiliary keyboard routines are listed in Table 1. These memory addresses are unique to the Apple //e expansion slot 5 and are called Peripheral Card I/O locations. They are used by the various subroutines within HANDLER-A and HANDLER-B.

Apple //e Character Input Routines

The Apple //e has two input routines. KEYIN is used when the system is in 40-column mode and GETKEY when the system is in 80-column mode. Both routines are located in the system monitor ROM on the Apple //e motherboard. The 80-column routine is used only when an 80-column board is

TABLE 1 Peripheral Card I/O Space Usage

LOCATION	NAME	INTERPRETATION
\$47D	BCD	Binary to bcd conversion routine
\$4FD	COUNTER	Temporary counter register
\$57D	MODEFLG	Determines keyclick or no keyclick
\$5FD	TEMP	Temporary memory register
\$67D	RETFLG	Identifies if HANDLER-A was entered through KEKIN.B

plugged into the auxiliary slot and the card is activated. Using the monitor ROM listings available from Apple Inc., each routine was identified [9]. These are listed in Appendix B.

In the Apple //e the KEYIN routine is located in two different ROM's on the motherboard. Earlier Apple machines such as the Apple II and the Apple II+ had the monitor routines in one ROM called the F8 ROM. To implement the advanced features of the Apple //e more program space was required. Therefore, the CX ROM was added and it occupies the same address space as I/O memory, i.e. from \$C100 to \$CFFF.

The memory device that is enabled in the address space \$C100 to \$CFFF is controlled by soft switches. A soft switch is a location in memory that controls memory space allocation whenever its contents are read or written. By adding the CX ROM, the F8 ROM entry points for character input and output were maintained the same as the Apple II and II+ and thereby making the Apple //e software compatible with the II and II+. A call to the KEYIN routine begins in the F8 ROM space which 1) activates the CX ROM space, 2) jumps to it, 3) executes the KEYIN.B routine, 4) jumps back to the F8 ROM, which then disables or enables the CX ROM and 5) returns to the calling program. When the KEYIN routine is first entered, the

status of the CX ROM (i.e., whether it is enabled or disabled) is pushed onto the stack. Before the return to the calling program is executed this status flag is pulled off the stack to determine if the CX ROM is to be left enabled or disabled.

KEYIN also provides a blinking cursor and allows for system interrupts. When the Apple //e has 80-column support hardware installed in the auxiliary card slot of the motherboard, its program ROM is also overlaid with I/O memory space. Physically the 80-column software is located in the CX ROM along with KEYIN.B and other routines, and it is also enabled or disabled using soft switches. For a program to know which input routine (KEYIN or GETKEY) is active, the address of the active routine is placed in memory at locations \$38 and \$39, called the keyboard input switch, KSW. Individually, location \$38 is called KSWL (for KSW Low) and location \$39 is called KSWH (KSW High). This link holds the starting address of the routine currently being used for single-character input. When the Apple //e is powered up, the 40-column input routine, KEYIN, is activated. To activate the 80-column routine, GETKEY, a "PR#3" instruction is issued from Basic or a "3 Cntrl-K" is issued while in the system monitor program. The system monitor routine, upon receipt of this instruction, loads KSW with the address of the first byte

of slot 3 peripheral-card ROM space. (An 80-column card is installed in the auxiliary card slot.) This will activate the 80-column firmware. Until the 80-column firmware is deactivated, character input is handled by GETKEY. The GETKEY routine does not provide a blinking cursor as does the KEYIN routine. Both GETKEY and KEYIN wait until the user presses a key, then returns with the key code in the accumulator. Also, each routine generates a random number for use by other Apple //e software. While a routine is waiting for the user to press a key, memory locations \$4E (RANDL) and \$4F (RANDH) make up a 16-bit number that is repeatedly incremented. This number keeps increasing from 0 to 65536 then starts over again at 0. The value of this number changes so rapidly that there is no way to predict what it will be after a key is pressed.

The Apple standard input handlers interface to the Apple's hardware encoding circuitry through locations \$C000 and \$C010. They each check for bit 7 of location \$C000, called KBD, to be set indicating that a key has been pressed. The lower 7-bits of location KBD contain the raw key code. Immediately after the code is read an access to location \$C010 is executed which resets bit 7 of KBD. This location, \$C010 is called Keyboard Strobe (KBDSTRB) in the software listings.

Other Systems Character Input Handlers

Microcomputer systems, such as the IBM PC and the TRS-80 or minicomputer systems such as the digital PDP-11 series have character input handlers that are part of the basic operating system software. The hardware interfaces for each are different and the exact function of the input routine may vary slightly, but all perform the task of interfacing to the input hardware and providing the calling program with a formatted key code and status. Status would be 1) any key down, 2) a flag to indicate that a key had been pressed, and 3) possibly hardware status such as operable or inoperable. Therefore, an approach similar to the one presented (i.e., to patch or change the input routine) could be implemented for other microcomputer systems.

Apple //e Software Implementation

A first approach to implementing software for the auxiliary keyboard was to change the input link locations to point to the input handler routine for the auxiliary keyboard. Several problems were encountered. It was found that the system I/O routines did not allow a simple change to the input link KSW. It was required to change both the input and output link to disengage the I/O program. Also at this entry level of the I/O program neither KEYIN nor

GETKEY is directly called for keyboard input. When the disk operating system (DOS) is running, the input link (KSW) holds the address of the disk operating system's input routine. The Disk Operating System (DOS) has internal locations that hold the addresses of the character input routine that is currently active. The I/O program is made up of many other routines besides the basic input routines KEYIN or GETKEY. The I/O program captures many special key sequences that are invoked by typing escape codes on the keyboard. An escape code is obtained by pressing the "ESC" key, releasing it, and then pressing another key on the keyboard. Table 2 lists the escape codes supported by the Apple //e. All of the routines required to implement each function described in Table 2 are included in the I/O program.

Because of the design criteria, all current functions of the Apple //e had to remain intact. To do this the entire I/O program with the necessary patches to service the auxiliary keyboard would have had to be duplicated in the auxiliary keyboard program memory. If this had been the approach taken, other software entry points of the I/O program would have been located at different addresses instead of at the expected memory locations. Further study of the monitor ROM listings, provided by Apple Inc., revealed that the most primitive

TABLE 2 Escape Codes

Code	Function	
(ESC)@	Clears the window and homes the cursor	
(ESC)A	Moves the cursor up one line	
(ESC)B	Moves the cursor right one space	
(ESC)C	Moves the cursor left one space	
(ESC)D	Moves the cursor down one line	
(ESC)E	Clears to the end of the line	
(ESC)F	Clears to the bottom of the window	
(ESC)I	Moves the cursor up one line and turns on escape mode	
(ESC)J	Moves the cursor left one space and turns on escape mode	
(ESC)K	moves the cursor right one space and turns on escape mode	
(ESC)M	Moves the cursor down one line and turns on escape mode	
(ESC)R	Turns on restricted-case mode	*
(ESC)T	turns off restricted-case mode	*
(ESC)4	Switches to 40-column mode	*
(ESC)8	Switches to 80-column mode	*
(ESC)Cntrl-Q	Deactivates the 80-column firmware	*

* = 80-column firmware active

routines which handle the keyboard input could be separated from the I/O program. These routines were KEYIN.B and GETKEY. Thus, in order to retain the same entry points to these two routines, they were left in the monitor ROM but modified to transfer program execution to the auxiliary keyboard input routines. These modifications or patches are discussed in detail in the following section.

System Software Patches

The KEYIN.B and the GETKEY routines are in firmware on the Apple //e motherboard in the CX ROM. KEYIN.B is 99 bytes long, begins at location \$C288 and ends at \$C2EA. Instead of putting a jump instruction to the auxiliary keyboard program memory at the first byte of the KEYIN.B memory space, the KEYIN.B space was utilized for the program code which handles the blinking cursor that is unique and original to KEYIN.B. Also, other program code was included in the KEYIN.B memory space to reduce the program code required in the auxiliary keyboard program memory space. The modified KEYIN.B program is listed in Appendix C. At lines 37 thru 43 the address of the auxiliary keyboard input routine and a return flag is pushed onto the stack. When a key is pressed on the system keyboard or the auxiliary keyboard a return instruction (RTS) is executed. This will cause program execution to

transfer to address \$C502. \$C502 is the entry to the auxiliary keyboard input routine HANDLER-A. The return flag tells HANDLER-A that it was entered from the 40-column mode input routine KEYIN.B. This information is required so that HANDLER-A, upon completion, will return program execution to the correct place in memory. Lines 56 and 57 were added to check for the KDR signal from the auxiliary keyboard hardware. These represent the major changes to the original KEYIN.B routine.

The GETKEY routine is much shorter than KEYIN.B. It is only 15 bytes long and is located at \$CB15. The extent of GETKEY was to increment the random number (RANDL and RANDH) and wait for a keypress on the system keyboard. The program code to increment the random number, service the system keyboard and service the auxiliary keyboard was included in the auxiliary keyboard program memory. Therefore a 3-byte jump instruction at the GETKEY routine entry address to direct program execution to the auxiliary keyboard program memory was all that was required to patch the original GETKEY routine.

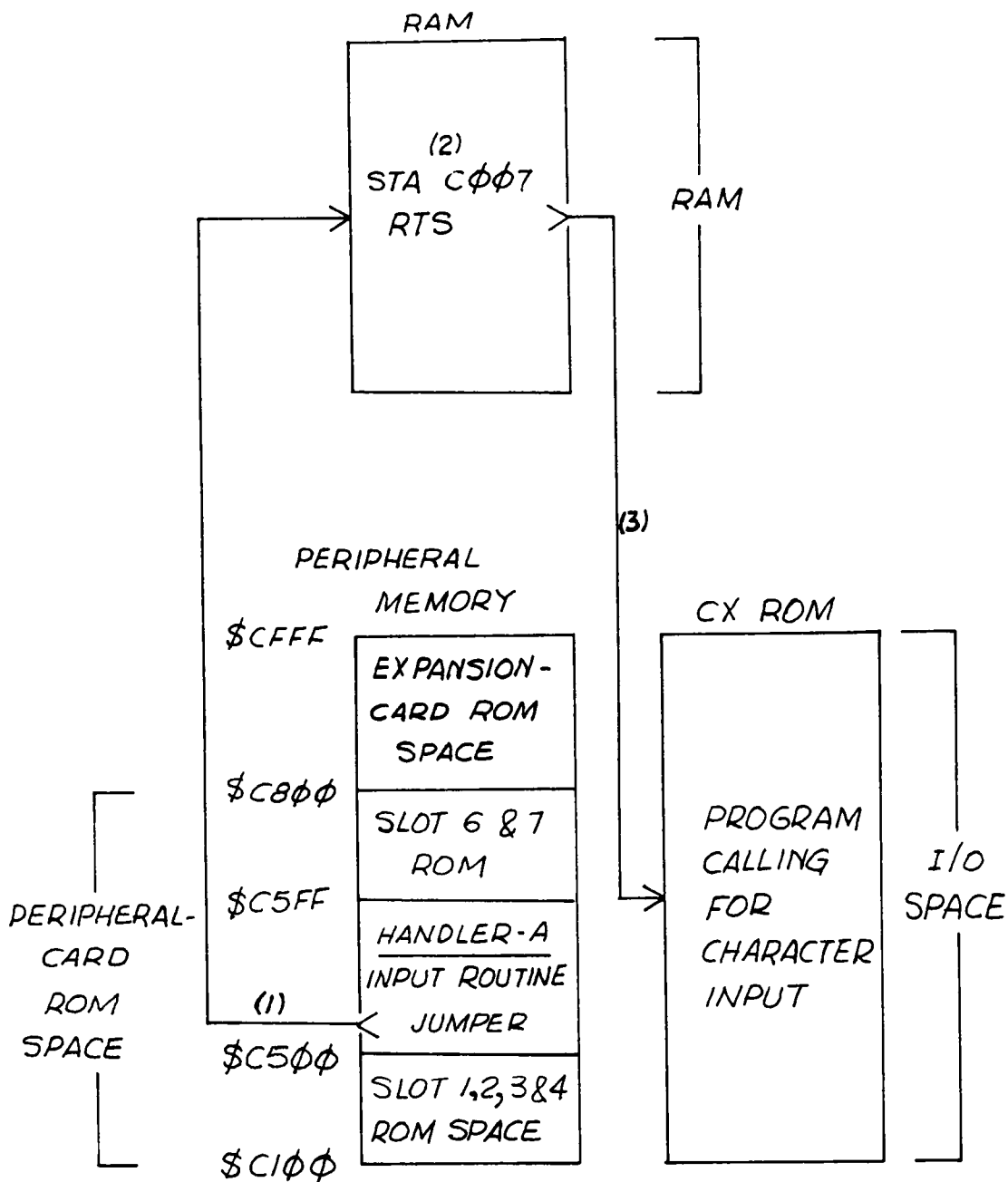
Auxiliary Keyboard Input Routine

HANDLER-A contains the program code to service the system keyboard and the auxiliary keyboard. Also, when the Apple //e system is powered up, an initialization procedure

to set up program flags and clear counters is performed. When KEYIN.B is called to input a character, the modified KEYIN.B routine manages the blinking cursor while it is waiting for the user to press a key either on the system keyboard or the auxiliary keyboard. Once a key has been pressed program execution jumps from the CX ROM (location of KEYIN.B) to the F8 ROM. The CX ROM is disabled and the program then jumps to the HANDLER-A input routine. The CX ROM is disabled because the auxiliary keyboard program memory is located in the same address space as the CX ROM. CX ROM is disabled by writing or reading to a soft switch at \$C006. HANDLER-A checks the auxiliary keyboard flag (KDR) for the key selected. If the key is not one of the auxiliary keyboard keys, then the system keyboard is checked for the key selected. When HANDLER-A is entered from KEYIN.B, either a system or an auxiliary keyboard key has been pressed. When the Apple //e is in the 80-column mode and GETKEY is called for an input character, the program immediately transfers execution to HANDLER-A. The auxiliary keyboard is checked for a key closure, then the system keyboard is checked for a key closure and this process continues until a key is pressed. Once a key is pressed, either on the auxiliary or the system keyboard, the input service routine is executed and returns to the calling program with the character code in the accumulator.

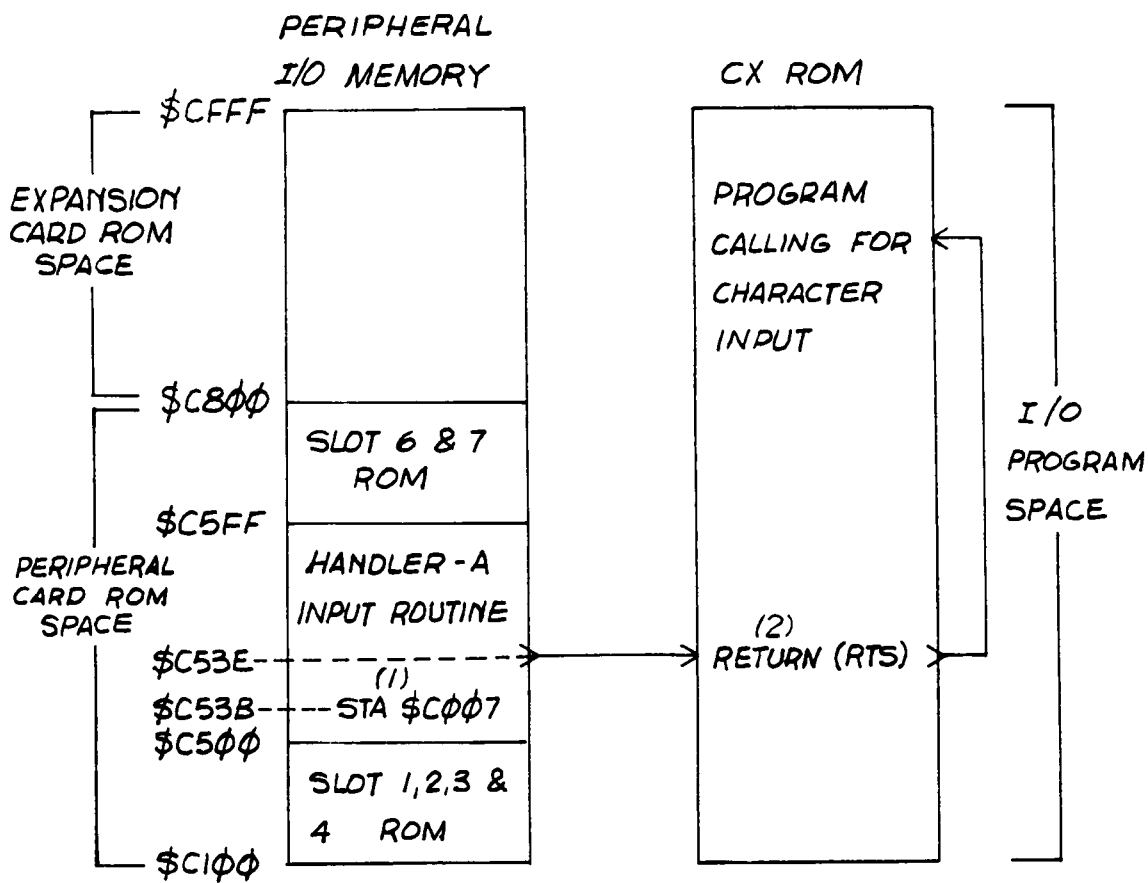
Special programming was required in the input routine (HANDLER-A) to return to the CX ROM space. One way to return to a calling program located in a different memory chip but in the same address space would be to first jump to a subroutine outside the common address space as shown in Figure 14. Then, the subroutine would disable the program memory, enable the CX ROM (by reading or writing to \$C007), and issue a return instruction. To implement this scheme would require that part of the initialization procedure of the HANDLER-A routine load the required code into some unused memory area. Unfortunately, it could not be verified that a certain memory area outside the I/O space would always be available. Therefore, another approach was taken as is illustrated in Figure 15.

The CX ROM memory was examined using the system monitor program and a return instruction was found at memory location \$C53E. To enable the CX ROM and disable program memory, an access to location \$C007 is required. Thus, the HANDLER-A program code was written to access location \$C007 at memory address \$C53D. The program counter of the 6502 CPU is then pointing to location \$C53E. The next instruction loaded and executed is then from the CX ROM and is a return instruction. This action exits the HANDLER-A routine, enables the CX ROM and returns program



<u>STEP</u>	<u>ACTION</u>
1	JUMP TO RAM
2	TURN ON CX ROM
3	RETURN

FIGURE 14. Returning to CX ROM Method 1



<u>STEP</u>	<u>ACTION</u>
1	TURN ON CX ROM
2	RETURN

FIGURE 15. Returning to CX ROM Method 2

execution to the CX ROM program that had requested character input.

Auxiliary Keyboard Programming Software

In the following paragraphs the auxiliary keyboard programming software, HANDLER-B, and the various subroutines within HANDLER-B are discussed. One of these subroutines is a binary-to-BCD routine named BCD. Also, there are two output subroutines, OUT2 and OUTPUT. OUTPUT is used by HANDLER-B to output messages to the programmer. OUT2 is a special routine that converts non-displayable character codes to displayed character codes and displays them on the Apple //e's CRT screen. Additionally, there is INCH, a character input routine. The HANDLER-B programming software listing is included in Appendix D.

The programming software, HANDLER-B, is located in the expansion-card ROM address space, \$C800 to \$CFFE. The expansion-card ROM address space is used by each peripheral card plugged into the Apple expansion slots. Upon return from a subroutine called by HANDLER-B outside its memory space, an interface program must ensure that any other enabled expansion-card ROM has been disabled and that the auxiliary keyboard expansion-card ROM is enabled. To implement this procedure each time a subroutine call outside HANDLER-B is required, a call is first made to an

interface routine in the peripheral-card ROM space (part of HANDLER-A). The interface program executes a call to the original required subroutine. When the required subroutine is complete and executes a return, the interface program issues a command to turn off any enabled expansion-card ROM (by addressing \$CFFF), then another return instruction is executed by the interface program. In the design of the hardware, location \$CFFF is decoded to turn off the expansion-card ROM used by the auxiliary keyboard. Conversely, access to the peripheral-card ROM space on the auxiliary keyboard will enable the expansion-card ROM on the auxiliary keyboard. After an access to \$CFFF, if the next instruction executed is from the slot 5 peripheral-card ROM space (HANDLER-A), the hardware on the auxiliary keyboard interface card will enable the slot 5 expansion-card ROM. Thus, the HANDLER-B routine is re-enabled and will continue program execution.

A binary to BCD (Binary Coded Decimal) routine was required in HANDLER-B to convert the key number and the key set number, from AUXFLG, to a packed BCD format. These numbers were further converted to a set of ASCII bytes, with the most significant bit set, in order to be displayed on the Apple screen. This is the format required by the Apple display routine, COUT.

The binary to BCD algorithm implemented is the following. Let $b_7, b_6, \dots, b_0$ represent the bit values of M , an 8-bit binary number. Then the number is given by the expression

$$M = b_7 2^7 + b_6 2^6 + b_5 2^5 + \dots + b_1 2^1 + b_0.$$

The binary-to BCD algorithm depends on the fact that this expression may be rewritten to give

$$M = ((((((b_7 + b_6) 2 + b_5) 2 + b_4) 2 + b_3) 2 + b_2) 2 + b_1) 2 + b_0.$$

Bit 7, b_7 , is moved to memory location BCD. BCD is doubled by adding it to itself in the decimal mode. Next, b_6 is added to BCD and the result is doubled. The doubling process is repeated seven times, and each of the eight bit values, $b_7, b_6, \dots, b_0$, are added by moving the bit value into the carry flag and then performing addition in the decimal mode. The X register serves as a bit counter. Once the routine is complete, a two-digit binary coded decimal number equivalent to the binary number is in memory location BCD [5].

Two output routines were needed to display messages and control codes. The capability to output the control codes is needed when the user is entering the codes to be programmed into the auxiliary keyboard key sequence memory.

Much of the time the entered key sequences will contain control codes. Normally control codes are not displayable characters. A routine called OUT2 was written to convert a control code to a displayable character code and display it. The control codes of the ASCII character set are represented by the numbers from \$00 to \$1F. Before a call is made to the OUT2 subroutine the accumulator is loaded with the code to be displayed. A subroutine call to OUT2 is then executed. The OUT2 subroutine tests the code in the accumulator to see if it is a displayable code. To do this, \$20 is subtracted from the input code. The result is tested for a positive or negative value. If the result is positive, OUT2 completes with a normal character output to the monitor output routine COUT. If the result of the subtraction is negative, the input code is used to index a table of replacement displayable codes. The table is made up of 31 (\$1F) displayable character codes that represent the keys on the system keyboard that are pressed, along with the control key, to produce the corresponding control codes. These are listed in Table 3. After a replacement code is loaded from the table, the inverse video flag is set, and the code is output to the Apple //e's screen through COUT. This will cause the output on the Apple //e screen to be in reverse video making control codes

TABLE 3 Control Codes

Key	Display	ASCII	Interpretation
Cntrl-@	*	NUL	Null(\$00)
Cntrl-A	A	SOH	Start Of Header(\$01)
Cntrl-B	B	STX	Start of Text(\$02)
Cntrl-C	C	ETX	End of Text(\$03)
Cntrl-D	D	EOT	End Of Transmission(\$04)
Cntrl-E	E	ENQ	Enquiry(\$05)
Cntrl-F	F	ACK	Acknowledge(\$06)
Cntrl-G	G	BEL	Bell(\$07)
Cntrl-H	H	BS	Backspace(\$08)
(Left-Arrow)	H		
Cntrl-I	I	HT	Horizontal Tab(\$09)
(TAB)	I		
Cntrl-J	J	LF	Linefeed(\$0A)
(Down-Arrow)	J		
Cntrl-K	K	VT	Vertical Tab(\$0B)
(Up-Arrow)	K		
Cntrl-L	L	FF	Form Feed(\$0C)
Cntrl-M	M	CR	Carriage Return(\$0D)
(RETURN)	M		
Cntrl-N	N	SO	Shift Out(\$0E)
Cntrl-O	O	SI	Shift In(\$0F)
Cntrl-P	P	DLE	Data Link Escape(\$10)
Cntrl-Q	Q	DC1	Device Control 1(\$11)
Cntrl-R	R	DC2	Device Control 2(\$12)
Cntrl-S	S	DC3	Device Control 3(\$13)
Cntrl-T	T	DC4	Device Control 4(\$14)
Cntrl-U	U	NAC	Negative Acknowledge(\$15)
(Right-Arrow)	U		
Cntrl-V	V	SYN	Synchronization(\$16)
Cntrl-W	W	ETB	End of Text Block(\$17)
Cntrl-X	X	CAN	Cancel(\$18)
Cntrl-Y	Y	EM	End of Medium(\$19)
Cntrl-Z	Z	SUB	Substitute(\$1A)
Cntrl-[	[	ESC	Escape(\$1B)
(ESC)	[		
Cntrl-\	/	FS	File Seperator(\$1C)
Cntrl-]	]	GS	Group Seperator(\$1D)
Cntrl-6	6	RS	Record Seperator(\$1E)
Cntrl- <u></u>	-	US	Unit Seperator(\$1F)

* = Not used

distinguishable from non-control codes. When COUT returns, the inverse flag is reset for normal video and OUT2 returns to the calling program.

To output a message to the Apple //e screen, OUTPUT was written. OUTPUT takes advantage of an Assembler directive, ASC. The ASC directive instructs the Assembler to create a list of ASCII codes from the input that follows the ASC mnemonic in the assembler source code [8]. The unique feature of ASC is that the most significant bit (MSB) of the last ASCII code created for the list is set. Each message is entered in the source code using the ASC directive and all messages are indexed by an offset number of bytes (ASCII characters) from the first message in the table of messages. A prerequisite to calling the OUTPUT subroutine is to load the Y register with the offset that points to the desired message to be output. The OUTPUT subroutine then loads the accumulator with the first character code of the message and checks to see if the MSB is set. If not, the code is output to the screen through the monitor subroutine COUT. When the last character code in the message is detected (MSB set), the code is compared to a predetermined delimiter code. If these codes do not match, then a carriage return code is generated and output to the screen. Conversely, when the delimiter code is the last character code in a message, a carriage return is not

output to the screen. This feature allows a message to be output to the screen and the cursor be left at the next character position on the screen. This will allow a response to be typed on the same line as a message.

Both OUTPUT and OUT2 are required by the auxiliary keyboard programming software to interface to the user. Messages are used to tell the user what to do and to request responses from the user.

INCH was written and employed to take input from the system keyboard during auxiliary keyboard programming. When a key on the auxiliary keyboard is selected to be programmed, the Keyboard Data Ready (KDR) signal is set (reference Chapter IV and the section entitled Keyboard Encoding). With KDR set, the normal input routines KEYIN.B and GETKEY will not take input from the system keyboard because the state of KDR will indicate that the auxiliary keyboard has input data ready. Therefore, INCH was written to take input from the system keyboard only.

When INCH is called for character input, it manages a blinking cursor while waiting for a key to be pressed on the system keyboard. Once a key is pressed, INCH returns with the character code in the accumulator.

EEPROM Programming Utility

The 2716 EPROM that is used to store the programs of the auxiliary keyboard system was replaced with an X2816 EEPROM during the development of the auxiliary keyboard software in order to provide a method to debug the program for the 2716 EPROM. The auxiliary keyboard hardware was not designed to provide the necessary signals and voltage levels required to directly program the 2716 EPROM. The X2816 EEPROM is capable of being programmed by the standard level signals provided by the Apple //e's expansion card slots. Writing data to the X2816 is analogous to writing data into a static RAM except that a 10 milli-second wait must be inserted between each byte write. MOVEIT was written to move program code from the Apple //e's RAM to the X2816 EEPROM (Reference Appendix E). The source starting and ending addresses and the destination address are poked into the MOVEIT routine and then executed. When writing to the EEPROM, an 11 milli-second wait is inserted and is followed by a read operation to verify that the information was written properly. If the read information is incorrect, error flag #1 is incremented and the write cycle is performed a second time. This time an arbitrarily selected 0.33 second wait is inserted before the write is again verified. Error flag #2 is incremented if the second verification fails. No further attempts are made to write

that byte of information. MOVIT continues to the next byte in the source code until the data at the ending address has been transferred. After all data has been transferred, error flag memory locations can be checked to ensure that all data was transferred into the EEPROM correctly.

During the development of the MOVEIT program and the other system software it became a requirement to add the second write/verify cycle to the MOVEIT program. Many times 40 to 50 bytes would not write correctly to the EEPROM. When the second write cycle was added with a longer delay inserted, all bytes that did not transfer in the first write cycle transferred properly in the second write cycle.

Once the development of the software was complete, the X2816 was removed from the auxiliary keyboard interface card and inserted into an EPROM programmer. The auxiliary keyboard software was read into the programmer and then it was loaded into the 2716 EPROM. The 2716 EPROM was then returned to the interface card of the auxiliary keyboard system and the operation of the software verified.

CHAPTER VI

APPLICATIONS

The utility of the auxiliary keyboard was verified with several user software programs and operating systems to prove compatibility. These programs are listed in Table 4. It was found that a few (indicated by a "no" in the * column) did not recognize the auxiliary keyboard. One of these, the Supertext Word Processing Program, uses its own character input routine. Since the program does not call the standard Apple input routines the auxiliary keyboard is bypassed. That is, when a key on the auxiliary keyboard is pressed, nothing happens unless the program running uses the standard input routines KEYIN and GETKEY. All CP/M tested software does call for the standard input routines.

Tables 5 and 6 show how the auxiliary keyboard was programmed for use with the Apple DOS and CP/M operating systems, respectively. Commands that are most frequently used were selected.

Table 7 illustrates sixteen of the Applesoft basic commands and four system commands that were programmed into the auxiliary keyboard.

TABLE 4 Software Compatibility

Name	Maker	Type	*
Apple Assembler	Apple	6502 Assembeler	no
Supertext	Muse	Wordprocessor	no
Applesoft Basic	Apple	Language	yes
ProDos	Apple	O.S.	yes
Apple DOS 3.3	Apple	O.S.	yes
CP/M Version 2.2	Digital Research	O.S.	yes
Wordstar		Wordprocessor	yes
dBase II	Ashton Tate	Data Base	yes
M Basic	Microsoft	Language	yes
Home Accountant	Continental	Accounting	yes
Dollars & Sense	Monogram	Accounting	no
Apple Plot	Utility	Apple	yes

* = Compatible

TABLE 5 Apple DOS System Commands

Key	Label	Key Sequence
1	PROGRAM	CALL 50432r
2	PRINTER	PR#1r
3	SCREEN	PR#0r
4	RUN ASSEMBLER	BRUN EDASM.OBJr
5	DELETE	DELETES
6	SAVE	SAVEs
7	LOAD	LOADs
8	RUN	RUNr
9	80-Column ON	PR#3r
10	80-Column OFF	(ESC)Cntrl-Q
11	Restrict Mode ON	(ESC)R
12	Restrict Mode OFF	(ESC)T
13	CATALOG,D1	CATALOG,D1r
14	CATALOG,D2	CATALOG,D2r
15	LIST	LISTr
16	HOME	HOMEr

r = (RETURN)

s = (SPACE)

TABLE 6 CP/M System Commands

Key	Label	Key Sequence
1	LOAD	LOADs
2	SAVE	SAVEs
3	TYPE	TYPEs
4	WARM BOOT	Cntrl-C
5	ERASE	ERAs
6	STATUS	STATr
7	PRINT	Cntrl-P
8	EDIT	EDr
9	ELITE	LPRINT CHR\$(27);"E"r
10	PICA	LPRINT CHR\$(27);"N"r
11	132COL	LPRINT CHR\$(27);"Q"r
12	Not programmed	
13	DIRECTORY	DIRr
14	DISK A	A:r
15	DISK B	B:r
16	PIP	PIPs

r = (RETURN)

s = (SPACE)

TABLE 7 Basic Language Commands

Key	Label	Key Sequence
1	HOME	HOMEr
2	GOTO	GOTOs
3	READ	READs
4	DATA	DATAs
5	GOSUB	GOSUBs
6	RETURN	RETURNr
7	OPEN	OPENs
8	CLOSE	CLOSEs
9	FOR	FORs
10	NEXT	NEXTs
11	IF	IFs
12	THEN	THENs
13	RUN	RUNr
14	LIST	LISTr
15	STOP LIST	Cntrl-C
16	PRINT	PRINTs

r = (RETURN)

s = (SPACE)

A set of commands used for editing, appending and modification functions of the dBase II relational data base program were also programmed into the auxiliary keyboard. These are listed in Table 8. Since many of these commands are very obscure and hard for a user to remember, the auxiliary keyboard makes the dBase II program easier to use.

TABLE 8 dBase II Commands

Key	Label	Key Sequence
1	EXIT-NO SAVE	Cntrl-Q
2	EXIT-SAVE	Cntrl-W
3	PRINTER	Cntrl-P
4	DISPLAY	DISPLAYr
5	DEL CHARACTER	Cntrl-G
6	DEL DATA	Cntrl-Y
7	DEL FIELD	Cntrl-T
8	DEL MARK	Cntrl-U
9	INSERT	Cntrl-V
10	INSERT FIELD	Cntrl-N
11	WRITE	Cntrl-C
12	WRITE & BACKUP	Cntrl-R
13	LEFT	Cntrl-S
14	RIGHT	Cntrl-D
15	DOWN	Cntrl-X
16	UP	Cntrl-E

r = (RETURN)

s = (SPACE)

CHAPTER VII

RESULTS AND CONCLUSIONS

As-Built System Specifications

All the basic design specifications described in Chapter II were either met or exceeded. A 17-key front sloped keyboard was fabricated to house an equivalent of 64 programmable special function keys. Sixteen keys make up a set of programmable function keys and an additional key selects one of four sets of 16 keys to be active; that is, each key of the 16-key set on the auxiliary keyboard can be programmed to generate four separate functions. In addition, each special function key can be programmed to generate up to a 32-code key sequence. The key sequences can contain every code generated by the Apple //e keyboard except a null character. The programmed key sequences are maintained indefinitely in an EEPROM when the system power is turned off. The 16 programmable keys are mounted in a 4 x 4 matrix on the auxiliary keyboard. An overlay identifying the function of each key is made to fit around the matrix of keys and around a LED (Light Emitting Diode) that indicates which set of keys is selected. The LED

indicators and the keyswitch matrix are aligned in such a way as to allow the overlay to be flipped over and provide the function identifications on the back side for a second set of keys. The integrity of all previous functions of the Apple //e was maintained. Available memory was not reduced.

In addition to meeting the basic design criteria a program which generates an audible keyclick was included in the design. The option to have the keyclick is given to the user when he is programming the auxiliary keyboard. When keyclick is selected, each time a key is pressed on the system keyboard or the auxiliary keyboard a click sound is heard. This provides feedback to the user that a key has been pressed without having to monitor the Apple screen.

Hardware Compatibility

The system interface circuit card and its hardware is compatible with the earlier Apple microcomputers, such as the Apple II and the Apple II+. However, because the CX ROM is present only in the Apple //e system, the 2764 EPROM chip that was used to replace the CX ROM would not be compatible with other systems.

A similar hardware system could be developed for other microcomputers such as the TRS-80 or an IBM PC. The

interfacing method to the system bus would be different but the same concept could be implemented.

Software Compatibility

The Apple //e system software patches that were made in the CX ROM would have to be made in the F8 ROM on the Apple II and Apple II+ machines. The 80-column patch would be the most difficult to make because the earlier Apple machines did not have 80-column software built into the machine like the Apple //e does. When an 80-column format was needed for the earlier machines, a peripheral card was plugged into slot 3 to provide an 80-column output.

The auxiliary keyboard program that is located on the interface circuit card would only require minor changes to operate in an Apple II or Apple II+. A similar concept could be utilized in the software design for a TRS-80 or an IBM PC but of course the software as presented would not be compatible with either of these two microcomputer systems.

LIST OF REFERENCES

LIST OF REFERENCES

1. Apple II Reference Manual For //e Only. Cupertino, California: Apple Computer, Inc., 1982. Pages 164-165.
2. VT100 User Guide. Manard, Massachusetts: Digital Equipment Corp., January 1979.
3. Visual 200 Reference Manual. Andover, Massachusetts: Visual Technology, Inc., November 1982.
4. DBASE II User Manual. Culver City, California: Ashton-Tate. Page 19.
5. De Jong, Marvin L., Apple II Assembly Language. Indianapolis, Indiana: Howard W. Sams Co., Inc., 1982.
6. Tenny, Ralph, Keyboard Encoding Schemes, Computers & Electronics. New York, New York: Ziff-Davis Publishing Co., October 1983. Pages 79-83.
7. Keryan, Michael, Add EEPROM To Your Computer, Computers & Electronics. New York, New York: Ziff-Davis Publishing Co., October 1983. Pages 88-89.
8. Apple 6502 Assembler/Editor. Cupertino, California: Apple Computer, Inc., 1980.
9. Apple II Reference Manual Addendum: Monitor ROM Listings For //e Only. Cupertino, California: Apple Computer, Inc., 1982.

APPENDIXES

APPENDIX B

Apple Standard Input Routines

KEYIN, 40-Column Input Routine

```

1  KEYIN      LDY    #6
2              JMP    GTOCX      ;(FBB4)
3  GTOCX      PHP      ;SAVE USER IRQ STATE
4              SEI      ;INHIBIT DURING BANKSWITCH
5              BIT     RDCXROM    ;(C015)GET CURRENT STATE
6              PHP      ;SAVE ROMBANK STATE
7              STA     SETINTCXROM ;(C007) SET ROMS ON
8              JMP     $C100
9              CPY     #6          ;IS IT KEYIN
10             BNE     B.FUNCNK    ;NO
11             JMP     B.KEYIN     ;(C288)YES
12  B.KEYIN    STA     TEMP1       ;(0478)SAVE ORIGINAL CHAR
13             PLA      ;HOLD ONTO
14             TAY      ;CXBANK STATUS
15             PLA      ;GET USER'S
16             PHA      ;IRQ STATE
17             ROR     A          ;MOVE IRQ BIT TO
18             ROR     A          ;THE
19             ROR     A          ;CARRY
20             TYA      ;PUT CXBANK STATUS
21             PHA      ;BACK ON STACK
22             TXA      ;SAVE
23             PHA      ;XREG
24             CLV      ;ASSUME NOT INTERRUPTIBLE
25             BCS     B.KEYIN2    ;=½WE WERE RIGHT
26             BIT     SEV        ;SAY "INTERRUPTIBLE" (CF00)
27  B.KEYIN2   LDA     #$SFF      ;CURSOR=NORMAL DELETE
28             LDY     CH         ;(0024)
29             STA     (BASL),Y    ;(0028)
30             JSR     KEYDLY      ;WAIT FOR A KEY
31             BCS     GOTKEY      ;=½GOT ONE (C2B5)
32             LDA     TEMP1       ;REPLACE ORIG CHAR
33             LDY     CH
34             STA     (BASL),Y
35             JSR     KEYDLY      ;WAIT FOR A KEY
36             BCS     GOTKEY      ;=½GOT ONE
37             BCC     B.KEYIN2    ;(ALWAYS TAKEN)
38  GOTKEY     LDA     TEMP1       ;RESTORE ORIGINAL
39             LDY     CH
40             STA     (BASL),Y    ;CHARACTER

```

```

41          PLA          ;RESTORE
42          TAX          ;XREG
43          LDA   KBD     ;(C000) GET THE NEW
                        KEYSTROKE
44          STA   KBDSTRB ;CANCEL THE STROBE
45          BMI   F.RETURN ;(ALWAYS TAKEN)(C2EB)
46 ***EXIT EITHER EXIT WITH OR WITHOUT
47 * ENABLING I/O SPACE
48          PLP          ;GET PRIOR I/O DISABLE
49          BMI   F.RET1  ;=½LEAVE IT DISABLED
50          JMP   FUNCEXIT ;=½EXIT & ENABLE I/O (FD29)
51 F.RET1     JMP   FUNCEXIT+3 ;EXIT DISABLED (FD2C)
52 FUNCEXIT   STA   SETSLOTCKROM;RESTORE BANK
53 FUNCEXIT+3 PLP          ;RESTORE IRQ
54          RTS          ;RETURN TO CALLER

```

```

1 KEYDLY     LDX   #$0C    ;SHORT DELAY FOR IRQ
2           BVS   IK1      ;-½INTERRUPTIBLE
3           LDX   #$31     ;LONG DELAY FOR NO IRQ
4 IK1        LDY   #0
5 IK2        BVC   IK2A    ;=½NOT INTERRUPTIBLE
6           PHP          ;SAVE OFLOW
7           JSR   SNIFFIRQ ;(FC75)ALLOW IRQ
8           PLP          ;RESTORE OFLOW
9 K2A        INC   RNDL    ;004E
10          BNE   IK3
11          INC   RNDH
12 K3        LDA   KBD     ;(C000) KEYPRESS?
13          BMI   KDRETY   ;=½YES (C2E9)
14          DEY
15          BNE   IK2
16          DEX
17          BNE   IK1
18 DRETN     CLC
19          BCC   KDRET
20 DRETY     SEC
21 DRET      RTS

```

GETKEY, 80-Column Input Routine

```

1 GETKEY     INC   RNDL    ;BUMP RANDOM SEED
2           BNE   GETK2
3           INC   RNDH
4 GETK2      LDA   KBD     ;KEY PRESSED?
5           BPL   GETKEY   ;=½NOPE
6           STA   KBDSTRB  ;CLEAR STROBE
7           RTS

```

APPENDIX C

Patched System Input Routines

KEYIN.B Input Routine

1	RTFLG	EQU	\$67D
2	FUNEXIT	EQU	\$FD29
3	SNIFFIRQ	EQU	\$FC75
4	AUXFLG	EQU	\$COD2
5	RANDL	EQU	\$4E
6	RANDH	EQU	\$4F
7	TEMP1	EQU	\$478
8	CH	EQU	\$24
9	BASL	EQU	\$28
10	KBD	EQU	\$C000
11		ORG	\$288
12		STA	TEMP1
13		PLA	
14		TAY	
15		PLA	
16		PHA	
17		ROR	A
18		ROR	A
19		ROR	A
20		TYA	
21		PHA	
22		TXA	
23		PHA	
24		CLV	
25		BCS	KEYIN2
26		BIT	SEV
27	KEYIN2	LDA	#\$FF
28		LDY	CH
29		STA	(BASL),Y
30		JSR	KEYDLY
31		BCS	GOTKEY
32		LDA	TEMP1
33		LDY	CH
34		STA	(BASL),Y
35		JSR	KEYDLY
36		BCC	KEYIN2
37	GOTKEY	LDA	#\$C5
38		PHA	
39		LDA	#\$02
40		PHA	

41		LDA	#\$04
42		PHA	
43		STA	RETFLG
44		JMP	FUNEXIT
45	KEYDLY	LDX	#\$0C
46		BVS	IK1
47		LDX	#\$31
49	IK1	LDY	#0
49	IK2	BVC	IK2A
50		PHP	
51		JSR	SNIFFIRQ
52		PLP	
53	IK2A	INC	RANDL
54		BNE	IK3
55		INC	RANDH
56	IK3	LDA	AUXFLG
57		BMI	KDRETY
58		LDA	KBD
59		BMI	KDRETY
60		DEY	
61		BNE	IK2
62		DEX	
63		BNE	IK1
64		CLC	
65		RTS	
66	KDRETY	SEC	
67	SEV	RTS	

GETKEY, Input Routine

1	GETKEY	LDA	#\$C5
2		PHA	
3		LDA	#\$02
4		PHA	
5		RTS	

APPENDIX D

Auxiliary Keyboard Input Routines

Program Equates

```
1 *****
2 *
3 *
4 * SETUP ALL EQUATES
5 *
6 *
7 *****
8 *
9 *SYSTEM FLAG TO DISPLAY INVERSE VIDEO
10 INVFLG      EQU  $32
11 *
12 *
13 *USED IN THE BINARY TO BCD CONVERSION ROUTINE
14 BCD         EQU  $47D
15 *
16 *
17 *INPUT LINE BUFFER
18 SAVELINE    EQU  $2E0
19 *
20 *
21 *TEMPORARY MEMORY LOCATION
22 TEMP        EQU  $5FD
23 *
24 *
25 *SYSTEM CLEAR SCREEN ROUTINE
26 CLSCR       EQU  $FC58
27 *
28 *
29 *USED BY THE SYSTEM TO HOLD THE CURRENT CURSOR
30 TEMP1       EQU  $478
31 *
32 *
33 *CURSOR HORIZONTAL POSITION
34 CH          EQU  $24
35 *
36 *
37 *BASE ADDRESS LOW OF THE CURRENT LINE
38 BASL        EQU  $28
39 *
40 *
```

```

41 *SWITCH TO TURN OFF EXPANSION ROM
42 ROMSOFF      EQU   $CFFF
43 *
44 *
45 *SYSTEM CHARACTER OUTPUT ROUTINE
46 COUT         EQU   $FDED
47 *
48 *
49 *SOFT-SWITCH TO ENABLE THE CX ROM
50 CXROM        EQU   $C007
51 *
52 *
53 *ACCESS CAUSES THE SPEAKER TO TOGGLE
54 SPEAKER      EQU   $C030
55 *
56 *
57 *TEMPORARY COUNTER
58 COUNTER      EQU   $4FD
59 *
60 *
61 *RETURN FLAG TO INDICATE IF ENTRY WAS FROM KEYIN.B
62 *   $04=ENTRY FROM KEYIN.B
63 RETFLG       EQU   $67D
64 *
65 *
66 *MODE FLAG FROM KEYCLICK
67 *   $80=NO KEYCLICK
68 *   $00=KEYCLICK
69 MODEFLG      EQU   $57D
70 *
71 *
72 *EEPROM ADDRESS
73 POINTE       EQU   $COD1
74 *
75 *
76 *AUX KEYBOARD FLAG
77 *   BIT 7 SET=KEYPRESSED
78 *   BIT 6 SET=POWER UP
79 *   BIT 4-5=SELECTED KEY SET
80 *   BIT 0-3=SELECTED KEY
81 AUXFLG       EQU   $COD2
82 *
83 *
84 *AUX KEYBOARD CHARACTER CODE
85 AUXCHR       EQU   $COD4
86 *
87 *
88 *AUX KEYBOARD STROBE ACCESS RESETS BIT 6 & 7 OF AUXFLG
89 AUXSTRB      EQU   $COD8
90 *

```

```

91 *
92 *RANDOM NUMBER LOCATIONS
93 RANDL      EQU  $4E
94 RANDH      EQU  $4F
95 *
96 *
97 *SYSTEM KEYBOARD FLAG AND DATA
98 *   BIT 7=1;DATA READY
99 KBD        EQU  $C000
100 *
101 *
102 *SYSTEM KEYBOARD STROBE ACCESS RESETS BIT 7 OF KBD
103 KBDSTRB    EQU  $C010
104 *
105 *
106 *SYSTEM WAIT ROUTINE
107 *   DELAY=1/2(26+27A+5A 2)MICROSECONDS
108 WAIT       EQU  $FCA8
109 *****

```

HANDLER-A Input Routine

```

110          ORG  $C500
111          JMP  PROG
112          BIT  AUXFLG
113          BVC  AUX
114          BVS  AROUND
115          NOP
116          DFB  $01
117          DFB  $A0
118          NOP
119 RETURN    PHA
120          LDA  MODEFLG
121          BNE  NOCLICK
122          LDX  #6
123 LOOPX     STA  SPEAKER
124          LDY  #0
125 LOOPY     DEY
126          BNE  LOOPY
127          DEX
128          BNE  LOOPY
129 NOCLICK   LDA  RETFLG
130          CMP  #$04
131          BNE  RET1
132 EXIT      LDA  TEMP1
133          STA  RETFLG
134          LDY  CH
135          STA  (BASL),Y
136 RET1      PLA
137          TAY

```

```

138          PLA
139          TAX
140          TYA
141          PLP
142          BPL    RET2
143          PLP
144          STA    CXROM
145 *
146 * TO INTERNAL CX ROM SPACE.
147 * AN RTS INSTRUCTION IS CONVENIENTLY
148 * LOCATED THERE; ($C53E).
149 *
150 RET2      PLP
151          RTS
152 AROUND    LDA    #0
153          STA    MODEFLG
154          STA    COUNTER
155          STA    AUXSTRB
156          LDA    RETFLG
157          CMP    #$04
158          BNE    AUXKBD
159          LDA    KBD
160          BMI    GETK3
161          LDA    #$8D
162          BMI    EXIT
163 AUXKBD    LDA    AUXFLG
164 AUX        BPL    GETKEY
165          LDA    COUNTER
166          STA    POINTER
167          LDA    AUXCHR
168          PHA
169          INC    COUNTER
170          LDA    COUNTER
171          CMP    #$1F
172          BEQ    OOPS
173          STA    POINTER
174          LDA    AUXCHR
175          BNE    NOCLICK
176          STA    COUNTER
177          JSR    SETSTRB
178          PLA
179          BMI    RETURN
180 OOPS      LDA    #0
181          STA    COUNTER
182          JSR    SETSTRB
183          PLA
184          JMP    RETURN
185 GETKEY    INC    RANDL
186          BNE    GETK2
187          INC    RANDH

```

```

188 GETK2      LDA   KBD
189           BPL   AUXKBD
190 GETK3      STA   KBDSTRB
191           JMP   RETURN
192 *
193 * SUBROUTINE SETSTRB
194 *
195 * TO CLEAR AUX KEYBOARD STROBE
196 *
197 SETSTRB     LDA   AUXSTRB
198           LDA   #$80
199           JSR   WAIT
200           LDA   AUXFLG
201           BMI   SETSTRB
202           LDA   #$80
203           JSR   WAIT
204           LDA   AUXFLG
205           BMI   SETSTRB
206           RTS
207 PROG       STA   ROMSOFF
208           JMP   PROGRAM
209 OUTCR      LDA   #$8D
210 OUTCH      JSR   COUT
211           STA   ROMSOFF
212           RTS
213 CLEAR      JSR   CLSCR
214           STA   ROMSOFF
215           RTS

```

HANDLER-B Programming Software

```

216           ORG   $C900
217 PROGRAM    JSR   START
218           JSR   INCH
219           JSR   OUTCH
220           CMP   #$D9           ;=Y?
221           BEQ   REPROG
222           CMP   #$F9
223           BEQ   REPROG
224 CLICK     JSR   OUTCR
225           LDY   #156
226           JSR   OUTPUT         ;KEYCLICK?
227           JSR   INCH
228           JSR   OUTCH
229           CMP   #$D9           ;=Y?
230           BEQ   Y
231           CMP   #$F9
232           BNE   OFF
233 Y         LDA   #0
234           STA   MODEFLG

```

```

235      BEQ    QUIT
236  OFF    LDA    #$80
237      STA    MODEFLG
238  QUIT   JSR    SETSTRB
239      RTS
240  REPROG JSR    OUTCR
241      JSR    OUTCR
242      LDY    #98
243      JSR    OUTPUT      ;END WITH CNTRL-2
244      JSR    OUTCR
245      LDA    #0
246      STA    TEMP
247  MORE   JSR    INCH
248      CMP    #$80
249      BEQ    STOP
250      LDX    TEMP
251      STA    SAVELINE,X
252      JSR    OUT2
253      INC    TEMP
254      LDA    TEMP
255      CMP    #32
256      BNE    MORE
257      DEC    TEMP
258      BEQ    STOP2
259  STOP   LDA    #0
260      STA    SAVELINE,X
261  STOP2  JSR    OUTCR
262      JSR    OUTCR
263      LDY    #122
264      JSR    OUTPUT      ;OK?
265      JSR    INCH
266      JSR    OUTCH
267      CMP    #$D9      ;=Y?
268      BEQ    STORE
269      BNE    PROGRAM
270      CMP    #$F9
271      BEQ    STORE
272      JMP    PROGRAM
273  *****
274  STORE  LDX    TEMP
275  AGAIN  STX    POINTER
276      LDA    SAVELINE,X
277      STA    AUXCHR
278      LDA    #80
279      JSR    WAIT
280      LDA    SAVELINE,X
281      CMP    AUXCHR
282      BEQ    OK
283      STA    AUXCHR
284      LDA    #$FF

```

```

285          JSR    WAIT
286          LDA    SAVELINE,X
287          CMP    AUXCHR
288          BEQ    OK
289          JSR    OUTCR
290          LDY    #175
291          JSR    OUTPUT
292          JMP    QUIT
293 OK        DEX
294          BPL    AGAIN
295          JMP    CLICK
296 *****
297 START    JSR    CLEAR
298          LDY    #0
299          JSR    OUTPUT          ;AUX KEYBOARD PROGRAMMING
300          JSR    OUTCR
301          JSR    OUTCR
302          LDY    #135
303          JSR    OUTPUT          ;SELECT A KEY
304 CKAUX    LDA    AUXFLG
305          BPL    CKAUX
306 SETUP    JSR    CLEAR
307          LDY    #0
308          JSR    OUTPUT          ;AUX KEYBOARD PROGRAMMING
309          JSR    OUTCR
310          JSR    OUTCR
311          LDY    #35
312          JSR    OUTPUT          ;BOARD #
313          LDA    AUXFLG
314          PHA
315          AND    #$30
316          LSR    A
317          LSR    A
318          LSR    A
319          LSR    A
320          CLC
321          ADC    #01
322          ORA    #$B0
323          JSR    OUTCH
324          LDY    #43
325          JSR    OUTPUT          ;KEY #
326          SED
327          LDA    #0
328          STA    BCD
329          LDX    #$08
330          PLA
331          AND    #$0F
332 BR1      ASL    A
333          TAY
334          LDA    BCD

```

```

335          ADC    BCD
336          STA    BCD
337          TYA
338          DEX
339          BNE     BR1
340          LDA     BCD
341          CLC
342          ADC     #01
343          STA     BCD
344          CLD
345          AND     #$F0
346          LSR     A
347          LSR     A
348          LSR     A
349          LSR     A
350          ORA     #$B0
351          JSR     OUTCH
352          LDA     BCD
353          AND     #$0F
354          ORA     #$B0
355          JSR     OUTCH
356          JSR     OUTCR
357          JSR     OUTCR
358          LDY     #54
359          JSR     OUTPUT          ;CURRENT PROGRAM
360          LDA     #0
361          STA     COUNTER
362          STA     POINTER
363          LDA     AUXCHR
364  ANOTHER    JSR     OUT2
365          INC     COUNTER
366          LDA     COUNTER
367          CMP     #$20
368          BEQ     OOPS2
369          STA     POINTER
370          LDA     AUXCHR
371          BNE     ANOTHER
372  OOPS2      LDA     #0
373          STA     COUNTER
374          JSR     OUTCR
375          JSR     OUTCR
376          LDY     #78'
377          JSR     OUTPUT          ;REPROGRAM (Y OR N)?
378          RTS
379  *****
380  INCH        LDA     #$AA
381          STA     TEMP1
382  KEYIN2      LDA     #$FF
383          LDY     CH
384          STA     (BASL),Y

```

```

385          JSR  KEYDLY
386          BCS  GOTKEY
387          LDA  TEMP1
388          LDY  CH
389          STA  (BASL),Y
390          JSR  KEYDLY
391          BCC  KEYIN2
392 GOTKEY    STA  KBDSTRB
393          RTS
394 KEYDLY    LDX  #$31
395 IK1       LDY  #0
396 IK2       LDA  KBD
397          BMI  KDRETY
398          DEY
399          BNE  IK2
400          DEX
401          BNE  IK1
402          CLC
403          RTS
404 KDRETY    SEC
405          RTS
406 *****
407 OUTPUT    LDA  LIST,Y
408          BMI  LAST
409          ORA  #$80
410          JSR  OUTCH
411          INY
412          BNE  OUTPUT
413 LAST      CMP  #$A3
414          BEQ  LAST2
415          JSR  OUTCH
416          JSR  OUTCR
417 LAST2     RTS
418 *****
419 OUT2      PHA
420          AND  #$7F
421          SEC
422          SBC  #$20
423          BMI  FIX
424          PLA
425          JSR  OUTCH
426          RTS
427 FIX      LDY  #$3F
428          STY  INVFLG
429          PLA
430          AND  #$7F
431          TAX
432          LDA  TABLE,X
433          JSR  OUTCH
434          LDY  #$FF

```

```

435          STY  INVFLG
436          RTS
437  TABLE    ASC  *2ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]6_*
438  *****
439  LIST      DCI  *AUXILIARY KEYBOARD PROGRAMMING*
440          DCI  *BOARD ##*
441          DCI  *KEY ##*
442          DCI  *- - CURRENT PROGRAM - -*
443          DCI  *REPROGRAM (Y OR N)?##*
444          DCI  *END INPUT WITH A CNTRL-2*
445          DCI  *OK (Y OR N)?##*
446          DCI  *- -SELECT A KEY- -*
447          DCI  *KEYCLICK (Y OR N)?##*
448          DCI  *ERROR IN PROGRAMMING EEPROM*

```

APPENDIX E

EEPROM Utility Program

```

1 *****
2 *
3 * TO MOVE DATA TO EEPROM
4 *
5 *****
6 MONITOR      EQU    $FF69      ;MONITOR STARTING ADDRESS
7 A1L          EQU    $EA        ;TEMPORARY
8 A1H          EQU    $EB        ;ZERO
9 A4L          EQU    $EC        ;PAGE
10 A4H         EQU    $ED        ;LOCATIONS
11 WAIT        EQU    $FCA8      ;DELAY ROUTINE
12             ORG     $300
13             JMP     BEGIN
14 STARTL      DFB     $00        ;SOURCE STARTING
                                ADDRESS,LOW BYTE
15 STARTH      DFB     $20        ;SOURCE STARTING
                                ADDRESS;HIGH BYTE
16 ENDL        DFB     $F        ;SOURCE ENDING
                                ADDRESS,LOW BYTE
17 ENDH        DFB     $20        ;SOURCE ENDING ADDRESS,
                                HIGH BYTE
18 DESL        DFB     $00        ;DESTINATION ADDRESS,
                                LOW BYTE
19 DESH        DFB     $C5        ;DESTINATION ADDRESS,
                                HIGH BYTE
20 BEGIN       LDA     STARTL     ;STORE SOURCE
21             STA     A1L        ;STARTING ADDRESS
22             LDA     STARTH     ;IN ZERO
23             STA     A1H        ;PAGE LOCATIONS
24             LDA     DESL       ;STORE DESTINATION
25             STA     A4L        ;STARTING ADDRESS
26             LDA     DESH       ;IN ZERO
27             STA     A4H        ;PAGE LOCATIONS
28 LOOP        LDY     #0         ;CLEAR Y REG
29             LDA     (A1L),Y    ;GET A BYTE
30             STA     (A4L),Y    ;TRANSFER IT
31             LDY     #128       ;FOR DELAY ROUTINE,SET FOR
                                11MS DELAY
32             JSR     WAIT       ;DELAY FOR EEPROM WRITETIME
33             LDY     #0
34             LDA     (A4L),Y
35             CMP     (A1L),Y

```

36		BEQ	OK	
37		LDA	(A1L),Y	
38		STA	(A4L),Y	
39		LDY	#255	
40		JSR	WAIT	
41		JSR	WAIT	
42		INC	ERR1	
43		LDY	#0	
44		LDA	(A4L),Y	
45		CMP	(A1L),Y	
46		BEQ	OK	
47		INC	ERR2	
48	OK	INC	A4L	; INCREMENT
49		BNE	AROUND	; DESTINATION
50		INC	A4H	; ADDRESS
51	AROUND	LDA	A1L	; CHECK AND
52		CMP	ENDL	; SEE IF
53		BNE	GOROUND	; IT IS
54		LDA	A1H	; THE LAST
55		CMP	ENDH	; BYTE
56		BEQ	QUIT	; IF SO, THEN QUIT
57	GOROUND	INC	A1L	; INCREMENT
58		BNE	ROUND	; SOURCE
59		INC	A1H	; ADDRESS
60	ROUND	BNE	LOOP	; ALWAYS TAKEN
61	ERR1	DFB	\$0	
62	ERR2	DFB	\$0	
63	QUIT	JMP	MONITOR	

APPENDIX F

OPERATIONAL INSTRUCTIONS

Parts of the System

The auxiliary keyboard system is made of three basic hardware parts (Reference Figure 4 on page 15 and Figure 1 on page 5). First, the circuit card contains various electronic components and two card-edge connectors. One connector mates to a 50-pin expansion slot connector on the Apple //e motherboard. The other connector mates to a 26-pin card edge cable connector from the auxiliary keyboard. The second hardware part is the auxiliary keyboard with seventeen momentary pushbutton switches and four indicators mounted in a slopped front metal enclosure. A 26-wire flat cable is permanently attached inside the auxiliary keyboard and leads out to connect to the auxiliary keyboard circuit card. The third part of the system is a 2764 EPROM chip that replaces the CX ROM chip on the Apple //e motherboard. The 2764 EPROM includes the software patches to the standard Apple input routines that recognize the auxiliary keyboard. If the 2764 EPROM is installed, the auxiliary keyboard circuit card must also be

installed for the Apple to operate properly. This is required because the patched input routines in the 2764 EPROM jump to the input routines in the 2716 EPROM on the auxiliary keyboard circuit card.

Hardware Installation

1. The auxiliary keyboard circuit card was designed to be installed in the Apple //e expansion slot number 5.
2. The flat cable from the auxiliary keyboard is routed through an opening at the rear of the Apple computer and plugged into the auxiliary keyboard circuit card.
3. The CX ROM chip from the Apple //e motherboard is removed and the patched CX ROM chip is installed in its place.

The CX ROM is identified on the Apple //e motherboard as CD ROM and the part number is 342-D135. This ROM is referred to as the CX ROM in the Apple literature but is labeled CD ROM on the motherboard.

Operation

The auxiliary keyboard is completely operational when the system power is turned on. No commands or other switches are necessary to initialize the auxiliary keyboard. When a key is pressed on the auxiliary keyboard,

the previously stored key sequence will be sent to the computer. The program or operating system currently being executed, such as Applesoft basic or the Wordstar Wordprocessing Program, responds to the input sequence as it had been input manually.

To program a key with a new key sequence a CALL 50432 from Applesoft Basic or the command \$C500G from the monitor is issued. The Apple screen is cleared and a message is displayed requesting the user to select the key to be programmed. Once a key is pressed the selected key number and key set number are displayed. The auxiliary keyboard has sixteen keys installed in a 4 x 4 matrix on the face of the auxiliary keyboard. Sixty-four different key sequences can be stored into the auxiliary keyboard. Therefore, the sixteen keys on the auxiliary keyboard are utilized as four sets (or boards) of sixteen keys each. A seventeenth key installed on the keyboard above the 4 x 4 matrix selects the active board. This seventeenth key is termed SELECT and is driven by the board select logic on the auxiliary keyboard circuit card (Reference Chapter IV and the section entitled Board Select Logic). Four LED's (Light Emitting Diodes) are used to indicate the active board. To select a different board the user momentarily presses the SELECT key. Each time the SELECT key is pressed the illuminated LED is sequenced to the next LED

indicating that a different board is active.

After the key number and board number are displayed the ASCII character sequence of the selected key and the question "Do you want to reprogram? (Y or N)" are displayed. If a "Y" is input from the system keyboard, the program responds with a prompt and the note "End input with a Cntrl-2". At this time the key sequence to be retained is entered.

The sequence can contain any character code that can be generated by the system keyboard except a Cntrl-2. Cntrl-2 is the null (\$00) character code. When thirty-two character codes are entered or the input sequence is ended by typing a Cntrl-2, the system will respond with the question "OK (Y or N)?". If the typed response is a "Y", the input key sequence will be stored in the key sequence memory for the selected key. If an "N" is typed, the auxiliary keyboard programming sequence is started at the beginning requesting the user to select a key to be programmed. If control codes are entered in the input key sequence, they are displayed in reverse video on the Apple screen. For example, the RETURN key generates the CR (carriage return) control code. When the RETURN key is typed while the system is accepting the input key sequence, an "M" is displayed in reverse video. Table 3 on page 65 is a list of control codes, the keys required to be typed

on the Apple system keyboard and the character displayed by the auxiliary keyboard programming software.

Once the new key sequence is entered and stored or the user has selected not to reprogram the current key sequence, the user is prompted with the question "Keyclick (Y or N)?". Keyclick is the addition of an audible click each time a key is pressed on the auxiliary keyboard or the main system keyboard. Some users like a audible feedback each time a key is pressed and some users do not. Hence, the option "Keyclick (Y or N)?" is offered. After entering the response to the keyclick question, a return to basic or the monitor is executed by the auxiliary keyboard programming software.

VITA

Larry Dwight Wilhite was born in Bolivar, Tennessee on June 21, 1954. He attended elementary schools in that city and graduated from Bolivar Central High School in 1972. In 1975 he transferred from The University of Tennessee at Martin to Auburn University, Auburn, Alabama, and in 1977 received a Bachelor of Science degree in Electrical Engineering. Upon graduation, the author was employed by the Federal Aviation Administration, Atlanta, Georgia. In October 1978, he accepted employment with Sverdrup Technology, Inc., Arnold Engineering Development Center, Arnold Air Force Station, Tennessee, as a controls engineer. He enrolled as a part-time student in the graduate program at The University of Tennessee Space Institute in January 1978. In March 1985, he received the Master of Science degree in Electrical Engineering.

Mr. Wilhite now resides in Winchester, Tennessee, with his wife, Teresa, and two children, Jake and Haley.