AIM 65

MICROCOMPUTER

USER'S GUIDE

Rockwell International

$5.00
Document No. 29650 N36
Rev. 2, March 1979

NOTICE

Rockwell International reserves the right to change this product and its specifications at any time without notice to improve its design or performance.

No responsibility is assumed by Rockwell International for use of this information; nor for any infringement of patents or other rights of third parties which may result from the use of this information.

TABLE OF CONTENTS

SECTION 7. AIM 65 SYSTEM DESCRIPTION

SECTION 8. THE R6522 VERSATILE INTERFACE ADAPTER

SECTION 9. INTERFACING WITH AUDIO CASSETTE RECORDERS AND TELETYPE

SECTION 1
INTRODUCTION AND STARTUP

Welcome to the Rockwell R6500 Advanced Interactive Microcomputer--the AIM 65. AIM 65 is a complete general purpose microcomputer featuring advanced hardware and software. Rockwell has designed AIM 65 to be the ideal introduction to the rapidly expanding world of microprocessing. Not only is AIM 65 a learning tool, it is a powerful dedicated microcomputer that can serve as a central processor, or controller/monitor. AIM 65 is also an excellent, low-cost microcomputer development system.

AIM 65 comes to you fully assembled, tested, and warranted by Rockwell. It is a simple task to unpack it, connect the two modules, attach the required power--and be ready to go. You will quickly discover how easy the AIM 65 is to use and understand. If you are anxious to get started, go directly to Section 1.4. We suggest, however, that you read the initial sections to gain an overall understanding of AIM 65, this manual, and other documentation. We wish you many satisfying and rewarding hours of AIM 65 operation.

1.1 AIM 65 OVERVIEW

AIM 65 consists of two modules--the Master Module and the Keyboard Module--interconnected by a short plug-in ribbon cable. The Master Module holds a printer, a display, and the microcomputer components. Figure 1-1 identifies the

peripherals and microcomputer devices and shows where they are located.

The R6502 Central Processing Unit (CPU) is the heart of the AIM 65. The R6502 is a very widely used and powerful 8-bit microprocessor. It operates at 1 MHz on AIM 65 to provide a minimum instruction execution time of two micro-seconds. With 56 instructions and 13 addressing modes, the R6502 is highly flexible, but easy to program. The R6502 can address 4K bytes of RAM and 20K bytes of ROM on the Master Module plus an additional 40K bytes of user provided external RAM, ROM, or I/O.

The other R6500 devices on the AIM 65 include the R6522 Versatile Interface Adapter (VIA), the R6532 RAM-Input/ Output Timer (RIOT), the R6520 Peripheral Interface Adapter (PIA), the R2332 Read Only Memory (ROM), and the R2114 Read/Write Random Access Memory (RAM).

The 20 column thermal printer provides a permanent record of user commands, data, and programs as well as AIM 65 status, prompts, and messages. Printing at 120 lines per minute, the 5 X 7 dot matrix printer provides rapid, quiet, and reliable operation. It can print the complete ASCII 64 character format.

The display provides visual feedback during keyboard operations. The display length of 20 characters allows complete correspondence with the printer. The display uses a 16-segment font to provide a full 64 character ASCII set. The high contrast characters are distinct and easy to read.

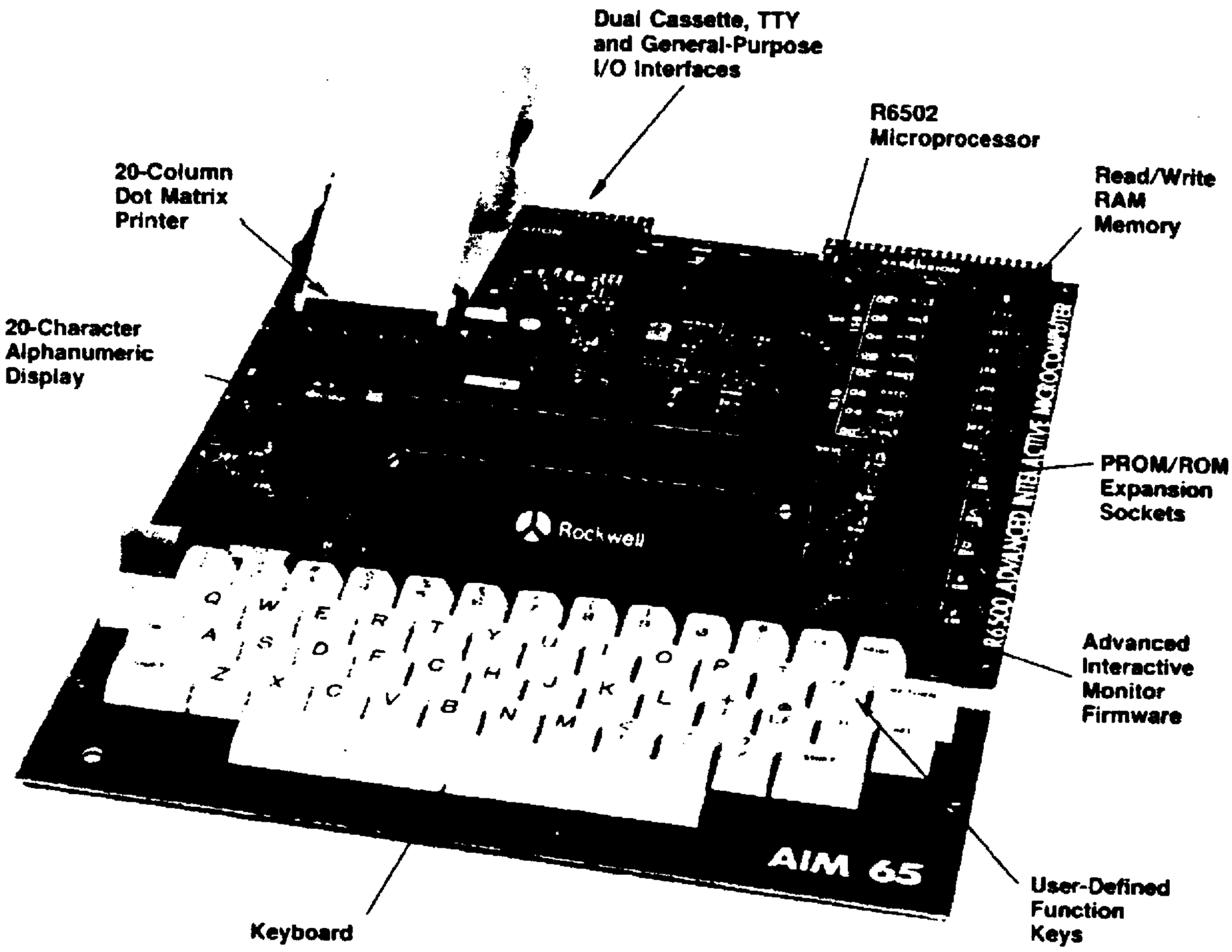


Figure 1-1. The AIM 65 Microcomputer

The Keyboard Module contains a 54-key full-size keyboard. The keyboard has 70 functions (26 alphabetic, 10 numeric, 22 special, 9 control, and 3 user-defined) used by AIM 65.

AIM 65 operation is controlled by a ROM-resident 8K Monitor which provides a comprehensive set of easy-to-use single keystroke commands. The Monitor commands greatly simplify the use of the CPU, memory, and I/O devices. The user can concentrate on application design and microprocessor software development at the functional level. By automatically translating functional commands to machine code, the Monitor makes development faster and more efficient.

The AIM 65 Monitor includes commands to:

Enter R6502 instructions in mnemonic form for direct translation to object code.

Disassemble R6502 instructions from object code to mnemonic form.

Execute user written programs with debugging aids such as instruction trace, register trace, and breakpoints.

Display and alter memory and registers.

Transfer object code to and from one or two audio cassette recorders or a teletype.

Allow user defined functions to interface with user provided peripherals.

An Editor allows easy entry, editing, and listing of R6502 source instructions, data, or general text. The Editor incorporates commands to:

Transfer text, enter programs or data into memory, or transfer source code to and from one or two audio cassette recorders or a teletypewriter.

Locate and change character strings.

Move the text line pointer.

List selectable lines on output devices.

The Master Module has three spare ROM sockets to allow the addition of 12K bytes of optional AIM 65 programs or user developed programs.

An optional R6502 Assembler, resident in a 4K R2332 ROM, may be installed in one of the spare ROM sockets. This two-pass program assembler converts R6502 source instructions into object code using symbolic labels and operands. Extensive error checking in an errors-only pass allows quick checking for proper instruction coding. By using the Assembler and Editor, the user can rapidly code, edit, assemble and debug programs.

The remaining spare ROM sockets may be used to add the optional AIM 65 8K BASIC Interpreter. This universal high level language is easy to learn and use. A fine, self-teaching introduction is Albrecht, Finkel, and Brown, BASIC for Home Computers, Wiley, New York, 1978.

AIM 65 is available in 1K and 4K RAM versions. In the 1K RAM version, the Master Module has six vacant RAM sockets for on-board expansion. The user can then add R2114 RAM chips.

More RAM, ROM, PROM, I/O, or other peripheral chips can be added by extending the AIM 65 address, data, and control bus lines to an expansion connector. The user may easily interface with these lines to meet specific requirements.

AIM 65 can be directly interfaced to external equipment through the application connector and the user dedicated R6522 VIA. The user R6522 has 16 bi-directional input/ output lines, four control lines, and two timers.

Interfaces for one or two low-cost audio cassette recorders allow permanent storing and loading of user programs and data. Recorder control lines enable semi-automatic remote control of the recorders. The optional Assembler allows the user to input source code from one audio cassette recorder and output object code to another recorder. This technique allows the on-board RAM to be dedicated to symbol tables during assembly.

1.2 AIM 65 USER MANUAL DESCRIPTION

This manual is designed to get you quickly into AIM 65 operation and then to tell you how to use it to its full potential. An in-depth description of the AIM 65 hardware and software is provided after the operating procedures.

The appendices contain common reference information.

Section 1, Introduction, explains how to properly set up AIM 65. The user should follow the instructions in Sections 1.4 through 1.8.

Section 2, Introduction to AIM 65 Operation, describes how to perform simple tasks with the AIM 65.

Section 3, AIM 65 Monitor, thoroughly describes each AIM 65 Monitor command and defines all options and AIM 65 prompts and messages.

Section 4, AIM 65 Editor, describes the AIM 65 Editor commands and explains the use of the text buffer.

Section 5, AIM 65 Assembler, provides a description of the assembly process, symbol table usage, assembler commands and options, assembler error detection capability, and error messages.

Section 6, R6500 Programming Concepts, offers an overview of R6500 assembly language programming.

Section 7, AIM 65 System Description, describes AIM 65 hardware and software. The interfaces with the keyboard, printer, and display are defined. The AIM 65 memory map is described and the AIM 65 software structure is shown. User available subroutines are identified along with the calling procedures and register utilization.

Section 8, R6522 Versatile Interface Adapter, describes the capabilities of the user R6522 and provides programming examples.

Section 9, Interfacing with Audio Cassette Recorders and Teletype, explains the interfaces with these user-provided peripherals. This section also describes audio tape recording formats, teletype connections and procedures, details on modifying user alterable variables in order to optimize your recorder interface, and procedures to input and output source and object code using a teletype paper tape punch and reader.

Section 10, Expanding the System, offers guidelines for connecting additional RAM, ROM, I/O or other peripheral devices to the AIM 65, using the expansion connector.

Section 11, Troubleshooting and Warranty, helps you isolate and correct certain problems. Hopefully, any problems are due to incomplete power or interface connections or improper switch positions that can be easily corrected. Should any uncorrectable problems occur, follow the instructions listed in this section for repair.

Appendices A through K offer detailed information for general or specific use that you may want to refer to often. Scan the appendices to become familiar with their content.

An assembly listing of the AIM 65 Monitor and Editor is provided in a separate volume, 29650N36L. This listing offers insight into the structure and design of a complete microcomputer program. Design techniques and algorithms included in the AIM 65 Monitor and Editor may be used in your own applications.

1.3 DESCRIPTION OF OTHER AIM 65 DOCUMENTATION

This manual does not describe either R6500 hardware or software design in detail. The R6500 Microcomputer System Hardware Manual describes the architecture, electrical and interface characteristics, and timing and other hardware considerations of all the R6500 devices used in the AIM 65. The R6500 Microcomputer System Programming Manual describes how each instruction operates in the R6502 CPU.

An R6500 Programming Reference Card and an AIM 65 Summary Card are included for handy reference during AIM 65 operation. A fold-out schematic poster of the complete AIM 65 is also enclosed.

1.4 HANDLING PRECAUTIONS

You should observe the following precautions to prevent damage to AIM 65 or interfacing equipment.

CAUTION - UNENCLOSED MODULES

Since AIM 65 has no protective enclosure, items dropped or set on the module may damage the printer, display, or other components. Liquid spilled on the modules may also damage the modules by inducing short circuits.

CAUTION - MOS DEVICES

Microcomputer devices are manufactured using the Metal-Oxide Semiconductor (MOS) process. The inadvertent application of high voltages may damage MOS devices.

You should take the following precautions:

A. Discharge any static electrical charge accumulated on your body by touching a ground connection (e.g., a grounded equipment chassis) before touching the AIM 65. This precaution is especially important if you are working in a carpeted area or in an environment with low relative humidity.

B. Make sure all test equipment, interfacing hardware, and electrical tools (e.g., soldering irons) are properly grounded before use with AIM 65.

CAUTION - EXPOSED VOLTAGES

The +5V and +24V supply voltages are routed to many exposed pins on the modules. Shorting these pins to ground or to other pins may cause improper operation or permanent damage.

WARNING - PROTRUDING LEADS

The bottom of the Master Module and Keyboard Module have component leads sticking through the mounting holes that may protrude past the solder cap. These clipped leads may be sharp and could puncture skin. To avoid injury, handle the modules by placing fingers between the component mounting holes.

1.5 PARTS INVENTORY

Report any damage to the shipping container to your dealer or shipping agent.

You may wish to save the shipping container and packaging material should you need to ship or store the AIM 65 at some future date.

After carefully removing the AIM 65 and accompanying documentation and loose equipment from the shipping container, locate the following items:

4 Manuals - AIM 65 User's Guide
- AIM 65 Monitor Program Listing
- R6500 Microcomputer System Programming Manual
- R6500 Microcomputer System Hardware Manual

1 R6500 Programming Reference Card
1 AIM 65 Summary Card
1 Loose Equipment Packet with:
1 Keyboard to Master Module Cable
15 Rubber Pads
1 Roll of Printer Paper
1 Warranty Card*

*Be sure to complete and mail the warranty card.

1.6 SET UP

SUPPORTING PAD INSTALLATION

Remove any conductive foam from underneath the Master Module. Attach the supporting rubber pads on the bottom of

the Master and Keyboard Modules at the approximate locations shown in Figure 1-2. First remove the protective film from the pad's sticky surface; next, lightly attach each pad at the proper location. Turn the boards top side up and set them down on a flat surface. Press down firmly, and carefully, on the modules over the pad locations to permanently affix the pads.

SOCKETED COMPONENTS CHECK

Inspect the socketed components on the Master Module. If any socketed devices have loosened during shipment, reseat them by firmly and evenly pressing down on the top of the device with one hand while supporting the Master Module under the loosened device with the other hand in order to prevent flexing.

KEYBOARD CONNECTION

Ensure that the pins on the Keyboard to Master Module cable are straight and positioned properly. Connect one end of the cable into Keyboard Module Connector J1 and the other end into the Master Module Connector J4.

CAUTION

The keyboard to Master Module Cable allows limited movement and positioning of the Keyboard Module with respect to the Master Module. Extreme relocation will cause the cable to pull out from one or both connectors and may damage the cable connector pins.

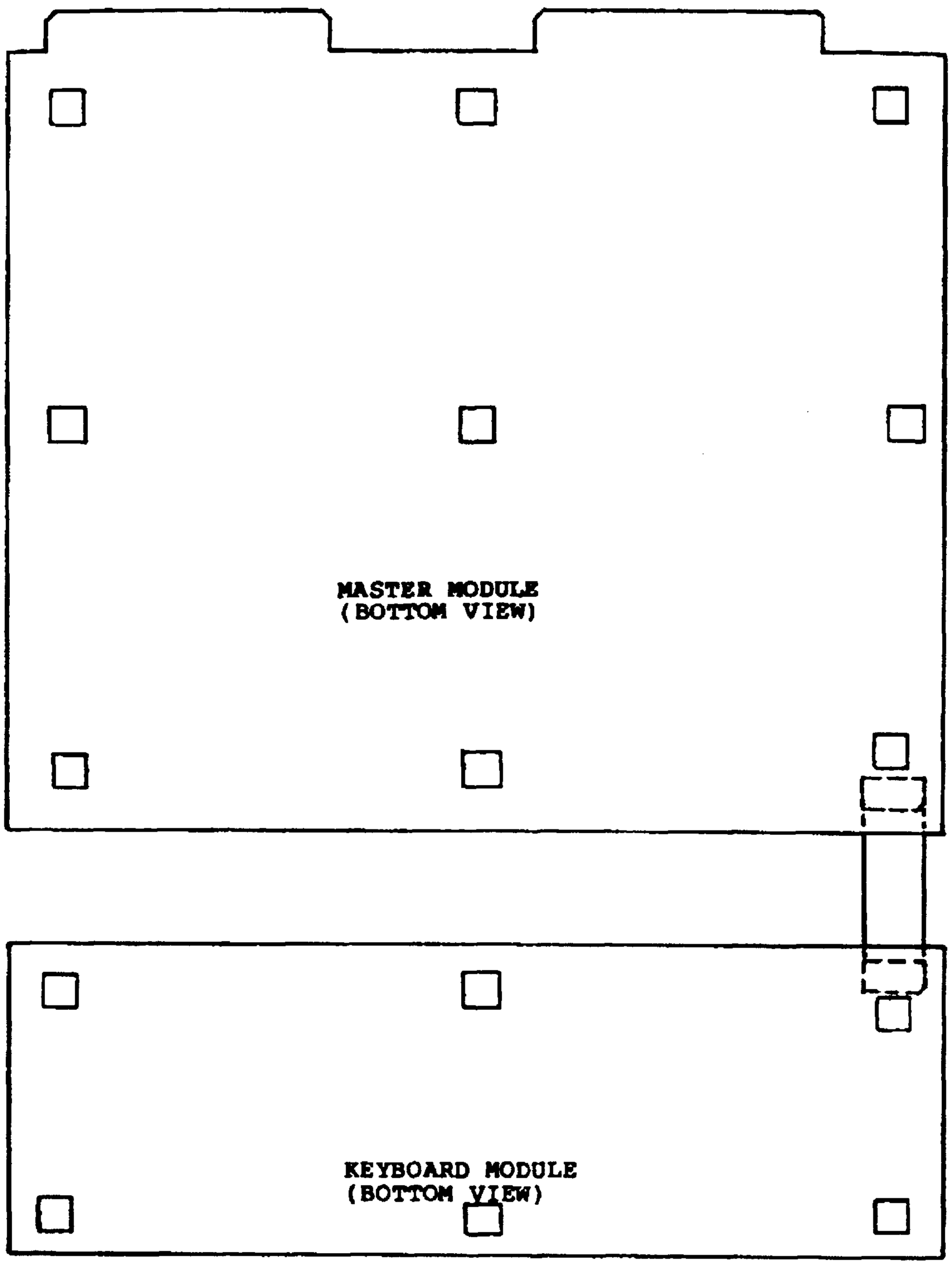


Figure 1-2. Supporting Pad Installation

A longer cable (up to three feet) may be used in place of the provided cable without affecting AIM 65 performance.

PAPER INSTALLATION

Separate the start of the printer paper from the supplied roll. Tear or cut the paper evenly, being careful to remove any adhesive or foreign material from the paper. Slide the roll of paper onto the wire paper holder. The paper should feed from under the roll toward the printer.

Pull the printer head release lever toward the keyboard edge of the Master Module to release the printer thermal head from the platen. Insert the paper into the back of the printer under the platen until it can be grasped from above, then feed the paper under the tear bar. Pull the paper up slightly until the entire leading edge is past the tear bar edge. Push the lever on the top of the printer toward the connector edge of the Master Module to position the printer thermal head on the platen.

CAUTION

Any adhesive or foreign material that comes in contact with the printer thermal elements may damage the printer.

1.7 POWER SUPPLY REQUIREMENTS

AIM 65 requires only two voltages to operate: +5V and +24V. The +5V supplies power to the microcomputer devices, the audio circuitry, and the TTY circuitry.

The +24V supplies power to the printer. With only the +5V supplied, AIM 65 will operate properly, but both the printer and the TTY interface will not work. AIM 65 will display PRINTER DOWN if an attempt is made to print without the +24V available.

The +5V requirements are:

- +5V + 5% (4.75 to 5.25V)
- Regulated
- 2.0A

The +5V current may vary from less than 1.0A, for a 1K AIM 65 with two ROMs installed and the display segments unilluminated, to greater than 1.5A for a 4K AIM 65 with five ROMs installed and all display segments illuminated.

The +24V requirements are:

- +24V + 15% (20.4 to 27.6V)
- Unregulated
- 2.5A peak
- 0.5A average

The +24V current may vary from less than 0.2A, when the printer is not activated, to greater than 2.0A during a

print cycle. Note that the peak current is of short duration and therefore may not appear this high when monitoring with a slow response meter.

1.8 POWER SUPPLY CONNECTION AND TURN-ON

Figure 1-3 shows the power supply connections. Do not hook-up and turn on power yet. Follow the procedure below to minimize the possibility of damaging your AIM 65 and power supply:

Step

1. Connect the +5V RETURN to TB1-2 (GND) and the +5V to TB1-3 (+5V). Recheck the connections.

2. Leave the +24V lines disconnected. Ensure that the +24V power line is touching neither the AIM 65 nor the +24V RETURN line.

3. Switch the KB/TTY switch to the KB position.

4. Switch the RUN/STEP switch to the RUN position.

5. Turn on +5V, or both the +5V and +24V (but leave the +24V supply disconnected) if they are supplied by the same power supply.

6. Verify that ROCKWELL AIM 65 flashes on the display followed by a steady display of PRINTER DOWN. If you are not sure, depress the RESET button and you should see it again. If the display is proper, go to Step 7.

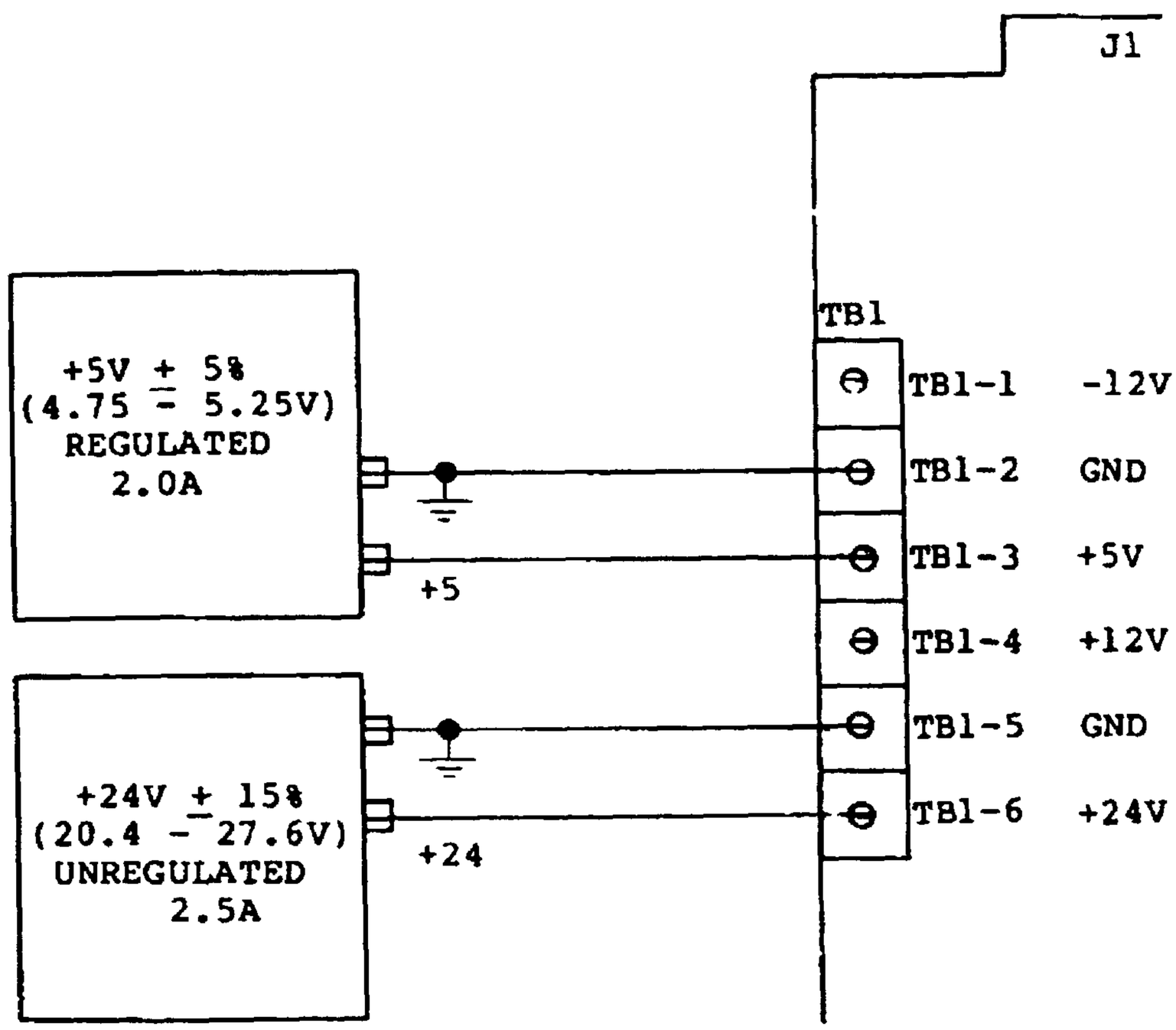


Figure 1-3. Power Supply Connections

If there is no display, the +5V power lines are probably incorrectly connected. Turn off the +5V power supply and repeat Steps 1 through 6. If the display still does not appear, refer to the troubleshooting procedure in Section 11.

7. Turn off the +5V power supply.

8. Connect the +24V RETURN to TB1-5 (GND) and the +24V to TB1-6 (+24V). Recheck the connections.

NOTE

Ground terminals TB1-2 and TB1-5 are connected on the Master Module. The +5V RETURN and the +24V RETURN may be connected together on either terminal if it is more convenient. If one power supply is used to supply both +5V and +24V, only one common RETURN is required, and may be connected to either TB1-2 or TB1-5.

9. Turn on the +5V and +24V power supplies. If they are separate, turn on the +5V supply first. If +24V is turned on first, the printer paper will continuously advance until +5V is applied. If +5V is turned on first, the PRINTER DOWN display will occur.

10. After both +5V and +24V are applied, press the RESET button. The ROCKWELL AIM 65 message will display and print followed by a display of < in the leftmost digit.

NOTE

The printer may have been inadvertently turned off during the AIM 65 turn-on process. Type PRINT to print the contents of the display regardless of the printer control. Type CTRL and PRINT simultaneously until <ON is displayed to turn the printer on. Then type the R key, which will display and print the register headings and contents (see Section 3.3.7).

If the printer does not operate, the +24V power lines are probably incorrectly connected; turn off the power supplies and repeat Steps 8, 9, and 10. If the display and printout still do not appear properly, refer to the troubleshooting procedure in Section 11.

11. The AIM 65 is now operative. Section 2 describes the basic concepts of using the AIM 65.

1.9 CONTROL SWITCHES

1.9.1 RESET Button

Depression of the RESET button will cause a hardware and software reset to be performed. All input/output devices, i.e., 6520, 6532, and 6522 (including the user 6522) and the 6502 CPU will be initialized to their reset state. Refer to the individual device description for the definition of the hardware reset operations.

When the RESET button is depressed, the current operation is interrupted and the AIM 65 Monitor initialization performed.

The Monitor checks to see if a "cold" reset or a "warm" reset is to be accomplished. A "cold" reset, or power-on initialization, will be performed if the Monitor determines that power has been interrupted. A "cold" reset causes all Monitor control parameters to be initialized including user alterable parameters. A "warm" reset performs initialization of only the Monitor control variables and does not initialize user alterable parameters. Refer to Section 7.6 for a description of the user alterable parameters.

A "warm" reset can be performed at any time by depressing the RESET button. This type of reset should be performed any time an unknown operation has taken place or if the AIM 65 appears lost or hung up in execution of a command. The AIM 65 Monitor control parameters can easily be altered if an unvalidated user program is executed using the Monitor User Function (F1, F2, or F3) or the Start Execution at Program Counter Address (G) commands. This will cause undetermined and probably improper AIM 65 Monitor operation if a reset is not performed. In this case, a reset is the only way to return complete control to the AIM 65 Monitor.

Some AIM 65 functions perform time critical operations such as reading or writing an audio tape file that does not have time for ESC key monitoring. In these cases, press RESET to abort the command.

A "cold" reset should be performed if it is desired to initialize the user alterable parameters to their default

values. The "cold" reset can be initiated by removing AIM 65 power for a couple of seconds then reapplying power. The power-on reset can also be performed without removing power by placing 00 in address A402 using the M and / Monitor commands then pressing the RESET button (see Section 3.4).

1.9.2 KB/TTY Switch

The position of the KB/TTY switch determines which keyboard controls the AIM 65 operation.

AIM 65 KEYBOARD CONTROL

To operate from the AIM 65 keyboard, place the KB/TTY switch in the KB position. When KB is selected, initial AIM 65 power application or depression of the RESET button will automatically enable inputs from the AIM 65 keyboard.

To switch keyboard control from the TTY when the TTY keyboard is active, place the KB/TTY switch in the KB position and type any key on the TTY keyboard or depress the AIM 65 RESET button.

TTY KEYBOARD CONTROL

To operate from the TTY keyboard, place the KB/TTY switch in the TTY position. If this is the initial transfer of control from the AIM 65 keyboard, or AIM 65 power is applied with the TTY selected, depress the AIM 65 RESET button followed by typing RUBOUT on the TTY keyboard.

If the TTY keyboard has previously been active and control switched over to the AIM 65 keyboard, the TTY keyboard can be reactivated by switching the KB/TTY switch to the TTY position and then typing any AIM 65 key.

Refer to Section 9.2 for a detailed description of TTY connection, and user procedures.

1.9.3 RUN/STEP Switch

The position of the RUN/STEP switch determines whether the user program is to execute in the RUN or single-step mode (see Section 3.6).

1.10 AIM 65 APPLICATIONS

How can you use your AIM 65? Let us take some time to offer some suggestions:

1. AIM 65 is an ideal low cost prototyping and development system for R6502 applications. AIM 65 is portable and easy to use. It provides a self-contained system with keyboard, display, monitor, editor, assembler, and printer.

2. AIM 65 is an ideal educational system for engineering and technology classes. It is low enough in cost so that schools can purchase many work-stations. Yet it offers such advanced features as mnemonic entry, ROM-based assembler and BASIC, and hard copy. Expensive terminals are unnecessary but students can still gain realistic experience.

But AIM 65 is more than just a prototyping or educational system. It is a full-fledged microcomputer with enough peripherals to handle the following applications:

1. Factory data collection terminal
2. Medical instrument controller
3. Navigational calculator
4. Integrated circuit tester
5. Remote instrument controller
6. Data logger
7. Power line monitor
8. Energy monitor
9. Message switching and buffering
10. Engine test controller
11. Alarm logger
12. Automatic Service monitor
13. Biomedical data acquisition system and signal processor
14. Process control supervisor
15. Frequency response analyzer
16. Transformer or motor controller
17. Solar heating controller
18. Security monitor

These are just a few of the AIM 65's potential applications. For others, you might wish to explore the following sources:

The Proceedings of the IEEE's Industrial Electronics and Control Instrumentation Group Annual Conference on "Industrial Applications of Microprocessors". These Proceedings (starting with 1975) are available from IEEE, 345 East 47th Street, New York, NY 10017.

The Proceedings of the IEEE special issues on microprocessor applications (June 1976, February 19780 available from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854.

The monthly section entitled "Digital Control and Auto mation Systems" in Computer Design, available from Computer Design Publishing Corporation, 11 Goldsmith Street, Littleton, MA 01460.

SECTION 2
INTRODUCTION TO AIM 65 OPERATION

This section presents the basic methods of AIM 65 operation. We will describe, in order, how to:

1. Examine the contents of a memory location.

2. Change the contents of a memory location.

3. Enter a machine language program into memory.

4. Enter data into memory.

5. Execute a machine language program.

6. Examine the contents of a register.

7. Change the contents of a register.

8. Use the printer.

9. Store a program or data on a cassette.

10. Load a program or data from a cassette.

Section 3 describes all of these operations in greater detail. Section 3 also discusses other features of the AIM 65 that are used in debugging, in conjunction with the optional assembler and BASIC, and in actual applications.

Let us first discuss how to get started. We assume that you have connected and checked the AIM 65 as discussed in Section 1. Press the RESET button which is located just below and to the left of the printer. You should now be ready to start operating the AIM 65. If you become confused or lost at any time, press RESET. This will return control to the monitor. If you just wish to terminate an operation, type ESC; this key concludes most simple monitor commands.

For the first part of this discussion, start the AIM 65 in the following state:

1. Printer off. You can toggle the printer (i.e. turn it on if it is off and off if it is on) by pressing CTRL and PRINT together. Note that the AIM 65 displays the current state of the printer. Type RETURN after you turn the printer off. We will explain how to use the printer later in this section.

2. KB/TTY switch in the KB (user keyboard) position.

3. RUN/STEP switch in the RUN position.

These switches are located just to the left of the display.

Table 2-1. Hexadecimal to Decimal Conversion

HEXADECIMAL DIGIT	DECIMAL VALUE	BINARY VALUE
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

2.1 EXAMINING MEMORY

You may examine the contents of memory by typing M and entering the hexadecimal address that you want to examine. Table 2-1 contains a list of the hexadecimal digits and their decimal and binary equivalents.

For example, type:

```
M

0

RETURN
```

to display the contents of memory addresses 0, 1, 2, and 3 from left to right. Note that the AIM 65 prompts you to enter an address after you type M.

Now type a space. The AIM 65 responds by displaying the contents of the next four memory addresses. Note that the AIM always displays the starting address to the left of the data. You can continue typing spaces and examining memory, four locations at a time.

Note the following features of the memory examination:

1. Memory addresses are 4 hexadecimal digits long (16 bits) while the contents of a memory location are 2 hexadecimal digits long (8 bits).

2. You can move forward in memory (by pressing the space bar) but not back.

3. All addresses and data are displayed in hexadecimal. If you are unfamiliar with this number system, refer to Table 2-1. There are also explanations of hexadecimal numbers in many books, including T. C. Bartee, Digital Computer Fundamentals, McGraw-Hill, New York, 1974.

Note that you can start examining memory at any address. For example, type:

```
M

F

8

A

6

RETURN
```

to display the contents of memory locations F8A6, F8A7, F8A8, and F8A9 from left to right.

2.2 CHANGING MEMORY

You may change the contents of a memory location (after you have examined it) by typing / and then entering the new contents in hexadecimal (2 digits).

For example, examine the contents of memory locations 0, 1, 2, and 3 by typing:

```
M
0
.
RETURN
```

Now you can change the contents of memory location 0 to 05 by typing:

```
/
0
5
RETURN
```

Note that the AIM 65 prompts you by displaying the starting address after you type /. You can check to be sure that the memory was actually changed by repeating the examination procedure, i.e. by typing:

```
M
0
RETURN
```

Memory location 0 should now contain 05. Be careful--you must enter 05, not just 5. What happens if you type /, 5, RETURN?

In fact, you can change any or all of the four displayed locations in one operation by typing the new values in order and typing spaces for any values that you want to leave unchanged.

For example, let us place 06 in memory location 0, B7 in memory location 1, and E3 in memory location 3 while leaving memory location 2 unchanged. Type:

M	EXAMINE LOCATION
0	
RETURN	
/	(0) = 06
0	
6	
B	(1) = B7
7	
SPACE	(2) unchanged
E	(3) = E3
3	

We use parentheses around a memory address to indicate "contents of", i.e., (5) refers to the 8-bit data located at memory address 5.

Note that the AIM 65 prompts you by moving its cursor across the display. If you change or space over the last memory location no RETURN is necessary.

If you want to continue changing the contents of memory just press / again. The AIM 65 will display the next higher address. Remember that you can use M and SPACE to check to see that you have entered the data properly.

2.3 ENTERING A PROGRAM

To enter a machine language program, you must use the R6500 Microprocessor Programming Card. This card contains the 3-letter mnemonics (see Table 5-2) for all R6500 instructions. These instructions and R6500 assembly language programming are described in more detail in Section 6 of this manual and in the R6500 Programming Manual. For now, we will just discuss some simple examples.

You can enter a program by typing I followed by the proper series of mnemonics and operands. The operands give the microprocessor the additional information that it needs to execute the instructions (e.g., the memory address from which to load the accumulator or the destination for a branch instruction). Some instructions like TAX (move A to X) or CLC (clear carry) need no operands since the processor knows what to do from the operation code alone. On the Reference Card, such instructions are described as having implied addressing.

Let us look at a simple example program that logically ANDs the contents of memory locations 40 and 41 and places the result in memory location 42. Remember that all the addresses are hexadecimal. The program is:

```
LDA        40

AND        41

STA        42

BRK
```

Note the following features of this program:

1. LDA 40 loads the accumulator from memory location 40. The address is really 0040 but we do not have to enter the leading zeros.

2. AND 41 logically ANDs the accumulator with the contents of memory location 41. The result is placed in the accumulator.

3. STA 42 stores the accumulator in memory location 42.

4. BRK returns control to the AIM 65 monitor after the program has been executed. You should place this instruction at the end of all your programs so that the computer does not go wandering off aimlessly. Remember that the computer will continue executing instructions sequentially unless it is specifically told to do otherwise.

Now let us enter the program into memory as follows:

1. Type I. The AIM 65 responds by displaying the memory address at which it will start placing the instructions.

2. If the starting address is not zero, type *, 0, RETURN to make it zero.

3. Type L, D, A, 4, 0, SPACE to enter the LDA 40 instruction. Note that the AIM 65 automatically displays the memory address in which the next instruction will be placed.

4. Type A, N, D, 4, 1, SPACE to enter the AND 41 instruction.

5. Type S, T, A, 4, 2, SPACE to enter the STA 42 instruction.

6. Type B, R, K to enter the BRK instruction. Note that no SPACE is necessary since the BRK instruction requires no operands.

7. Type ESC to end program entry.

If you make a mistake, you can generally recover quite easily. In fact, the AIM 65 simply ignores most typing errors such as SAT instead of STA. The problems come when you accidently type a valid code that is not the one you wanted (like STX instead of STA) or type an address incorrectly (e.g., 43 instead of 42).

If you catch the error before you complete the mnemonic code or type RETURN, you can backspace and erase by typing DEL. Note that a character disappears from the display and the marker moves backward. However, this does not work if you have entered a 3-letter mnemonic or typed RETURN. Then you must correct the line by restarting the entry procedure at the address where you made the error.

For example, if I typed STX 42 instead of STA 42 at address 4, I could correct my error by typing:

```
*           AT ADDRESS 4

4

RETURN

S           STA 42

T

A

4

2

SPACE
```

Note that all we have done so far is enter the program into memory. We have not yet entered any data, executed the program, or produced any results.

Still another simple program takes the contents of memory location 40, clears the four most significant bits, and stores the result in memory location 41. We can clear the four most significant bits by logically ANDing the accumulator with 0F hex (00001111). Remember that logically ANDing with a '0' always gives zero (why?). The program is:

```
LDA         40

AND         #0F

STA         41

BRK
```

Note the following features of this program:

1. AND #0F logically ANDs the accumulator with the number 0F. This is called immediate addressing. Note the difference between AND #0F and AND 0F which logically ANDs the accumulator with the contents of memory location 000F. That memory location could contain any 8-bit number.

2. The '#' sign means "immediate', i.e. the following number is data rather than an address.

3. The BRK instruction at the end of the program restores control to the monitor just as in the previous example.

We can enter this program as follows:

1. Type I

2. If necessary, type *, 0, RETURN to make the starting address zero.

3.. Type L, D, A, 4, 0, SPACE to enter the LDA 40 instruction.

4. Type A, N, D, #, 0, F, SPACE to enter the AND # OF instruction. Remember to shift to type "#".

5. Type S, T, A, 4, 1, SPACE to enter the STA 41 instruction.

6. Type B, R, K to enter the BRK instruction.

7. Type ESC to end program entry.

You should read the description of the I command in Section 3.5.1 for details on how to enter instructions that we have not discussed here.

2.4 ENTERING DATA

Before we have the AIM 65 execute a program, we need some way of entering data and observing the results. This is simple since we can use the procedures that we have described previously for examining and changing the contents of memory.

For example, the first program from the previous discussion was:

```
LDA     40
AND     41
STA     42
BRK
```

This program requires data in memory locations 40 and 41. The result is saved in memory location 42.

Entering the data requires the following steps:

1. Type M, 4, 0, RETURN to observe the contents of memory locations 40 through 43.

2. Type /, B, 7, 6, 3, RETURN to enter the data into memory locations 40 and 41. We have placed B7 in memory location 40 and 63 in memory location 41 but any other values would be just as easy to enter.

Note that you might want to put zero in memory location 42 just to be sure that the answer was not already there.

To observe the result after the program has been executed, all we have to do is type M, 4, 0, RETURN. The first two numbers are the original data (memory locations 40 and 41) while the third number is the result (memory location 42). The situation is even simpler for the second example program since it only uses memory locations 40 (for the original data) and 41 (for the result).

2.5 EXECUTING A PROGRAM

To have the AIM 65 execute a program all we have to do is tell it where to start and then use the G command (for GO). Remember to put a BRK instruction at the end of your program or the AIM 65 may go and never come back. If this happens, press the RESET button.

So, to have the AIM 65 execute a program starting in memory location 0, set the RUN/STEP switch to RUN and type:

Key	Comment
*	STARTING ADDRESS
0	
RETURN	
G	GO
RETURN	

Note that typing G is just one step in a long process.
To actually run a program we must:

1. Enter the program into memory using the I command.

2. Enter the data into memory using the M and / commands.

3. Execute the program using the G command.

4. Observe the results using the M command.

Let us now see how the entire procedure works in some simple cases.

Example:

Logically AND the contents of memory locations 40 and place the result in 42.

DATA:

(40)=B7
(41)=63

RESULT:

(42)=23

Remember that the parentheses around the address means "contents of".

1. PROGRAM ENTRY

Type

```
I                   BEGIN PROGRAM ENTRY
*                   AT ADDRESS 0
0
RETURN
L                   LDA 40
D
A
4
0
SPACE
```

```
A                    AND 41
N
D
4
1
     SPACE
     S               STA 42
     T
     A
     4
     2
     SPACE           BRK
     B
     R
     K
     ESC             END PROGRAM
```

2. DATA ENTRY

```
Type

M                    EXAMINE MEMORY
4                    AT ADDRESS 0
0
RETURN
/                    CHANGE MEMORY
B                    (40)=B7
7
6                    (41)=63
3
0                    (42)=00
0
RETURN
```

3. PROGRAM EXECUTION

Type

```
*                    STARTING ADDRESS=0
0
RETURN
G                    GO
RETURN
```

4. OBSERVING RESULTS

Type

```
M                    EXAMINE MEMORY
4                    AT ADDRESS 40
0
RETURN
```

The result is the third number.

Try going through this procedure once. Repeat it for the following sample cases.

A. (40)=F3
   (41)=9A

   Result = (42)=92

B. (40)=D7
   (41)=AB

   Result = (42)=83

Example:

Clear the four most significant bits of memory location 40 and place the result in memory location 41.

DATA:

(40)=B7

RESULT:

(41)=07

1. PROGRAM ENTRY

```
        Type

        I                   BEGIN PROGRAM ENTRY
        *                   AT ADDRESS 0
        0
        RETURN
        L                   LDA 40
        D
        A
        4
        0
        SPACE
        A                   AND #0F
        N
        D
        #
        0
        F
```

```
SPACE
S
T
A
4
1
SPACE
B                   BRK
R
K
ESC                 END PROGRAM ENTRY
```

2. DATA ENTRY

```
Type

M                   EXAMINE MEMORY
4                   AT ADDRESS 40
0
RETURN
/
B                   (40)=B7
7
0                   (41)=00
0
RETURN
```

3. PROGRAM EXECUTION

Type

```
*                    STARTING ADDRESS = 0
0
RETURN
G                    GO
RETURN
```

4. OBSERVING RESULTS

Type

```
M                    EXAMINE MEMORY
4                    AT ADDRESS 41
1
RETURN
```

Result = (41) = 07

Try going through this procedure once. Repeat it for the following sample cases:

A. (40)=F3

Result = (41)=03

B. (40)=AB

Result = (41)=0B

2.6 EXAMINING REGISTERS

The R6502 microprocessor actually performs its operations using the following registers:

Program Counter
Processor Status or P register
Accumulator or A register
Index register X or X register
Index register Y or Y register
Stack pointer or S register

Let us now briefly discuss each of these registers. There is a more complete description in the R6500 Programming Manual.

1. PROGRAM COUNTER (or PC)

This is a 16-bit register which holds the address of the next instruction to be executed. Every time the processor uses this register, it adds one to the contents. Thus, the processor executes instructions sequentially unless a JUMP or BRANCH instruction specifically places a new value in the program counter.

2. PROCESSOR STATUS (or P)

This is an 8 bit register which reflects the current status of the CPU. Its bits are: (See Figure 2-1.)

Bit 7 (N)=1 if the last result had a 1 in its most significant bit, 0 if the last result had a 0 in its most significant bit. This bit is often called the NEGATIVE or SIGN flag.

Bit 6 (V)=1 if the last arithmetic operation produced a two's complement overflow, 0 if it did not. This bit is called the OVERFLOW flag.

Bit 5 = not used.

Bit 4 (B)=1 if the last instruction was BRK, 0 otherwise. This bit is called the BREAK COMMAND flag.

Bit 3 (D)=1 if the processor is in decimal mode, 0 if it is not. The bit is called the DECIMAL MODE flag.

Bit 2 (I)=1 if interrupts are <u>not</u> allowed, 0 if they are. This bit is called INTERRUPT DISABLE flag..

Bit 1 (Z)=1 if the last result <u>was zero</u>, 0 if it was not. This bit is called the ZERO flag.

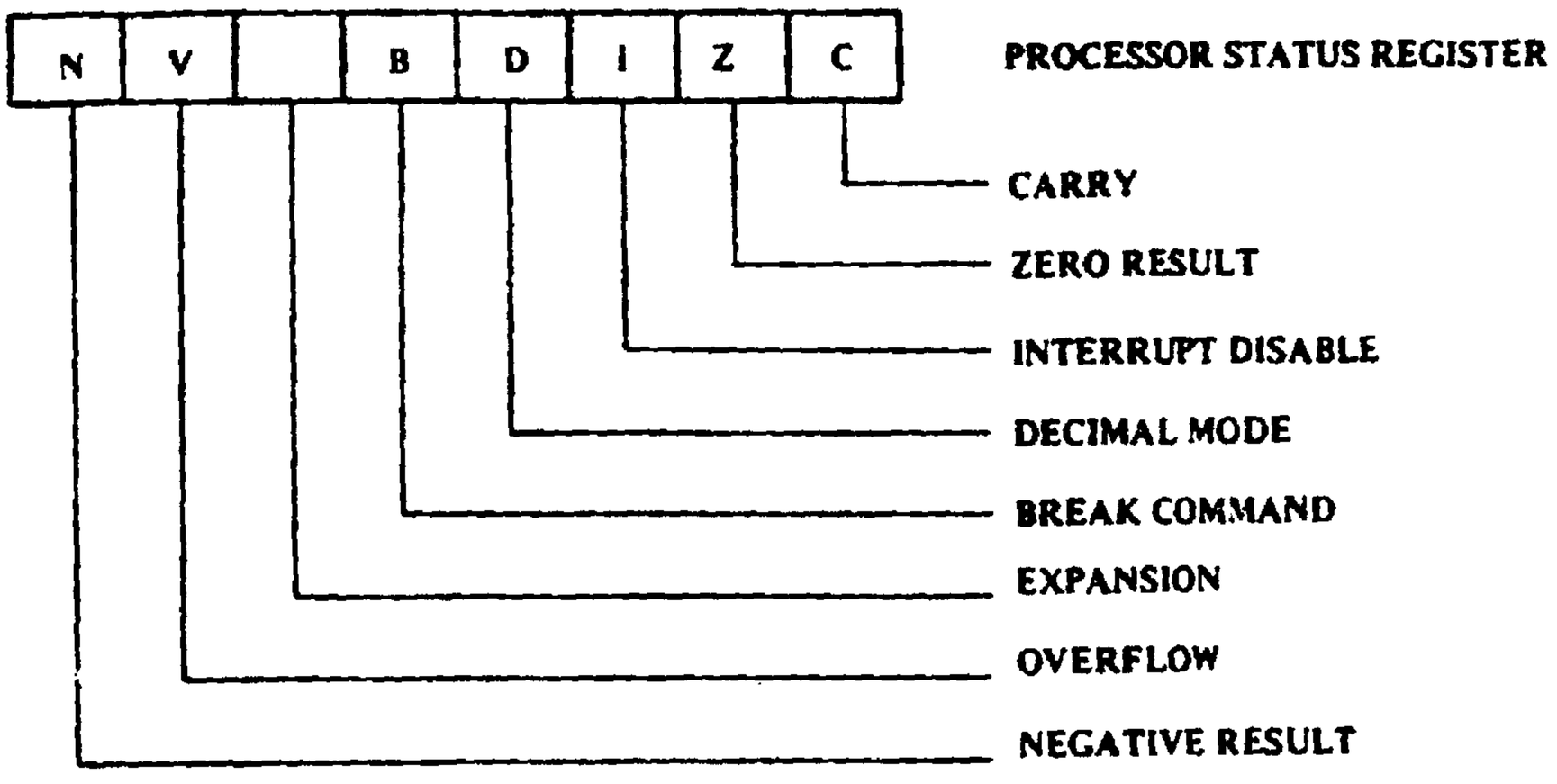


Figure 2-1. Processor Status Register

Bit 0 (C)=1 if the last addition produced a carry or the last subtraction did not require a borrow, 0 if the opposite conditions held. This bit is called the CARRY flag.

NOTE

Only the individual bits in the P register are meaningful. If you wish to observe or change those bits, you should consult Table 2-1 to convert between binary and hexadecimal.

3. ACCUMULATOR (or A)

This is an 8 bit register which is the center of processor operations. It acts much like the current sub-total in a calculator.

4/5. INDEX REGISTERS X and Y

These are two 8-bit registers which can be used as counters or indexes.

6. STACK POINTER (or S)

This is an 8-bit register which contains the address of the stack on page 1 of memory. If S contains F3, the next available stack location is at address 01F3.

To observe the current contents of all registers, type R. The AIM 65 will display the registers in the following order.

PC P A X Y S

Note that the program counter is 4 digits long while the other registers are 2 digits long.

2.7 CHANGING REGISTERS

You may change the contents of the registers with the following commands. Remember that PC is 4 digits long:

1. PC - *
2. Accumulator -A
3. X register -X
4. Y register -Y
5. Stack pointer -S
6. Processor status -P

We have listed these roughly in the order of frequency of use. You will find that you often want to change the program counter, accumulator and index registers. You will seldom want to change the stack pointer or processor status.

Examples:

1. Place 03E1 in the Program Counter.

Type	
*	ALTER PC
0	(PC)=03E1
3	
E	
1	
RETURN	

2. Place 5F in the accumulator.

Type	
A	ALTER A
5	(A)=5F
F	

3. Place 10 in index register X.

```
Type

X                    ALTER X
1                    (X)=10
0
```

4. Place 3 in index register Y.

```
Type

Y                    ALTER Y
3                    (Y)=37
7
```

Remember that all entries are in hexadecimal.

2.8 USING THE PRINTER

You can control the printer as follows:

1. Press CTRL and PRINT simultaneously to turn the printer on if it is off, and off if it is on. Note that the displays tell you the current state of the printer.

2. Press PRINT to have the printer print whatever is on the display. This command works even if you have turned the printer off.

3. Press LF (line feed) to advance the paper.

The printer output can give you a permanent record to study or retain. But there is no use wasting a lot of paper if you are just trying things out or checking operations.

2.9 RECORDING ON CASSETTES

Once you have entered a program into memory and corrected it you will probably want to record it on a cassette rather than re-enter it each time from the keyboard. We assume that the audio cassette recorder has been previously attached in position 1 according to the instructions in Section 9. We also assume that the volume control has been set appropriately (usually to the highest level).

The following procedure will allow you to record your program on tape.

1. Place a cassette in the recorder, rewind it, and then play it until you are past the leader. If your recorder has a counter, allow at least five counts.

2. Type D (for dump).

3. In response to the AIM 65 displaying FROM =, type the starting address of the program followed by RETURN.

4. In response to the AIM 65 displaying TO =, type the ending address of the program followed by RETURN.

5. In response to the AIM 65 displaying OUT =, type the device code T for audio tape in AIM 65 format.

6. In response to the AIM 65 displaying F =, type the file name. This can be any five alphanumeric or special characters. If the name is less than five characters long, type SPACE at the end. Simple file names would be PROG1, A1P6, SUM, or TEST.

7. In response to the AIM 65 displaying T =, type the recorder number (1).

8. Place the recorder in the record mode by pressing PLAY and RECORD simultaneously.

9. Type RETURN. The AIM 65 will now record the program on tape.

10. In response to the AIM 65 displaying MORE?, type N (for NO). The AIM 65 will then complete the recording.

11. When the recording has been completed, stop the recorder.

2.10 LOADING FROM CASSETTES

To load a program from a cassette, use the following procedure. We again assume that the audio cassette recorder has been attached according to the instructions in Section 9 and that the volume control has been set appropriately.

1. Place the cassette in the recorder and rewind it.

2. Type L (for load).

3. In response to the AIM 65 displaying IN =, type the device code T for audio tape in AIM 65 format.

4. In response to the AIM 65 displaying F =, enter the file name that you used in recording the tape. Type SPACE at the end if the name is less than five characters long.

5. In response to the AIM 65 displaying T =, type the recorder number (1).

6. Place the recorder in the Read Mode by pressing PLAY.

7. Type RETURN. The AIM 65 will now load the program from tape into memory.

8. When the program has been completely loaded, turn the recorder off.

Added 3/79

SECTION 3
THE AIM 65 MONITOR

The Monitor controls AIM 65 operation. The Monitor is a computer program that provides powerful software features and linkages to both AIM 65 and user programs. The Monitor is located in two 4K R2332 ROM's that are installed in sockets Z22 and Z23. An AIM 65 Text Editor is physically included with the Monitor but is described separately. (See Section 4.) The structure of the Monitor and Editor software is described in Section 7.4.

3.1 AIM 65 MONITOR FEATURES

The features of the AIM Monitor include:

- Major function entry and re-entry linkage--easy linkage to and from Editor, Assembler, BASIC, and user functions including initial entry and reentry capabilities. Single keystroke or RESET button depression returns control to the Monitor.

- Display and alter any register--any of the six registers may be displayed and altered.

- Display and alter memory--any memory location may be displayed and altered.

- Instruction mnemonic entry--R6500 machine language instructions may be directly entered into memory from typed mnemonic operation codes and hexadecimal operands.

- Disassemble memory--R6500 object code may be decoded (disassembled) from memory into R6500 mnemonics and hexadecimal operands.

- Selectable RUN/STEP program execution--user programs may be executed in the RUN Mode at full R6502 speed or in STEP Mode for debugging.

- Execution control--user programs can be initiated at specified program counter values. From one to 99 instructions or an indefinite number of instructions may be executed in the STEP Mode. Execution may be terminated at any time with the ESC key.

- Trace--instruction, register, and program counter trace capability exists in the STEP Mode. Either instruction or register trace may be performed during execution. Program counter trace may be performed after execution is terminated.

- Breakpoints--up to four breakpoint addresses may be entered, displayed, and selectively enabled to stop user program execution at specified addresses in the STEP Mode.

- RUN Mode BRK instruction control--BRK instructions may be placed in a user program to stop execution.

- Load and dump memory to and from various peripherals--memory may be loaded from, and dumped to, AIM 65 and user provided I/O devices. AIM 65 peripherals include keyboard, printer, and display. AIM 65 provides hardware and software to directly interface with audio cassette recorders and teletypewriter keyboard, printer, and paper tape reader/punch.

- Verify tape checksum--the record checksum on the audio tape can be checked to verify proper recording.

- User defined interface keys--three keys are dedicated to link directly to user-defined functions with simple return capability to the Monitor.

Table 3-1 lists the Monitor commands by functional grouping.

3.2 MAJOR FUNCTION ENTRY AND EXIT

Five commands are provided to enter other major AIM 65 functions from the Monitor. Four of these commands allow both initial entry and re-entry into the Editor and BASIC. There is only one entry command into the Assembler. An ESC command provides re-entry into the Monitor from most AIM 65 functions. The RESET button always returns control to the Monitor and performs "cold" or "warm" initialization (see Section 1.9).

3.2.1 E Command - Enter and Initialize the Editor

The E command enters and initializes the AIM 65 Text Editor. Refer to Section 4.2.1 for a detailed description.

CAUTION

Be careful not to initialize the Editor before desired information in the Editor Text Buffer has been permanently stored (see Section 4.2.1).

3.2.2 T Command - Re-enter the Editor

The T command re-enters the AIM 65 Text Editor at the top of the existing Text Buffer. Refer to Section 4.2.2 for details.

3.2.3 N Command - Enter Assembler

The N command enters the optional AIM 65 Assembler. Refer to Section 5.4 for a description of assembler command processing. The Monitor enters the assembler by executing a jump to subroutine (JSR) to address $D000. If a user provided function other than the Assembler is programmed in ROM or PROM and is installed in socket Z24, it may be called directly from the Monitor by typing N. Return to the Monitor, if desired, with an RTS.

3.2.4 5 Command - Enter and Initialize BASIC

The 5 command enters the optional AIM 65 BASIC Interpreter. Refer to the AIM 65 BASIC User's Manual for a description of the BASIC commands.

CAUTION

Be careful not to initialize BASIC before any desired BASIC program or data in RAM has been permanently stored.

The Monitor enters BASIC by executing a JSR to address $B000. If a user provided function other than BASIC is installed in socket Z25/Z26, see Section 10.2.

3.2.5 6 Command - Re-enter BASIC

The 6 command re-enters the AIM 65 BASIC Interpreter. Refer to the AIM 65 BASIC User's Manual for a description of the BASIC operation and commands.

The Monitor re-enters BASIC by executing a JSR to address $B003. If a user provided function other than BASIC is installed in socket Z25/Z26, see Section 10.2.

3.2.6 RESET - Enter and Initialize Monitor

The RESET command performs a hardware reset of the peripheral devices and initializes the AIM 65 Monitor.

Perform a "warm" reset by depressing the RESET button..

Perform a "cold" reset by either turning AIM 65 power off, waiting a couple of seconds, and then reapplying AIM 65 power or by changing address $A402 to $00 and then depressing the RESET button.

Example:

Press RESET

ROCKWELL AIM 65

3.2.7 ESC Command - Re-enter Monitor

The ESC command escapes from the existing command and returns to the Monitor. ESC is operative only in the commands that sample the keyboard. AIM 65 will respond to ESC by displaying the AIM 65 Monitor prompt:

<

3.3 DISPLAY/ALTER REGISTERS

Seven commands are provided to display or alter the contents of the six registers (program counter, processor status, accumulator, X register, Y register, and stack pointer). The alter commands are used most often to establish initial register values for checkout purposes. During normal program operation, the register contents would be initialized by previously executed instructions.

TABLE 3-1. AIM 65 MONITOR COMMANDS

CATEGORY	COMMAND	FUNCTION
Major Function Entry	RESET	Enter and Initialize Monitor
	ESC	Re-Enter Monitor
	E	Enter and Initialize Text Editor
	T	Re-Enter Text Editor
	N	Enter Assembler
	5	Enter and Initialize BASIC
	6	Re-Enter BASIC
Display/Alter Registers	*	Alter Program Counter
	P	Alter Processor Status
	A	Alter Accumulator
	X	Alter X Register
	Y	Alter Y Register
	S	Alter Stack Pointer
	R	Display Registers
Display/Alter Memory	M	Display Specified Memory Contents
	SPACE	Display Next Four Memory Contents
	/	Alter Memory Contents
Instruction Entry/ Disassembly	I	Instruction Mnemonic Entry
	K	Disassemble Memory

TABLE 3-1. AIM 65 MONITOR COMMANDS (Cont.)

CATEGORY	COMMAND	FUNCTION
Execution/Trace	G	Start Execution at Program Counter Address
	Z	Toggle Instruction Trace Mode On/Off
	V	Toggle Register Trace Mode On/Off
	H	Trace Program Counter History
Manipulate Breakpoints	?	Display Breakpoints
	#	Clear All Breakpoints
	B	Set/Clear Breakpoints
	4	Toggle Breakpoint Enable On/Off
Load/Dump Memory	L	Load Memory
	D	Dump Memory
Peripheral Control	CTRL PRINT	Toggle Printer On/Off
	PRINT	Print Display Contents
	LF	Line Feed
	1	Toggle Tape 1 Control On/Off
	2	Toggle Tape 2 Control On/Off
	3	Verify Tape Checksum
User Function Interface	F1	User Function 1
	F2	User Function 2
	F3	User Function 3

3.3.1 * Command - Alter Program Counter

The * command changes the value of the program counter.

Use the * command as follows:

1. Type SHIFT and * simultaneously. AIM 65 will respond with:

   <*>= ^

2. Enter the new hexadecimal value of the program counter. End the input with RETURN or a SPACE.

   Example:

   <*>=0300

In the example above, the program counter was changed to $0300. The instruction in memory location $0300 will be executed first when the G command (Start Execution at Program Counter Address) is entered.

3.3.2 P Command - Alter Processor Status

The P command alters the contents of the processor status register.

To alter the processor status register, type P. AIM 65 will respond with:

<P>= ^

Enter the new value of the processor status register as a two digit hexadecimal number. A leading zero must be entered in the left digit position if the left digit value is zero.

Example:

```
<P>=00
```

In the above example, the value of the processor status register was changed to $00.

3.3.3 A Command - Alter Accumulator

The A command alters the contents of the accumulator.

To alter the accumulator register, type A. AIM 65 will respond with:

```
<A>= ^
```

Enter the new value of the accumulator register as a two digit hexadecimal number. A leading zero must be entered in the left digit if the left digit value is zero.

Example:

```
<A>=01
```

In the above example, the value of A was changed to $01.

3.3.4 X Command - Alter X Register

The X command alters the contents of the X register.

To alter the X register, type X. AIM 65 will respond with:

```
<X>= ^
```

Enter the new value of the X register as a two digit hexadecimal number. A leading zero must be entered in the left digit if the left digit value is zero.

Example:

```
<X>=02
```

In the above example, the value of the X register was changed to $02.

3.3.5 Y Command - Alter Y Register

The Y command alters the contents of the Y register.

To alter the Y register, type Y =. AIM 65 will respond with:

```
<Y>=
```

Enter the new value of the Y register as a two digit hexadecimal number. A leading zero must be entered in the left digit if the left digit value is zero.

Example:

```
<Y>=03
```

In the above example, the value of the Y register was changed to $03.

3.3.6 S Command - Alter Stack Pointer

The S command alters the value of the stack pointer.

To alter the value of the stack pointer, type S. AIM 65 will respond with:

```
<S>=
```

Enter the new value of the stack pointer as a two digit hexadecimal number. A leading zero must be entered in the left digit if the left digit value is zero.

Example:

```
<S>=FF
```

In the above example, the value of the stack pointer was changed to $FF. Note that the stack is always in page one of memory, so the address of the stack is therefore $01FF.

3.3.7 R Command - Display Register Contents

The R command is used to display the current contents of the six registers.

To display the contents of the registers, type R. AIM 65 will print two lines. The first line shows the symbols for the registers and the second line shows the actual contents. The registers and their corresponding symbols are:

Program counter	****
Processor status	PS
Accumulator	AA
X register	XX
Y register	YY
Stack pointer	SS

Example:

```
.R
 **** PS AA XX YY SS
 0300 00 01 02 03 FF
```

In the above example, the registers and their contents are:

Program counter	(****)	= $0300
Processor status	(PS)	= $00
Accumulator	(AA)	= $01
X register	(XX)	= $02
Y register	(YY)	= $03
Stack pointer	(SS)	= $FF (which means that the stack is at address $01FF since it is always in page one.)

The R command also provides column headings for reference or when the register trace or breakpoints are being used.

3.4 DISPLAY/ALTER MEMORY

Three commands are provided to display or alter memory. The memory addressed may be used for program (instructions), data, or I/O.

3.4.1 M Command - Display Specified Memory Contents

The M command displays the hexadecimal contents of four consecutive memory locations, starting at the specified address.

Use the M command as follows:

1. Type M. AIM 65 will respond with:

   <M>=

2. Enter the hexadecimal address of the first of the four memory locations to be displayed. If the hexadecimal address is less than four digits long, end the input with RETURN or SPACE.

3. AIM 65 will display the contents of the four memory locations.

   Example:

   <M>=0300 EA AD 00 A2

In the above example, the memory locations and their contents are:

ADDRESS	CONTENTS
0300	EA
0301	AD
0302	00
0303	A2

Uninstalled memory will respond with a value equal to the two high order digits of the address.

Example:

```
<M>=1000 10 10 10 10
```

3.4.2 SPACE Command - Display Next Four Memory Contents

The SPACE command displays the contents of the next four memory locations, after the initial address value has been entered using the M Command. Use the SPACE command as follows:

1. Use the M command to display the first four memory locations.

2. Type SPACE. AIM 65 will display the contents of the next four memory locations.

After the initial use of the M Command, the SPACE Command may be used any number of times.

NOTE

If the M command is not used first to initialize the starting memory location, a random starting memory location will appear.

3.4.3 / Command - Alter Memory Contents

The / command alters any memory location displayed with the M command or the SPACE command.

Use the / command as follows:

1. Display the memory location to be altered using M command or SPACE command.

2. Type /.

3. AIM 65 will respond with the address of the first memory location that was displayed on the previous line.

4. If the first memory location is to be altered, enter the new contents in hexadecimal. If the location is to be left as is, type one SPACE.

5. Proceed to the next location and alter it, if needed.

6. When the altering of the locations displayed is

complete, type RETURN. If the last memory location on the line was altered, no RETURN is necessary.

7. To alter the next four locations, re-enter the command /.

Example:

```
<M>=0300 EA A0 00 A2
</> 0300 0F    27
```

In the above example, the following operations were performed:

Location 0300 was changed to $0F.
Location 0301 was left unchanged (one SPACE was entered).
Location 0302 was changed to $27.
Location 0303 was unchanged (RETURN was entered after location 0302 was changed).

If an attempt is made to alter uninstalled, protected, write-only address, or failed memory, AIM 65 will display a MEM FAIL message along with the address that caused the error.

Example:

```
<M>=1000 10 10 10 10
</> 1000 10
 MEM FAIL 1000
```

3.5 INSTRUCTION ENTRY/DISASSEMBLY

Two commands allow easy entry of R6500 instructions into memory and examination of instructions already in memory.

The I command encodes (or assembles) symbolic instructions entered on the keyboard into directly executable object code stored in memory. The K command decodes (or disassembles) object code from memory into symbolic instructions for user examination.

3.5.1 I Command - Instruction Mnemonic Entry

The I command enters R6500 instructions directly into memory as object code from symbolic instructions entered from the keyboard. Starting from a user entered address, operation codes (op codes) are entered using three-digit alphabetic abbreviations. Operands, if required, are entered in hexadecimal in accordance with the addressing mode formats. Invalid op codes and operands are ignored but cause an ERROR messsage to be displayed.

Use the I command as follows:

1. Type I. AIM 65 will respond with the current program counter address:

```
<I>
XXXX
```

2. The program counter address can be changed by typing * followed by a four-digit hexadecimal address. If address 0300 is entered, AIM 65 will respond with:

```
XXXX   *=0300
0300
```

3. Enter the three-digit alphabetic abbreviation of the operation code. An input error in either of the first two digits may be corrected by typing DEL and the correct digit.

If the entered op code does not require an operand, the object code is computed, stored in memory, and displayed in object code form along with the program counter address and the symbolic op code. The program counter is incremented by one. If you want to enter additional instructions in successive addresses, return to Step 3. If instruction entry is complete, return to the Monitor by typing ESC.

If the op code requires an operand, continue to Step 4.

If the op code is invalid, an ERROR message will appear. The correct op code may then be re-entered without altering the program counter address since it has not been incremented.

If a valid but undesired op code was entered, it may be corrected in one of two ways:

A. If the op code requires an operand, enter RETURN before entering an operand or deliberately enter an invalid operand. An ERROR message will be generated and the whole instruction can be re-entered since the program counter address was not changed.

B. If the op code does not require an operand, the object code was entered into memory and the program counter incremented. In this case, re-establish the previous program counter address as in Step 2.

4. Enter the operand in hexadecimal in accordance with the addressing mode formats. Refer to the R6500 Programming Manual for a complete description of the addressing formats. In some cases, a short form is allowed. The display, however, shows the standard form except for conditional branch instructions, which show the absolute address rather than the relative address. The form for operand entry in the appropriate address mode is shown below (where H is the hexadecimal data):

ADDRESSING MODE	OPERAND FORMAT	NOTES
Accumulator	A	
Immediate	#HH	
Zero Page	HH	
Zero Page, X	HH,X or HHX	
Zero Page, Y	HH,Y or HHY	
Absolute	HHHH	
Absolute, X	HHHH,X or HHHHX	
Absolute, Y	HHHH,Y or HHHHY	
Relative	HH or HHHH	(4)
(Indirect, X)	(HH,X) or (HHX) or (HH,X or (HHX	
(Indirect), Y	(HH),Y or (HH) Y	
(Indirect)	(HHHH)	

NOTES

1. Immediate, page zero, or relative addresses require the entry of two digits (HH).

2. Absolute addresses require the entry of four digits (HHHH).

3. The $ symbol preceeding hexadecimal digits is not permitted since all entries are defined as hexadecimal.

4. For conditional branches, the displacement from the program counter may be entered as a two-digit relative address or as a four-digit absolute address, in which case the correct value of the displacement is automatically computed.

End the operand entry with RETURN or SPACE. The op code and operand are computed and stored in memory. The program counter address, the op code object code, and the symbolic form of the op code and operand are displayed. If SPACE was used,a second line is displayed. This line contains the program counter and the object code form of both the op code and the operand.

If the operand is invalid, an ERROR message will be generated and the entire instruction must be re-entered.

An error in operand entry before RETURN or SPACE is entered may be corrected by entering DEL and re-entering the correct

data. An error in operand entry after RETURN or SPACE is entered may be corrected by typing ESC, re-entering the I Command, re-establishing the correct program counter address, and re-entering the complete instruction.

When entering additional instructions, return to Step 2. If instruction entry is complete, return to the Monitor by typing ESC.

Example:

```
<I>
 0300     *=0300
 0300 EA NOP
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0309     *=0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0318
```

3.5.2 K Command - Disassemble Memory

The K command disassembles object code from memory into symbolic R6500 instructions. Starting from a specified address, each byte of memory is disassembled until a valid op code is decoded. Once a valid op code is found, the appropriate number of following bytes are disassembled to

determine and display the instruction operand. Invalid op codes are indicated by question marks. Refer to Appendix K for a list of valid instructions.

Use the K Command as follows:

1. Type K. AIM 65 will respond with:

```
<K>*=
```

2. Enter the starting address in hexadecimal, then type RETURN. If 0300 was entered, AIM 65 will respond with:

```
<K>*=0300
/
```

3. Specify the number of instructions to disassemble by entering a decimal count from 01 to 99, RETURN meaning one instruction, or a . or SPACE meaning continuous disassembly. 00 means 100 instructions.

   AIM 65 will respond by disassembling instructions until the specified number of instructions are disassembled, RESET is pressed, or ESC is typed. The disassembly can be suspended by typing SPACE (type any key to resume the disassembly).

Example:

```
<K>*=0300
/05
```

```
0300 EA NOP
0301 A2 LDX #FE
0303 E8 INX
0304 D0 BNE 0303
0306 4C JMP 0310
<K>*=0310
/04
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
```

3.6 EXECUTION/TRACE COMMANDS

Four commands allow execution and detailed examination of a user written program. The G command executes the user program in the mode determined by the RUN/STEP switch. In the RUN mode, the program executes in real time with complete control of the CPU turned over to the user program.

In the STEP mode, program execution is stopped after each instruction for instruction trace, register trace, and breakpoint examination. The Z command controls the instruction trace while the V command controls the register trace. The breakpoint control is described in Section 3.7. After execution is terminated and control returned to the Monitor, a history trace of the program counter may be obtained using the H command.

3.6.1 G Command - Start Execution at Program Counter Address

The G command starts execution of a user program at the current value of the program counter.

Use the G command as follows:

1. Establish the desired position of the RUN/STEP switch.

2. If the STEP position is selected, perform the following:

   A. Initialize the value of the program counter using the * command.

   B. Set the desired state of the instruction trace mode using the Z command.

   C. Set desired state of the register trace mode using the V command.

   D. Establish any desired breakpoint addresses using the B command.

   E. Enable or disable breakpoint addresses using the 4 command.

   F. Display the register headings and contents using the R command.

3. Type G. AIM 65 will respond with:

   G/

4. In STEP Mode, enter the number of instructions to execute by entering a decimal count from 01 to 99, or a RETURN (meaning one instruction), or a . or SPACE (meaning continuous instruction execution).

5. The AIM 65 will execute instructions as follows until a terminating condition occurs:

A. In the STEP Mode, the next instruction to be executed will be disassembled and printed if the instruction trace mode (Z command) is on. The contents of the six registers will be printed prior to execution of the next instruction if the register trace mode (Y command) is on. Execution will terminate and control will be returned to the Monitor if the entered count of instructions is reached, a BRK instruction is executed, or a breakpoint address is reached (if breakpoints are enabled).

B. In the RUN Mode, execution will continue until a BRK instruction is executed, at which time control will be returned to the Monitor. Execution may also be terminated by moving the RUN/STEP switch to the STEP position. If the G command was initiated using the RETURN key, only one instruction will be executed in the STEP Mode before return to the Monitor.

NOTE

If the CPU attempts to execute an unimplemented op code or a jump to an improper address, it may hang up. If this occurs, the RESET switch must be pressed to interrupt program execution and allow the AIM 65 Monitor to regain control.

If execution termination is due to one of the G command terminating conditions, execution may be resumed at the current program counter address by repeating portions of Step 2 without reinitializng the program counter (*). Use the R command to check the value of the program counter before resuming execution.

Beispiel 1: Betriebsart STEP, Befehlsprotokollprogramm "Ein", Register-protokollprogramm "Ein".

ANMERKUNG

Die Beschreibung der Protokollprogramme (Trace-Programme) sind in Abschnitten 3.6.2 und 3.6.3 enthalten.

```
0306  22  00 00  01 FF
0306  4C  JMP  0310
0310  22  00 00  01 FF
0310  A0  LDY  #02
0312  20  00 00  02 FF
0312  88  DEY
0313  20  00 00  01 FF
0313  D0  BNE  0312
0312  20  00 00  01 FF
0312  88  DEY
0313  22  00 00  00 FF
0313  D0  BNE  0312
0315  22  00 00  00 FF
0315  4C  JMP  0301
0301  22  00 00  00 FF
 0301  A2  LDX  #FE
 0303  A0  00 FE  00 FF
 0303  E8  INX
 0304  A0  00 FF  00 FF
 0304  D0  BNE  0303
 0303  A0  00 FF  00 FF
 0303  E8  INX
 0304  22  00 00  00 FF
 0304  22  00 00  00 FF
<Z>ON
<V>ON
<*>=0300
<R>
 **** PS  AA XX YY SS
 0300  A0  00 FF  01 FF
<G>.
 0301  A0  00 FF  01 FF
0301  A2  LDX  #FE
0303  A0  00 FE  01 FF
0303  E8  INX
0304  A0  00 FF  01 FF
0304  D0  BNE  0303
0303  A0  00 FF  01 FF
0303  E8  INX
0304  22  00 00  01 FF
0304  D0  BNE .0303
```

Beispiel 2: Betriebsart STEP, Befehlsprotokollprogramm "Ein", Registerprotokollprogramm "Aus".

```
<Z>OFF
<Z>ON
<V>OFF
<*>=0300
<R>
 **** PS AA XX YY SS
 0300 22 00 00 00 FF
<G>/
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0303 E8 INX
 0303 E8 INX
```

Example 3: STEP Mode, instruction trace off, register on.

```
<Z>OFF
<W>ON
<*>=0300
<R>
 **** PS  AA XX YY SS
 0300 22 00 00 00 FF
<G>/.
 0301 22 00 00 00 FF
 0303 A0 00 FE 00 FF
 0304 A0 00 FF 00 FF
 0303 A0 00 FF 00 FF
 0304 22 00 00 00 FF
 0306 22 00 00 00 FF
 0310 22 00 00 00 FF
 0312 20 00 00 02 FF
 0313 20 00 00 01 FF
 0312 20 00 00 01 FF
 0313 22 00 00 00 FF
 0315 22 00 00 00 FF
 0301 22 00 00 00 FF
 0303 A0 00 FE 00 FF
 0304 A0 00 FF 00 FF
 0303 A0 00 FF 00 FF
 0304 22 00 00 00 FF
 0306 22 00 00 00 FF
 0306 22 00 00 00 FF
```

3.6.2 Z Command - Toggle Instruction Trace Mode On/Off

The Z command controls the instruction trace mode when the RUN/STEP switch is in the STEP position and instructions are being executed in response to the G, F1, F2 or F3 command. The instruction trace shows a disassembly of each instruction before the instruction is executed.

To use the Z command, type Z. AIM 65 will respond with the state of the instruction trace mode:

Z ON or
Z OFF

Example:

```
<Z>ON
<Z>OFF
```

In the above example, the first Z command toggled the instruction trace mode to ON. The second Z command toggled the instruction trace mode to OFF.

3.6.3 V Command - Toggle Register Trace Mode On/Off

The V command controls the register trace mode when the RUN/STEP switch is in the STEP position and instructions are being executed in response to the G, F1, F2, or F3 command. The register trace shows the contents of each register, in the format of the R command (Section 3.3.7), after each instruction is executed.

To use the V command, type V. AIM 65 will respond with the status of the register trace mode:

<V> ON or
<V> OFF

3.6.4 H Command - Trace Program Counter History

The H command displays the addresses of the last four instructions that were executed and the address of the next instruction to be executed. This trace capability exists only after the AIM 65 has been executing instructions in the STEP Mode in response to the G, F1, F2, or F3 commands.

Use the H command as follows:

1. Execute the desired instructions in the STEP Mode using the G, F1, F2, or F3 commands.

2. After the Monitor prompt, type H. AIM 65 will respond with:

```
<H>
XXXX  Earliest of last four instructions executed.
XXXX
XXXX
XXXX  Address of instruction just executed.
XXXX  Address of next instruction to be executed.
```

Example:

```
<H>
 0303
 0304
 0306
 0310
 0312
```

The example above shows a program which is a string of sequential non-jump instructions starting at $0303 with a JMP $0310, RTS, RTI or a branch instruction at $0306.

3.7 MANIPULATE BREAKPOINTS

Four commands are provided for breakpoints to check, totally clear, selectively set or clear, and/or enable or disable breakpoints. These commands are used in conjunction with the G command in the STEP Mode to stop instruction execution at specified breakpoint addresses. These commands can, therefore, be used during program checkout to ensure program sequencing to expected addresses or to allow intermediate data in memory to be checked at specified addresses.

3.7.1 ? Command - Display Breakpoints

The ? command displays the address of each of the four breakpoints. The leftmost, four-digit hexadecimal value is the address of breakpoint 0 while the rightmost value is the address of breakpoint 3. $0000 indicates that the breakpoint is cleared, i.e., no breakpoint address is set.

To use the ? command, type SHIFT and ? simultaneously. AIM 65 will respond with:

```
<?>
AAAA AAAA AAAA AAAA
```

Example:

```
<?>
0312 0000 0000 0000
```

In the above example, the breakpoint numbers and their corresponding addresses are:

BREAKPOINT NUMBER	BREAKPOINT ADDRESS
0	$0312
1	Not Set
2	Not Set
3	Not Set

3.7.2 # Command - Clear Breakpoints

All breakpoints may be cleared by using the # command. Breakpoints should be cleared when AIM 65 power is turned on. RESET does not alter the breakpoint addresses.

To use the # Command, type SHIFT and # simultaneously.

AIM 65 will respond with:

<#>OFF

This indicates that all the breakpoints have been set to $0000.

Example:

<#>OFF

3.7.3 B Command - Set/Clear Breakpoints

The B command sets or clears the address for any of the four breakpoints (breakpoint 0 through breakpoint 3).

For breakpoints to be checked, AIM 65 must be in the STEP Mode, and the breakpoints must be enabled with the 4 command.

When the AIM 65 is in the STEP Mode, and the breakpoints are enabled, the processor halts each time an instruction fetch is made in the address range $0001 to $9FFF. Entry is made to the Monitor via the NMI interrupt vector (unless the NMI vector address in location $A402 has been altered). A check is made to see if the breakpoints are enabled (see the 4 command). If the breakpoints are enabled, each set breakpoint address is compared with the address of the instruction about to be executed. If the address of one of the set breakpoints matches the address of the instruction about to be executed, execution of the program is halted and control is returned to the Monitor.

Use the B command as follows:

1. Type B. AIM 65 will respond with:

   <B> BRK/

2. After the / prompt, specify the breakpoint to be set/cleared by entering a digit between 0 and 3. AIM 65 will respond by printing the number of the breakpoint entered, and an = prompt. For example, if 0 is entered:

   <B> BRK/0=

3. To set a breakpoint, enter the four-digit hexadecimal address at which the program is to halt. To clear a breakpoint, enter 0.

4. After the address has been entered, type RETURN. Control will return to the Monitor. Re-enter the B command to set or clear additional breakpoints.

Example:

```
<B>BRK/0=0310
<B>BRK/1=0312
<B>BRK/2=0
```

In the above example, breakpoint 0 was set to location $0310, breakpoint 1 was set to location 0312, breakpoint 2 was left unchanged and breakpoint 3 was set to location $0000 (i.e., cleared).

3.7.4 4 Command - Toggle Breakpoint Enable On/Off

The 4 command toggles the breakpoint enable ON or OFF. When the breakpoint enable is ON, and the Step mode is selected, the breakpoints in a program are checked.

The normal mode of operation is to assign the breakpoint addresses with the B command and then enable these breakpoints with the 4 command when the user wants them to be checked.

The 4 command also allows breakpoint addresses to be disabled temporarily without requiring them to be reentered later.

The breakpoint enable is automatically turned OFF when AIM 65 power is turned on. Subsequent RESETS do not affect breakpoint enable.

To use the 4 command, type 4. The system will toggle the breakpoint enable and display the result:

```
<4> ON or
<4> OFF
```

Example:

```
<4>ON
<4>OFF
```

In the above example, the breakpoints were enabled (toggled ON) when the first 4 command was entered. The breakpoints were disabled (toggled OFF) when the second 4 command was entered.

3.8 LOAD/DUMP MEMORY

Two commands allow R6500 object code to be loaded into memory from an input device or dumped from memory to an output device.

3.8.1 L Command - Load Memory

The L command loads object code from any system device into memory.

Use the L command as follows:

1. Type L. AIM 65 will respond with:

```
<L> IN=
```

2. Type the code of the input device from which the object code is to be loaded:

DESIRED INPUT DEVICE	ENTER DEVICE CODE	ADDITIONAL PROCEDURE
Keyboard	<RETURN> or <SPACE>	
Audio Tape - AIM 65 Format	<T>	See Section 9.1.2
Audio Tape - KIM-1 Format	<K>	See Section 9.1.2
TTY Punched Tape	<L>	See Section 9.2.2
User Defined	<U>	See Section 7

3. AIM 65 will load the object code from the specified device into memory. When all the code has been loaded, the AIM 65 will print the Monitor prompt (<).

   If any of the records being read contains a checksum error, or if any part of the memory fails to write, an error message will be printed (see Appendix L), indicating the first address of the record which caused the error.

3.8.2 D Command - Dump Memory

The D command is used to dump the contents of memory to an output device. Memory contents dumped are in R6500 object code format (see Appendix F.2) from the address specified after FROM=, through the address specified after TO=. Multiple dumps from different portions of memory may be performed by entering new beginning and ending addresses after responding Y to the MORE? prompt. An N response is required to terminate the dump properly.

Use the D command as follows:

1. Type D. AIM 65 will respond by asking for the dump beginning address:

```
<D>
FROM=
```

2. Enter the beginning address to be dumped, in hexadecimal. An input error may be corrected using DEL, or by continuing to enter up to 11 numbers; AIM 65 will accept only the last four numbers entered. End the input with RETURN or SPACE.

   If 0300 was entered, AIM 65 will respond by asking for the dump ending address:

```
FROM=0300    TO=
```

3. Enter the ending address to be dumped, in hexadecimal. An input error may be corrected in the same manner as

in the beginning address. End the input with a RETURN or SPACE. If 0340 was entered, AIM 65 will respond with:

```
FROM=0300     TO=0340
OUT=
```

4. Type the code of the output device to which the dump is to be directed:

DESIRED OUTPUT DEVICE	ENTER DEVICE CODE	ADDITIONAL PROCEDURE
AIM 65 Display/ Printer	<RETURN> or <SPACE>	
AIM 65 Printer	<P>	
Audio Tape - AIM 65 Format	<T>	See Section 9.1.2
Audio Tape - KIM-1 Format	<K>	See Section 9.1.2
TTY Punched Tape	<L>	See Section 9.2.2
User Defined	<U>	See Section 7
Dummy Output (None)	<X>	

5. The memory contents will be dumped to the specified output device in R6500 object code format. When memory has been dumped through the specified ending address, AIM 65 will display:

   MORE?

6. If another section of memory is to be dumped, enter a Y (yes) response. AIM 65 will ask for the new beginning and ending addresses. If no more memory is to be dumped, enter an N (no) response.

7. After an N response, AIM 65 will output the terminating record with a zero byte count (see Appendix F.2).

NOTE

If the dump is not terminated with an N response, the last file record containing the file record count will not be recorded causing a subsequent improper load or tape verify.

Example 1:

```
<D>
FROM=0200 TO=0366
OUT=T F=DUMP1 T=1
MORE?N
```

This example dumps memory locations $200 to $366 to a tape file called DUMP1 located on the tape recorder number 1. No other segments were dumped.

Example 2:

```
<D>
FROM=0200 TO=0366
OUT=T F=DUMP2 T=2
MORE?Y
FROM=0300 TO=0380
MORE?N
```

This example dumps memory locations $200 to $366 to tape file called DUMP2 located on tape recorder number 2. More memory locations were to be dumped, so the question MORE? was answered with Y. The second dump was from location $0300 to location $0380. No more segments were dumped to this file.

Example 3:

```
<D>
FROM=0300 TO=0316
OUT=

;170300E9A17EE30CFD4
C10364F1F0DCEE1172A
00298D0FD4C010DE1
MORE?Y
FROM=0380 TO=03B0

;180380FCF0F25564D1F
1D0E9B0F4E0FED8C172F
F76F2E2D22F6B0129F
;18039BE7F4E363EB547
250F6507672709F0F072F
7D1F4600AE74901007
;0103B0F301A7
MORE?N;0000050005
```

This example shows an actual memory dump from location $0300 to $0316 and from $0380 to $03B0. The question OUT= was answered with the response RETURN, causing the dump to be printed on the AIM 65 printer and displayed on the AIM 65 display.

NOTE

If memory is dumped to audio cassette using KIM-1 format (OUT=K), the TO address entered must be one byte greater than the last address to be dumped.

3.9 PERIPHERAL CONTROL

The peripheral control commands allow the printer and audio tape recorders to be controlled and miscellaneous functions to be performed.

3.9.1 Control Print Command - Toggle Printer On/Off

The CTRL PRINT command turns the printer control on if it is off, and off if it is on.

If the printer control is ON, entering CTRL PRINT will toggle it to OFF. If the printer control is OFF, entering CTRL PRINT will toggle it to ON.

The command is entered by depressing the PRINT and CTRL keys together. The status of the printer control will be displayed but not printed.

When the printer control is ON, the Monitor will print the same commands and data that are displayed. If the printer control is OFF, the only time information will be printed is when the PRINT key alone is depressed.

NOTE

The printer control is turned ON by the first RESET after AIM 65 power is turned on. Subsequent RESETs will not change the active state of the printer control.

3.9.2 PRINT Command - Print Display Contents

The PRINT command causes the displayed information to be printed. This commanded print will occur regardless of the printer control state as long as the Monitor, Editor, Assembler, or BASIC Interpreter is active.

3.9.3 LF Command - Advance Paper

The LF (Line Feed) Command is used to advance the printer one line. The LF Command will operate whenever the Monitor, Editor, Assembler, or BASIC Interpreter is active.

3.9.4 1 (2) Command - Toggle Tape 1 (2) Control On/Off

The text to follow describes the 1 command. It applies equally to the 2 command.

The 1 command is used to toggle the Tape 1 control. If the Tape 1 control is ON, entering the 1 command will turn it

OFF. If the Tape 1 control is OFF, entering the 1 command will turn it ON.

The Tape 1 control is usually connected to the tape recorder number 1 Remote line (see Section 9). If so connected, the tape recorder will not record, play, advance, or rewind unless the Tape 1 control is ON. The Monitor and Editor commands requiring tape recorder number 1 operation will command the Tape 1 control ON when required and turn the control signal OFF upon completion of the command.

These commands are L (Load) and D (Dump) in the Monitor and R (Read) and L (List) in the Editor. To manually operate tape recorder number 1, the Tape 1 control must be turned ON using the 1 command.

NOTE

The Tape controls are turned on when AIM 65 power is turned on. RESET does not change the active state of the Tape controls.

3.9.5 3 Command - Verify Tape

The 3 command is used to verify that the block checksum for either source or object code was properly recorded on audio tape using the dump command. This verification should be performed before the contents of memory are altered, in case the data was not properly recorded.

To use the 3 command, type 3. AIM 65 will respond with:

```
<3>IN=
```

Type T. AIM 65 will respond with:

```
<3>IN=T  F=
```

Example:

```
<3>IN=T F=OBJ  T=1
```

Enter the file name and tape recorder number as shown in Section 9.1.6. The operation will continue as described in that section.

3.10 USER FUNCTION INTERFACE

Three commands allow execution of three separate user written functions (programs) from the AIM 65 Monitor using keys F1, F2, and F3. In order to use a function key, the linkage to the user function must be provided. The linkage is in the form of a JMP instruction to the function starting address. The JMP instruction is to be located at the function linkage address (see the specific function number for the actual address). At the completion of the user function, control to the AIM 65 Monitor may be regained with an RTS instruction at the end of the user function instruction.

The JMP instruction can be established at the function linkage addresses using the I Instruction Mnemonic Entry, the N Assembler, or by entering the JMP instruction in hexadecimal form using the / Alter Memory commands.

3.10.1 F1, F2, F3 Command - User Functions 1, 2, and 3

The F1, F2, and F3 commands are used to enter user functions.

To use the F1, F2 or F3 command, proceed as follows:

1. Establish a JMP instruction in address $010C to the Function 1 starting address, in $010F for Function 2, or in $0112 for Function 3.

2. Enter the function by entering command F1, F2, or F3. AIM 65 will respond with the following display and will start function 1, 2, or 3.

KEY	PROMPT
F1	<[>
F2	<]>
F3	<^>

3. Return to the Monitor by executing an RTS instruction without a preceding JSR in the user function program or by depressing the RESET button.

F1 Example:

```
<I>
 0200     *=010C
 010C 4C JMP 0220
 010F     *=0220
 0220 60 RTS
 0221
<E>
```

F2 Example:

```
<1>
 0200     *=010F
 010F 4C JMP 0240
 0112     *=0240
 0240 60 RTS
 0241

<1>
```

F3 Example:

```
<1>
 010F     *=0112
 0112 4C JMP 0260
 0115     *=0260
 0260 60 RTS
 0261

<1>
```

SECTION 4
AIM 65 TEXT EDITOR

The AIM 65 Text Editor allows you to create and edit files of text. These files are most commonly used as the source program into the R6500 Assembler, and are written in R6500 assembly language (see Section 5). The Text Editor files are also used to store messages that are operated on by user written programs. Another use is that of general documentation.

The Editor is used by entering data into a Text Buffer from an input device, editing the data, then either storing the data on an output device or operating directly on the data using another AIM 65 or user program. Usually the AIM 65 keyboard is used to input data and an audio cassette recorder is used for storing data. Editing is performed from the keyboard. The most common output devices are the display/printer or printer for hardcopy printout and the audio cassette for permanent data storage.

It is possible that a given program may be too long to store and edit in the Text Buffer at one time. In this case, the total program should be divided into several smaller files. The Assembler has a .FILE directive that allows individual source files to be linked together to form a total source program. See Section 5.8.8 for details.

The process of dividing large programs or data into smaller segments (modules) provides a convenient method for data handling and editing.

4.1.1 AIM 65 Text Editor Features

The features of the AIM 65 Text Editor include:

Complete User Control of Memory -- The user may establish the text buffer anywhere in available RAM.

Input Device Flexibility -- The input to the Editor may come from any of several peripheral devices:

- AIM 65 or TTY keyboard
- Audio cassette
- TTY punched paper tape
- User defined device

Output Device Flexibility -- The output from the Editor may be directed to any of several peripheral devices:

- AIM 65 display/printer
- AIM 65 printer only
- Audio cassette
- TTY punched paper tape
- User defined device

Flexible Editing Commands -- Simple single character commmands provide quick and easy edit functions:

Go to top or bottom of text
Go up or down a line
Find a character string
Change a character string to a different character string
List one or more lines to an output device
Insert one line from the keyboard
Read one or more lines from an input device
Delete a line
Display the current line
Quit the Editor

Re-enter and Re-edit Capability -- The Editor also provides a re-entry capability so that previously entered text may be re-edited to correct errors. This is especially useful when assembling: a source program may be entered in the Text Editor, assembled to identify any coding errors and the Editor re-entered to correct the errors. When an error-free assembly is attained, the source file may be permanently saved on audio cassette.

4.1.2 Text Buffer

The text is stored in a user-specified area of RAM called the Text Buffer. Upon initial entry into the Editor, the user must define the starting and ending limits of the Text Buffer. If it is desired to allocate all of available memory to the Text Buffer, default limits determined by AIM 65 may be used. The default lower limit is $0200. This bypasses RAM page 0, which is reserved for user and AIM 65 data, and page 1, which is reserved for the user and AIM 65 stack and for AIM 65 data. The default upper limit is

determined by the amount of contiguous installed RAM, starting with address $0200. AIM 65 checks for the existence of a page of RAM by performing a write and read test every $100 addresses.

Data is stored in the Text Buffer in ASCII format (see Appendix E for the ASCII code format). Each character entered requires one byte (8 bits) of RAM. The text is also stored in variable length lines. Each text line ends with a RETURN (ASCII $0D) which is stored after the last text character. Line Feed (ASCII $0A) characters are not stored.

A Null character (ASCII $00) is stored after the last text line to indicate the end of active text. Care should be taken that the character $00 is not inadvertently stored in the active text area of the Text Buffer. If a $00 is stored in the active text area, all the text past the $00 will not be available to the user using normal editor commands (see Editor Data Recovery Techniques, Section 4.1.3).

To estimate the amount of RAM required for the Text Buffer: allow one byte for each text character, one byte for each line terminated with a RETURN (ASCII $0D) and one byte for the text end character (ASCII $00). Additional memory should be allocated to allow for text additions and changes.

The actual starting and ending addresses of the active text in the Text Buffer as well as the Text Buffer ending address can be determined by examining the following dedicated memory locations:

ADDRESS	PARAMETER	EXAMPLE
00E1	Text Ending Address Low	$42
00E2	Text Ending Address High	$03
00E3	Text Starting Address Low	$00
00E4	Text Starting Address High	$02
00E5	Text Buffer Ending Address Low	$00
00E6	Text Buffer Ending Address High	$04

Figure 4-1 illustrates this example.

4.1.3 Editor Data Recovery Techniques

Data in the Text Buffer may appear to be lost due to two types of inadvertent actions:

- Initialization of the Editor using the Monitor E command before data in the old Text Buffer is permanently stored.

- Storing of the text end character ($00) somewhere in the active text area.

INADVERTENT INITIALIZATION RECOVERY

When the Text Buffer is initialized with the Monitor E Enter Editor command, only one byte of text in the old text area may be changed. This will occur only if the new Text Buffer starting address is the same as the old starting address or is somewhere in the old active text area. The

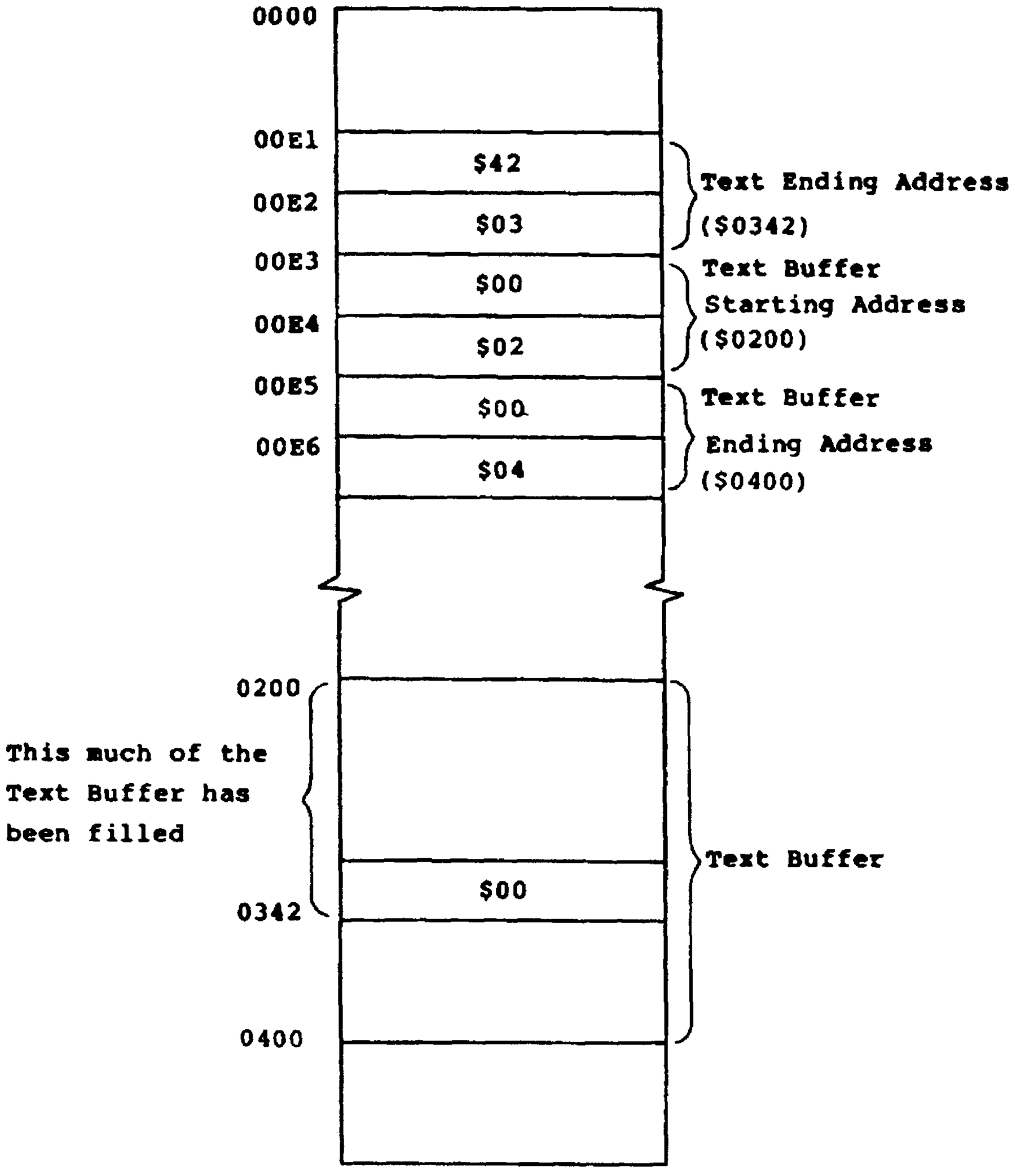


Figure 4-1. Example Text Buffer in RAM

remaining old text will still be intact, but a little difficult to find. This assumes, of course, that AIM 65 power has not been turned off or other data has been loaded into the old active text area in the meantime.

In order to recover the old Text Buffer contents, the old active text conditions must be reconstructed.

The recovery procedure is:

1. Initialize the Editor with the Monitor E command and set the Text Buffer starting "FROM=" and ending "TO=" address limits to the same value as the old buffer.

2. Using the Monitor M and / commands, change the $00 value located at the address specified by the starting address ($00E3 and $00E4) to a temporary valid ASCII character, e.g., $41 for "A". After the Text Buffer is completely recovered, the "A" can be easily changed back to its correct value.

3. When the Editor was inadvertently initialized, the Editor stored a $00 at the address specified by the new Text Buffer starting address to indicate an unfilled buffer. If this $00 is in the old active text area and is at an address different from the $00 in Step 2, it too must be located and changed to a temporary valid ASCII text character code. This character can also be easily changed to its proper value later. Use the Monitor M and SPACE commands to scan memory starting with the address specified by

the current Text Buffer starting address ($00E3 and $00E4). If data has been read into the current Text Buffer, the $00 may be located further down in the Text Buffer.

4. Using the Monitor M and SPACE commands, scan memory starting with the Text Buffer starting address specified in $00E3 and $00E4, locate the old end of active text indicator ($00). Enter the located address into the Text Buffer ending address ($00E1 and $00E2).

5. Re-enter the Editor using the Monitor T command. The top line of the data in the old Text Buffer should be displayed. Go to bottom of the data using the Editor B command. The last line of data in the old Text Buffer should be displayed. If not, one of the recovery steps was not completed properly.

6. Find any temporary ASCII characters loaded into memory in place of $00 to help recover the old Text Buffer. Change them to the desired values.

   Note that the $00 character will be found immedately after a $0D end of line indicator. If the $00 is not located, the $00 can be entered into memory after any $0D character and that address entered as the Text Buffer ending address ($00E1 and $00E2) to help locate the old end of text.

TEXT BUFFER RECOVERY PROGRAM

A program may be entered using the I command that will automatically perform the Text Buffer recovery. This program performs the steps discussed in the previous paragraph. The program starts at $03D0 in order to place it at a high RAM value in the 1K RAM version. By changing the high address value from "03" to "0F" in the program counter and conditional branch operand fields, the program can be moved to high RAM in the 4K version.

The program can easily be typed in when needed or may be typed in and then recorded on audio cassette in object code form using the Monitor D command for future use, if required. The program operates by changing the text ending address to the address of the first $00 located past the initial $00 at the text starting address. As such, the program may be run more than once if the first run found another $00 which was not the expected end of text indicator.

Type in the Text Buffer recovery program as follows:

```
<I>
 F474     *=03D0
 03D0 AD  LDA 00E3
 03D3 8D  STA 00E1
 03D6 AD  LDA 00E4
 03D9 8D  STA 00E2
 03DC A9  LDA #00
 03DE A8  TAY
 03DF A9  LDA #01
 03E1 91  STA (E3),Y
 03E3 98  TYA
 03E4 D1  CMP (E1),Y
 03E6 F0  EE 03F1
 03E8 E6  INC E1
 03EA D0  BNE 03E4
 03EC E6  INC E2
 03EE 4C  JMP 03E4
 03F1 4C  JMP E1A1
```

Verify the program by disassembling 16 instructions using the K command.

RUNNING THE TEXT BUFFER RECOVERY PROGRAM

1. Initialize the Text Editor to the initial Text Buffer limit starting and ending addresses, then return to the Monitor using ESC.

2. Enter the Text Buffer recovery program using the Monitor I command or load from audio cassette using the Monitor command.

3. Set the program counter to the starting address of the Text Buffer Recovery program.

   For example:

   ```
   *=$03D0
   ```

4. Type G/. to execute in the program. When the program finds $00, it will display the Monitor prompt.

NOTE

If a $00 is not found, the program may hang up. Return to the Monitor by depressing RESET.

5. Type T to re-enter the Text Editor at the top of the Text Buffer. Change the first character as appropriate.

EXAMPLE OF TEXT BUFFER RECOVERY USING THE TEXT BUFFER RECOVERY PROGRAM

```
<E>
EDITOR
FROM=0200   TO=0380

IN=
LINE 1
LINE 2
LINE 3
=<?
=<?
<T>
LINE 1
=<L>
/.
OUT=
LINE 1
LINE 2
LINE 3

END
<E>
EDITOR
FROM=0200   TO=0380

IN=
END
=<Q>
```

```
<T>
END
=<Q>

<*>=03D0
<G>/.
<T>
AINE 1
=<L>
/.
OUT=
AINE 1
LINE 2
LINE 3

=<T>
AINE 1
=<C>
A
AINE 1
TO=L
LINE 1
=<T>
LINE 1
=<L>
/.
OUT=
LINE 1
LINE 2
LINE 3
```

4.1.4 Line Pointer

The AIM 65 Text Editor is a line oriented editor; all Editor operations begin from the start of the active line. The active line is identified by the line pointer, which is always positioned in front of the first character of the active line. After an Editor operation is performed, the line pointer is positioned to the start of either the last line operated on or one line down from the last line operated on, depending upon the command.

The active line is displayed at the completion of most Editor commands, depending on output device selection. If there is any doubt where the line pointer is positioned, the Editor SPACE command will display the active line.

Line pointer positioning commands allow easy manipulation of the line pointer. Using these commands, the line pointer can be easily and quickly moved to the top of the text, to the bottom of the text, up one line, or down one line.

4.1.5 Dummy Line

A dummy line is provided after the last active text line to allow text to be added at the end of the buffer. If the line pointer is positioned on the last active line of text in the Text Buffer, it must be moved down one line to the dummy line using the D command to read or insert new text at the end of the active text.

When the line pointer is positioned on the dummy line, either "END" or no data will be displayed.

An L command listing all the active text will leave the line pointer positioned on the dummy line.

4.2 EDITOR ENTRY AND EXIT COMMANDS

Two commands permit the Editor to be entered from the Monitor. One command initializes the Text Buffer upon entry, the other allows re-entry to the Editor without affecting the text stored in the Text Buffer. Two separate commands allow return to the Monitor: one is used to quit the Editor during an Editor idle state, the other is used the escape during Editor command execution.

4.2.1 E Command - Enter and Initialize the Editor

The Monitor E command is used to enter the Editor, initialize the Text Buffer, and to read data into the text buffer.

Use the E command as follows:

1. After the Monitor prompt, type E. AIM 65 will enter the Editor mode and respond with:

```
E
EDITOR
FROM=
```

Table 4-1. AIM 65 Text Editor Commands

CATEGORY	COMMAND	FUNCTION
Editor Entry and Exit	E	Enter and Initialize Editor
	T	Re-Enter Editor
	Q	Quit Editor and Re-Enter Monitor
	RESET	Enter and Initialize Monitor
	ESC	Re-Enter Monitor
Text Input/ Output and Update	R	Read Into Text Buffer
	L	List From Text Buffer
	I	Insert One Line
	K	Delete One Line
Line Pointer Positioning and Display	T	Move the Line Pointer to the Top
	B	Move the Line Pointer to the Bottom
	U	Move the Line Pointer Up One
	D	Move the Line Pointer Down One
	SPACE	Display Current Line
Character String	F	Find Character String
	C	Change Character String

2. Enter the text buffer starting address as a hexadecimal number, followed by a RETURN or SPACE. If more than four digits are entered (up to 11) AIM 65 will use only the last four. Entering RETURN or SPACE without a start address will cause the default value of $0200 to be used. If the default address is used or $0200 is entered, AIM 65 will respond with:

```
FROM=0200   TO=
```

3. Enter the Text Buffer ending address as a hexadecimal number, followed by a RETURN or SPACE. If more than four digits are entered (up to 11), AIM 65 will use only the last four. Entering RETURN or SPACE without entering an ending address will cause the last address of continguous installed RAM (starting with address $0200) to be used as the ending address.

The Text Buffer is allocated from the start address specified through the ending address specified. A write/read memory check is made at an address of each page of memory to make sure the specified memory space is available. If an ending address was entered and the memory write fails, the message MEM FAIL will be printed, indicating the first address of the memory page that failed to write, and the system will return to the Monitor. The memory write test is performed every $100 addresses, starting with the entered FROM address. For example, if FROM=0250 and TO=0440, the test is performed at $0350 and $0450. If memory is

installed from $0000 to $03FF, the memory test will fail at location $0450 with the message:

```
<MEM FAIL 0400
```

If an ending address was not specified, i.e., the default ending address used, and the memory write fails, the ending address of the Text Buffer is established as the ending address of the failed page. For example, in the 1K RAM version of AIM 65 the memory write will fail at address $04FF. The default ending address will, therefore, be set at address $0400.

If the default address in the 1K version of AIM 65 is used, or address 0400 is entered, AIM 65 will respond with:

```
FROM=0200   TO=0400
IN=
```

4. After the Text Buffer is initialized, the E command allows the user to go directly into the Editor read mode by asking for the input device. Enter the code of the input device from which the text is to be entered.

```
<E>
EDITOR
FROM=0200  TO=0400

IN=
TOP LINE IN BUFFER
LINE 2
LINE 3
BOTTOM LINE
```

Any time a serious error occurs while using the Editor, the user may return to the Monitor Mode by pressing ESC or depressing the RESET button. The Editor can be re-entered using the T command without any loss of input data.

4.2.2 T Command - Re-enter the Editor

The T command is used to re-enter the Editor from the Monitor, without altering the text in the Text Buffer. The Line Pointer is automatically positioned at the top line.

Use the T command by typing T as response to the Monitor prompt. AIM 65 will respond with:

```
<T>
THE TOP LINE IS DISPLAYED
```

NOTE

> The T command will not operate properly unless the Text Buffer has been previously initialized with the E command.

Example:

```
=<T>
TOP LINE OF TEXT
```

4.2.3 Q Command - Exit the Editor and Re-Enter the Monitor

The Q command is used to exit the Editor and return to the Monitor.

Use the Q command by typing Q. AIM 65 will respond with:

```
=<Q>
<
```

The Monitor is now active as indicated by the Monitor prompt and monitor commands may be entered.

4.2.4 ESC Command - Re-enter the Monitor

The ESC command is used to escape from the Editor and return to the Monitor. The ESC key is examined by the Editor along with the other command keys to determine if a valid Editor command has been entered. A check to determine if the ESC command has been entered is also made at the end of each line of text during the L command (see Section 4.3.4).

Escape from the Editor by typing ESC. AIM 65 will respond with the Monitor prompt:

```
<
```

The Monitor is now active and Monitor commands may be entered.

NOTE

If an L command (List from Text Buffer) is in progress, the check for entry of the ESC command is made only at the end of each line that is listed. The ESC key must be held down long enough for the Editor to sample it in this case.

4.2.5 RESET Command - Enter and Initialize the Monitor

The RESET command is used to interrupt the Editor at any point of operation and re-enter the Monitor. The RESET command causes a hardware interrupt to occur, so any Editor operation in progress will be terminated without completion. See Section 3.2.6 for a complete description of the RESET command.

Use the RESET command by pressing the RESET button. AIM 65 will respond with:

```
ROCKWELL AIM 65

.
```

The Monitor is now active and Monitor commands may be entered.

4.3 TEXT INPUT/OUTPUT AND UPDATE

Four commands allow text to be read into, listed from, and deleted from the Text Buffer on a single- or multiple-line basis.

4.3.1 R Command - Read Lines Into Text Buffer

The R command is used to read multiple lines from an input device into the Text Buffer. Text read into the text buffer is inserted in front of the active line.

There may be a noticeable pause at the end of each line if text is being read in front of already existing lines.

If any attempt is made to read more text than can be stored into the Text Buffer, an END message will be displayed. If this occurs, the Buffer must be expanded by changing the Text Buffer ending address (in $00E5 and $00E6) to a larger value using the Monitor M and / commands.

Use the R command as follows:

1. Type R. AIM 65 will ask for the input device by displaying:

   =<R>
   IN=

2. Enter the code of the input device from which the text will be entered.

A. If the keyboard code (RETURN or SPACE) is entered, AIM 65 will display a character position cursor, (^) to indicate where the next digit will be entered:

```
IN=
^
```

Start entering text from the keyboard, terminating each line with RETURN. An input error may be corrected by entering DEL and re-entering the desired character.

Up to 60 characters may be entered on a line. The first 20 characters are entered from left to right as seen on the display. The cursor will disappear when the 20th character is entered. Starting with character 21, the displayed data will scroll to the left one character position as each new character is entered. Upon entry of the 21st character, the first 20 entered characters will print. Likewise, upon entry of the 41st character, characters 21 through 40 will print. Characters 41 through the end of entered text will print when RETURN is entered.

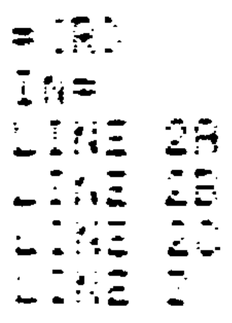
The final line should be terminated with two RETURNs, which will end the Keyboard Read function and allow the Editor to accept a new command.

NOTE

After character 21 or 41 is entered, time must be allowed for the line to be printed before more data is entered. Since the keyboard is not scanned during printing, any character entered during this period may be lost. At normal typing speed no data should be lost; however, if data is typed in rapidly, additional time should be allowed.

B. If the input is from AIM 65 audio tape, see Section 9.1.6. If the input is from TTY punched tape, see Section 9.2.3 for the detail procedure. When the file read is completed, END will be displayed.

Example:



4.3.2 I Command - Insert One Line

The I command is used to insert one line of text ahead of the active line. Input is always from the keyboard.

Use the I command as follows:

1. Use the T, B, U, D, or F command to locate the desired active line.

2. Type I. AIM 65 will respond with:

```
=<I>
^
```

3. Enter the line of text to be inserted. Terminate the entry with a RETURN. AIM 65 will display the next line down.

Example:

Suppose the program in the Text Buffer is (using the T and L commands):

```
=<T>
TOP LINE OF TEXT
=<L>
/
OUT=
TOP LINE OF TEXT
LINE 2
LINE 3
LINE 4
BOTTOM LINE
```

It is desired to add a line before the third line. Using the T, F, and I commands, the new line is inserted:

```
=<T>
TOP LINE OF TEXT
=<F>
3
LINE 3
=<I>
LINE 2A
LINE 3
```

After inserting the new line of text, the updated program is (using the T and L commands):

```
=<T>
TOP LINE OF TEXT
=<L>
1
OUT=
TOP LINE OF TEXT
LINE 2
LINE 2A
LINE 3
LINE 4
BOTTOM LINE
```

4.3.3 K Command - Delete One Line

The K command is used to delete the active line of text.

Use the K command as follows:

1. Locate the line of text to be deleted using the T, B, U, D, or F command.

2. Type K. AIM 65 will respond with:

=<K>

DISPLAY OF NEXT LINE DOWN

Example:

Assume the program in the Text Buffer is (using the T and L commands):

```
=<T>
TOP LINE OF TEXT
=<L>
/
OUT=
TOP LINE OF TEXT
LINE 2
LINE 2A
LINE 3
BOTTOM LINE
```

Locate the line to be deleted (line 2A) using the T and F commands, then delete it with the K command:

```
=<T>
TOP LINE OF TEXT
=<F>
2A
LINE 2A
=<K>
LINE 2A
LINE 3
```

After deleting the desired line of text, the updated program is (using the T and L commands):

```
=<T>
TOP LINE OF TEXT
=<L>
/
OUT=
TOP LINE OF TEXT
LINE 2
LINE 3
BOTTOM LINE
```

4.3.4 L Command - List Lines From Text Buffer

The L command is used to list a specified number of lines in the Text Buffer to an output device, starting at the beginning of the active line.

CAUTION

If the text is to be listed to cassette and read back into a partially-filled text buffer, the interblock gap size in $A409 must be increased to $80.

Use the L command as follows:

1. Type L. AIM 65 will respond with:

```
=<L>
/
```

2. Specify the number of lines to be listed by entering a decimal number from 01 to 99; 00 lists 100 lines, RETURN lists one line, a . or SPACE lists all the lines in the Text Buffer. AIM 65 will ask to which device type the text is to be listed:

OUT=

3. Enter the code of the device to which the text is to be listed:

DESIRED OUTPUT DEVICE	TYPE DEVICE CODE	ADDITIONAL PROCEDURE
AIM 65 Display/ Printer	RETURN or SPACE	
AIM 65 Printer	P	
Audio Tape - AIM 65 Format	T	See Section 9.1.2
TTY Punched Tape	L	See Section 9.2.2
User Defined	U	See Section 7

4. AIM 65 will then proceed to list the contents of the Text Buffer, beginning with the active line and ending with the specified line, to the specified output device. If the specified output device is the audio cassette recorder, a count of each 80 character block will be indicated as it is listed.

CAUTION

If the listed output is to be recorded on audio cassette tape for subsequent input to the assembler or as added text to a partially loaded Text Buffer, the TSPEED parameter in address $A409 must be changed to $80 to provide a large gap time between recorded data blocks. See Section 7.6.

To save the entire contents of the Text Buffer on tape, the T command should be used to position the line pointer at the beginning of the Text Buffer and the L/. or L/SPACE commands used to list all the text lines in the Text Buffer.

Example:

```
=<T>
TOP LINE OF TEXT
=<L>
/
OUT=
TOP LINE OF TEXT
LINE 2
LINE 3
BOTTOM LINE OF TEXT
```

4.4 LINE POINTER POSITIONING AND DISPLAY

Five commands allow moving the line pointer to the top or bottom of the Text Buffer, moving the line pointer up or down one line, and displaying the contents of the active line. In all cases, the line pointer is positioned at the beginning of the line after each command.

4.4.1 T Command - Move The Line Pointer To The Top

The T command is used to move the line pointer to the top line of the text in the Text Buffer and to display the line contents.

If the Text Buffer has been initialized with the E command and no text has been read into the buffer, END will remain displayed.

Use the T command as follows:

Type T. AIM 65 will position the line printer to the top line of the Text Buffer and will display the contents of that line.

Example:

```
=(T)
TOP LINE OF TEXT
```

4.4.2 B Command - Move the Line Pointer to the Bottom

The B command is used to move the line pointer down to the last line of text in the Text Buffer and to display the line contents. If no data has been entered into the Text Buffer, END will be displayed.

To append text to the end of the text currently in the Text Buffer, the line pointer must be moved down one line after the B command is used. Following the B command, the D command will cause the line pointer to be moved down one

line, to a dummy line following the last line of actual text, and to display END. The R or I command may be used at this point to read or insert text ahead of the dummy line.

Use the B command as follows:

Type B. AIM 65 will respond by positioning the line pointer to the last line of text and will display the contents of that line.

Example :

```
=<B>
BOTTOM LINE
```

4.4.3 U Command - Move the Line Pointer Up One Line

The U command is used to move the line pointer up one line and to display the contents of that line. If an attempt is made to ascend past the top line of text, the line pointer will remain on the top line.

Use the U command as follows:

Type U. AIM 65 will respond with:

```
=<U>
DISPLAYED TEXT LINE
```

Example:

```
=< >
LINE 3
=<U>
LINE 2
```

4.4.4 D Command - Move the Line Pointer Down One Line

The D command is used to move the line pointer down one line and to display the contents of that line. If the line pointer is moved down one line from the last line of text, AIM 65 will display END.

Use the D command as follows:

Type D. AIM 65 will respond with:

```
=<D>
DISPLAYED TEXT LINE
```

Example:

```
=< >
LINE 3
=<D>
LINE 4
```

4.5 CHARACTER STRING

Two commands allow text location and manipulation by entry of a character string. Using these commands text can be located, or located and changed, without knowing where the text is located in the Text Buffer.

4.5.1 F Command - Find Character String

The F command is used to find a specified character string, of up to 19 characters long. The search will start at the beginning of the active line and continue until the first occurrence of the string is found or the end of the Text Buffer is encountered. This command is very useful to locate a certain line of text to delete, or a text line for reference prior to an insert or read command, or to determine if a certain character string exists in the Text Buffer.

Use the F command as follows:

1. Type F. AIM 65 will respond with:

```
=<F>
```

2. Enter the character string that is to be found. End the input with RETURN. AIM 65 will display and search for the entered string. If line 2 had been entered, the response will be:

```
=<F>
LINE 2
```

3. A. If the string is found, AIM 65 will display the line that contains the entered character string, and position the line pointer at the beginning of

that line. If the entered character string "Line 2" is found in a line containing "Line 2 of Text", the response will be:

```
=<F>
LINE 2
LINE 2 OF TEXT
```

B. If the string is not found, an END message will be displayed.

   The search can be restarted by entering the T command (moving the line pointer to the top of the Text Buffer) and re-entering the entire F command.

C. If the string is found but is not on the desired line, the search can be continued by entering F followed by RETURN after display of the cursor without re-entering the character string. The line pointer will be positioned at the beginning of the next line and the search will continue as described above.

Example:

```
=<T>
TOP LINE OF TEXT
=<F>
2A
LINE 2A
```

4.5.2 C Command - Change Character String

The C command is used to search for a given character string (up to 19 characters long), and either delete or alter it to a character string of up to 20 characters long. The search will start at the beginning of the active line, and continue until the entered string is found or the end of the Text Buffer is encountered.

Use the C command as follows:

1. Type C. AIM 65 will respond with:

   =<C>

2. Enter the string that is to be changed, and end the input with RETURN. If string LAB 2 had been entered, the response will be:

   =<C>
   LAB2

3. The line containing the first occurrence of the string will be displayed, and the Editor will wait for an indication that the displayed line contains the desired occurrence of the string. If the string LAB 2 is found in a line containing LAB2; COMMENT, the response will be:

   LAB2 ;COMMENT

   If the entered string is not found, the AIM 65 will display an END message.

The change can be restarted by entering the T command (to move the line pointer to the top of the Text Buffer) and re-entering the entire C command.

4. If the displayed line does not contain the correct occurrence of the string, press any key except RETURN to resume the search.

5. When the correct occurrence of the string is found, press RETURN to terminate the search. AIM 65 will respond with:

```
TO=^
```

6. Enter the new character string, ending the input with a RETURN. The old text string will be replaced with the new text string.

If the old string is to be deleted rather than replaced, enter RETURN immediately after the cursor is displayed. If LAB2 is to be changed to NOP, enter NOP. AIM 65 will respond with:

```
TO=NOP
```

Once RETURN is entered, the updated line of text will be displayed:

```
NOP ;COMMENT
```

Example:

```
=<T>
TOP LINE OF TEXT
=<C>
INK
LINK 4
TO=INE
LINE 4
=<T>
TOP LINE OF TEXT
=<L>
/
OUT=
TOP LINE OF TEXT
LINE 2
LINE 3
LINE 4
BOTTOM LINE
```

SECTION 5
THE AIM 65 ASSEMBLER

The process of translating individual computer program instructions written in mnemonic or symbolic form (the source program) into actual processor instructions in machine code form (the object program) is called an assembly. The computer program that performs this translation is an Assembler. The mnemonics and symbols used in writing the source program, commonly called the source code, is called the assembly language. One assembly language instruction translates into one processor instruction. The object program is commonly referred to as the object code.

The optional AIM 65 Assembler is a ROM resident, two-pass assembler. It is supplied as one 4K R2332 ROM that plugs into socket Z24 on the AIM 65 Master Module. The assembler allows both instruction and data addresses to be specified symbolically rather than requiring absolute addresses. During Pass 1 the Assembler determines the values of the symbols. The symbols and their corresponding values are placed in a symbol table for use during Pass 2. The assembler generates the actual object code during Pass 2, using the symbol values from the symbol table to generate addresses and displacements and to create data values. Extensive error checking is performed during Pass 2 to determine if the instructions have been correctly coded. Any detected errors are displayed/printed. The assembly listing is also generated during Pass 2.

To use the Assembler, source code is first prepared using the Editor then stored on audio cassette tape, or passed directly from the Text Buffer in RAM to the assembler. If a teletype is used, the source code may be stored on punched paper tape and read back as the input to the Assembler.

Operation of the Assembler is completely automatic within each pass once you specify such information as where the input source code is coming, where the output object code is to be directed, if a full assembly listing or just an errors-only listing is to be generated.

Assembler directives included in the source code provide additional and overriding instructions for listing generation. When audio cassette tape is used as input, a file linkage capability allows multiple files to be used.

5.1 ASSEMBLER ROM INSTALLATION

Before removing the ROM from its shipping package, be sure to observe the handling precautions in Section 1.4. Turn off power to the AIM 65. Remove the ROM from the shipping package. Inspect the pins to be sure they are straight and free from any shipping or protective foam material. Insert the ROM into socket Z24, being careful to observe the proper device orientation. (See Figure 1-1). Support the Master Module from underneath the socket after the pins are aligned and started into the socket. Press firmly and evenly on the ROM until it is inserted completely into the socket.

5.2 THE SYMBOL TABLE

Working space in RAM must be allocated for symbol table storage during Pass 1. Eight bytes of RAM must be allocated for each uniquely defined symbol in the source code; six bytes for the symbol itself and two bytes for the symbol value. Since each symbol requires eight bytes of memory, the end of the symbol table will be a multiple of eight bytes from the starting address. If the allocated symbol table area is not large enough, the assembler will terminate Pass 1 after displaying the error message:

SYM TBL OVERFLOW

The starting and ending address limits of the symbol table are entered during assembly Pass 1.

The actual starting address will correspond to the entered starting address limit. The actual ending address will be lower than the ending address limit.

The starting and upper limit addresses of the symbol table and the number of symbols in the table can be determined by examining these memory locations:

ADDRESS	PARAMETER	EXAMPLE
$003A	Symbol Table Starting Address Low	$00
$003B	Symbol Table Starting Address High	$03
$003E	Symbol Table Upper Limit Address Low	$FF
$003F	Symbol Table Upper Limit Address High	$03
$000B	Number of Symbols (HEX) High	$00
$000C	Number of Symbols (HEX) Low	$12

The address of the last symbol can be computed by multiplying the number of symbols in the symbol table by eight and adding the result to the starting address.

For example,

$0300 + ($0012 X 8) = $0300 + $0090 = $0390
(Address of the last symbol)

NOTE

If the source code, object code, and symbol table are all to reside in RAM during the assembly, take care to prevent overwriting the source code with either the symbol table or the object code, or the symbol table with the object code. Extreme care must be taken to avoid overwriting the source code; it will have to be read into the Text Buffer again. It is good practice to periodically save the source code on permanent storage media (e.g., audio cassette) to prevent inadvertent loss due to overwriting, editor initialization, or AIM 65 power loss.

5.3 ASSEMBLY CONSIDERATIONS

5.3.1 Memory to Memory Assembly

The actual object code addresses can be examined by printing the assembly listing during Pass 2 without directing the object code to memory. The object code can be directed

either to audio cassette tape or to the dummy device (i.e., no output). If the output is directed to audio cassette, it can be then loaded into memory using the Monitor L command without regard to the previously used symbol table locations. If the object code addresses conflict with the source code, the source code should be saved on audio cassette tape before loading the object code.

If no object code output was generated, and examination of the object code addresses on the assembly listing shows no source code or symbol table address conflicts, Pass 2 can be re-run to direct the object code to memory and not to generate an assembly listing (since one was generated during the first Pass 2).

5.3.2 Tape To Tape Assembly

A program with many symbols may require a major portion or all of RAM for the symbol table. In this case, the source code should be saved on audio cassette before the assembly. Pass 1 and Pass 2 should both be run with the input from audio cassette. The output object code should be directed also to a different audio cassette (see Section 9 for detailed audio cassette operation). The size of the program is now limited only by the RAM memory available to handle the number of symbols in the source code.

5.3.3 User Program Constant Storage

The Assembler uses Page 0 locations 0004 through 00DE and Page 1 locations 0170 through 0183. For this reason, you should not assemble any instructions or constants into these locations when assembling to memory (OBJ? N). User program variables can be assigned to these locations, however, and instructions/constants can be loaded in these locations, after the assembly is complete.

5.4 USING THE ASSEMBLER

Use the AIM 65 Assembler as follows:

1. To enter the assembler, type N after display of the Monitor prompt. AIM 65 will respond with:

```
<N>
ASSEMBLER
FROM=^
```

2. Enter the symbol table starting address in hexadecimal. Terminate the address by typing RETURN. Typing RETURN without entering a value will cause the starting address to default to its previously entered value. The newly entered or default value will be displayed. For example, if 0300 is entered, AIM 65 will respond with:

```
FROM=0300  TO=^
```

CAUTION

Since all of Page 0 is used for assembler variables and Page 1 is reserved for user and AIM 65 Monitor stack usage and for AIM 65 variables, the symbol table starting address must be equal to or greater than 0200.

3. Enter the symbol table ending address in hexadecimal. Terminate the address entry with RETURN. Typing RETURN without entering a value will cause the ending address to default to its previously entered value. The newly entered or default value will be displayed. For example, if 0400 is entered, AIM 65 will respond with:

```
FROM=0300  TO=0400.

IN=^
```

4. Enter the code of the input device that contains the source code. Valid options are:

```
M = Text Buffer in memory (RAM)
T = audio cassette tape
L = TTY punched paper tape
U = user defined peripheral
```

CAUTION

If Pass 1 is to be performed from memory (IN = M), be sure that the symbol table addresses do not conflict with the addresses of the source code in the Text Buffer. Part or all of the source code will be overwritten with the symbol table in this case.

Note that the source code will be displayed as it is being read from the input device.

A. If M is entered, AIM 65 will display M. Go to Step 5.

B. If T is entered, AIM 65 will ask for the file name:

```
IN=T  F=^
```

NOTES

1. In order to use an audio cassette for input, the audio cassette recorder must have a remote control capability, with connections as described in Section 9.1. The Assembler processes source code by operating on a block of 80 bytes at a time. During this time, the cassette recorder is halted using the recorder remote control input after a block of source code is read. The recorder will stop with the cassette tape positioned between blocks of input data. When the assembler has processed the block of 80 bytes, the recorder will be restarted to read another block of data.

2. The source code must have been recorded with the GAP variable in $A408 equal to $80, or larger.

Enter the file name under which the source code was stored. If the file name is less than five digits, end the input with a

RETURN or SPACE. AIM 65 will then ask for the audio cassette recorder number. For example:

IN=T F=SRCE1 T=^

Enter the the audio cassette recorder number (1 or 2) from which the source code is to be loaded. End the input with a RETURN or a SPACE. If 1 was entered, AIM 65 will respond with:

IN=T F=SRCE1 T=1

AIM 65 will search for the specified file name. Upon locating a readable tape file, the file name on tape will be compared to the entered file name. If the file names are not identical, AIM 65 will display the search message and block count of the recorded file as the file passes. If file name PROG1 is read, AIM 65 will respond with:

SRCH F=PROG1 BLK=XX Where XX=the block count.

When the entered file name is located on the tape, The Assembler will continue to Step 5.

C. If L is entered, reading of the source code on punched paper tape from the TTY should be initiated as described in Section 9.2.7.

D. If U is entered, Pass 1 will be initiated using the user defined input as described in Section 7.

5. AIM 65 will ask if the total assembly list or just an errors only list is to be displayed/ printed:

LIST?^

If only errors are to be listed, type N. An errors-only listing will be generated during Pass 2.

If the full assembly listing is to be produced, type Y. The complete assembly listing includes the total source program, reformatted for proper output spacing, the address with each label, the generated object code and any detected errors.

6. AIM 65 will ask where the full assembly or the errors-only listing is to be directed.

LIST-OUT=^

Type one of the following valid options:

RETURN or SPACE = Display/Printer
P = Printer
T = Audio Cassette (AIM 65 Source Code Format)
U = User defined
L = TTY (See Section 9.2)

7. AIM 65 will then ask where the object code is to be directed:

OBJ?^

If the object code is to go directly into memory, type N. Go to Step 6.

CAUTION

If the output is to go into memory, be sure the object code addresses do not conflict with source code in the Text Buffer if the input is from memory or with the symbol table addresses.

If the object code is to be directed to an output device and not to memory, type Y. AIM 65 will respond by asking for the output device code.

OBJ-OUT=^

8. Type one of the following valid option codes:

RETURN or SPACE = Display/Printer
P = Printer only
T = Audio Cassette (AIM 65 Format)
L = TTY (See Section 9.2)
X = Dummy Device (no output)
U = User Defined

NOTE

The selected OBJ-OUT= option must not conflict with the previously selected LIST-OUT option or else both listing and object code output will be directed to the same output device in an intermixed manner.

9. AIM 65 will initiate Pass 1 and display:

   PASS 1

   During Pass 1, the Assembler creates the symbol table. If the allocated symbol table area is too small to store all the symbols, AIM 65 will display SYM TBL OVRFLOW and return to the Assembler entry point.

10. If Pass 1 is completed successfully, AIM 65 will automatically initiate Pass 2 if the input (IN=) is from memory (M) or user-defined (U). If the input is from audio tape (T) or punched tape (L), the Assembler will halt and display

    PASS 2

    Rewind the tape and type SPACE to start Pass 2.

11. Pass 2 will be performed. The selected errors-only or full assembly listing will be directed to the LIST-OUT device. The assembled object code wlll be directed to the OBJ?/OBJ-LIST device. Upon completion of Pass 2, control will be returned to the Monitor.

Any error detected during Pass will be identified by a number corresponding to the error code (see Table 5-1) in this format:

**ERROR XX where XX = 01 to 21.

At the completion of the assembly listing the number of detected errors will be reported.

ERRORS= XXXX where XXXX = the error count.

NOTE

Any .OPT LIS, NOL, ERR or NOE directives in the user program will override the user response to the LIST? prompt.

5.5 ASSEMBLER EXPRESSIONS

Assembler expressions are very useful tools to facilitate programming and to generate both readable and easily changeable code.

There are two components of assembler expressions: elements and operators.

5.5.1 Elements

Elements may be classified into three distinct types: constants, symbols, and the location counter.

Table 5-1. Assembler Error Messages

	SYM TBL OVERFLOW During Pass 1, more unique symbols were detected than allowed in the symbol table. The symbol table length allocation can be enlarged by changing either the symbol table starting and/or ending address by re-entering Pass 1.
01	** UNDEFINED SYMBOL The assembler has found a symbol in an operand expression which is nowhere defined (as a label or as the destination field of an equate directive) in the source code. This error will also occur if a reserved name (A, X, Y, S, or P) is referenced as a symbol in an expression.
02	** LABEL PREVIOUSLY DEFINED The first field on the line, interpreted as a symbol, has been found already defined with a value in the symbol table. A forward reference to a page zero defined symbol has caused a misalignment in address values from Pass 1 to Pass 2.
03	** ILLEGAL OR MISSING OPCODE The assembler has found a line containing a label, followed by an expression, which it tried to interpret as an instruction.
04	** ADDRESS NOT VALID An address referenced in an instruction or in one of the assembler directives (.BYTE, .WORD, or .DBYTE) is invalid.

Table 5-1. Assembler Error Messages (Cont.)

05	** ACCUMULATOR MODE NOT ALLOWED Following a legal instruction mnemonic and one or more spaces, is the letter A followed by one or more spaces (denoting the accumulator addressing mode). The assembler tried to use the accumulator as the operand. However, the instruction in the statement is one which does not allow reference to the accumulator.
06	** FORWARD REFERENCE TO PAGE ZERO An operand expression containing a forward reference has been found.
07	** RAN OFF END OF LINE This error message occurs when the assembler is looking for a needed field and runs off the end of the line image before the field is found.
08	** LABEL DOESN'T BEGIN WITH ALPHABETIC CHARACTER The first non-blank field, being neither a comment nor a valid instruction, is assumed to be a label. However, the first character of the field begins with a numeric character (0-9), violating the rules of symbol construction.
09	** LABEL GREATER THAN SIX CHARACTERS The first field on the line is a string containing more than six characters. Lacking a semicolon prefix, denoting a comment, it is assumed to be a symbol whose length limit has been exceeded.

Table 5-1. Assembler Error Messages (Cont.)

10	** LABEL OR OPCODE CONTAINS NON-ALPHANUMERIC The label or opcode field on a line (illegally) contains a character which is not alphanumeric.
11	** FORWARD REFERENCE IN EQUATE OR ORG The expression on the right side of an equal sign contains a symbol that has not been previously defined.
12	** INVALID INDEX - MUST BE X OR Y A legal operand expression follows an opcode. Following this expression, is a comma (denoting indexed addressing) and an invalid string where either X or Y was expected. This error will be given whether an indexed addressing mode is legal for the corresponding instruction mnemonic or not.
13	** INVALID EXPRESSION While evaluating an expression, the assembler found a character that it could not interpret.
14	** UNDEFINED ASSEMBLER DIRECTIVE If a period is the first character in a non-blank field, the assembler interprets the following three characters as an assembler directive. Either an invalid directive has been found or the first three characters of one of the options in the .OPT directive are uninterpretable.
15	** INVALID PAGE ZERO COMMAND An invalid page zero indexed operand has been detected, for example STA #20,X. An invalid non-page zero indexed operand, for example STA #20, or an invalid page zero operand without indexing will result in error 18.

Table 5-1. Assembler Error Messages (Cont.)

17	** RELATIVE BRANCH OUT OF RANGE A branch instruction can branch only 127 bytes forward or 128 bytes backward. This error message indicates an out-of-range branch.
18	** ILLEGAL OPERAND TYPE FOR THIS INSTRUCTION After finding an instruction mnemonic that does not allow implied addressing, the assembler passes to the operand field and determines what type of operand it is (indexed, absolute, etc.). This error message is printed if the type of operand found is invalid for the instruction.
19	** OUT OF BOUNDS ON INDIRECT ADDRESSING An indirect address is recognized as such by the parentheses that surround it in the operand field of an instruction mnemonic. Since indirect addressing requires two bytes of page zero memory, the address referencing this area must be less than or equal to 254.
20	** A, X, Y, S, AND P ARE RESERVED LABELS One of the five reserved names (A, X, S, X, and P) has been used as a symbol.
21	** PROGRAM COUNTER NEGATIVE - RESET TO 0 An attempt to reference a negative memory location will cause this error message, and the Program Counter to be reset to 0.

5.5.2 Constants

Numeric constants may be written in several bases. The base is specified by the type of prefix character preceding the digits, as defined in the following table.

PREFIX CHARACTER	BASE
(none)	10 (Decimal)
$	16 (Hexadecimal)
@	8 (Octal)
%	2 (Binary)

Some examples are:

```
==0000
          *=$200
==0200
10        BYT $10,10,@
10,%10
0A
08
02
A010F7 LDA $F710
A51D   LDA 29
A57E   LDA @176
A56D   LDA %01101101
```

ASCII LITERAL CONSTANTS

ASCII literal constants, enclosed in quotes, are used to insert the ASCII representation of character strings into memory.

For example:

```
25      .BYT '%','I'
M',''''
492740
==0211
27
A950    LDA #'P
A927    LDA #''''
A935    LDA #'5
```

Note that two quotes are needed to represent a quote in memory. Thus, in the last field of the .BYT directive, the first represents a single quote, and the last closes off the string.

5.5.3 Symbols

Symbols are names used to represent numerical values. They may be from one to six alphanumeric characters long, and the first character must be alphabetic. The 56 valid opcodes (listed in Table 5-2) and the reserved symbols A, X, Y, S, and P have special meaning to the Assembler, and may not be used as symbols.

For example:

```
==0218 VARBLE
       =$2C
==0218 DATA1
AB     .BYT $AB,VARB
LE
2C
==021A LAB190
AD1802 LDA DATA1
A52C   LDA VARBLE
```

5.5.4 Location Counter

The location counter, referenced by the character "*", is a sequential counter used by the Assembler to keep track of its current position in memory. It may be freely used in expressions within a program.

For example:

```
021F    .DBY *
A02102 LDA *
```

5.5.6 Operators

Two arithmetic operators are allowed in the R6500 assembly language:

OPERATOR	OPERATION
+	Addition
-	Subtraction

Evaluation of expressions proceeds strictly from left to right, with no parenthetical grouping allowed; all operators have equal precedence.

In addition, there are two special operators:

CHARACTER	OPERATION
>	High-Byte Selection
<	Low-Byte Selection

Operators < and > truncate a two-byte value to its high or low byte, respectively.

For example:

```
==0224 HIGH
A8      .BYT >$A5CD,<
*,5+<HIGH-%10
25
27
A541    LDA <*+>$1AC2
A51A    LDA %101+7+$7
+07
A503    LDA >HIGH-$40
+<65
```

Expressions which evaluate to negative values are illegal. The twos complement representation of a negative number must be expressed as an unsigned (preferably hexadecimal) constant (e.g., write "-1" as "$FF").

Note especially that expressions are evaluated at assembly time, not at execution time.

5.6 ASSEMBLER SOURCE STATEMENTS

Assembler source statements are comprised of up to four fields:

[label] [opcode [operand]] [;comment]

Brackets surrounding a field indicate that it is optional. Thus, although none of the fields is mandatory, an opcode field must precede an operand field. Input to the Assembler is free form; any field may start in any column.

In particular, note that due to the reserved opcodes, the user is able to precede labels with spaces. If no

label is present, an opcode may be placed in the first column.

Fields in a statement need only be separated by a single space. If the fields are separated in this manner, the Assembler will columnize the fields and produce a readable listing. The user's program may then be stored on audio cassette in a highly compressed form.

Also note that the comment field should be preceded by a semicolon. If the semicolon is omitted, the comment field will not be placed in its proper column in the listing.

5.6.1 Labels

A label is a one- to six-character string of alphanumeric characters. It must begin with an alphabetic character, and must appear as the first field on a line, although it may begin in any column. Using a label is a way to assign the current value of the location counter to the symbol before the rest of the line is processed by the Assembler. Labels are used with instructions as branch targets and with memory data cells for reference in operands.

A line containing only a label is valid, so several labels may be assigned to the same memory location by putting each on a separate line:

```
==0220 SAME1
==0220 SAME2
==022D SAME3
AD2D02 LDA SAME1
```

5.6.2 Opcodes and Operands

Two distinct classes of assembler instructions are available to the programmer: machine instructions and assembler directives.

5.6.3 Machine Instructions

The 56 valid machine instruction mnemonics (Table 5-2) represent the operations implemented on the R6500 family of microprocessors. When assembled, each mnemonic generates one byte of machine code, the actual bit pattern depending upon both the operation specified in the opcode field and the addressing mode determined from the operand field. The operand field may generate one or two bytes of address.

Table 5-2. R6500 Microprocessor Instruction Set--Alphabetic Sequence

	ADC	Add Memory to Accumulator with Carry		LDA	Load Accumulator with Memory
	AND	AND Memory with Accumulator		LDX	Load Index X with Memory
	ASL	Shift Left One Bit (Memory or Accumulator)		LDY	Load Index Y with Memory
				LSR	Shift Right One Bit (Memory or Accumulator)
	BCC	Branch on Carry Clear			
	BCS	Branch on Carry Set	*	NOP	No Operation
	BEQ	Branch on Result Zero			
	BIT	Test Bits in Memory with Accumulator		ORA	OR Memory with Accumulator
	BMI	Branch on Result Minus			
	BNE	Branch on Result Not Zero	*	PHA	Push Accumulator on Stack
	BPL	Branch on Result Plus	*	PHP	Push Processor Status on Stack
*	BRK	Force Break	*	PLA	Pull Accumulator from Stack
	BVC	Branch on Overflow Clear	*	PLP	Pull Processor Status from Stack
	BVS	Branch on Overflow Set			
				ROL	Rotate One Bit Left (Memory or Accumulator
*	CLC	Clear Carry Flag		ROR	Rotate One Bit Right (Memory or Accumulator)
*	CLD	Clear Decimal Mode			
*	CLI	Clear Interrupt Disable Bit	*	RTI	Return from Interrupt
*	CLV	Clear Overflow Flag	*	RTS	Return from Subroutine
	CMP	Compare Memory and Accumulator			
	CPX	Compare Memory and Index X		SBC	Subtract Memory from Accumulator with Borrow
	CPY	Compare Memory and Index Y	*	SEC	Set Carry Flag
	DEC	Decrement Memory by One	*	SED	Set Decimal Mode
*	DEX	Decrement Index X by One	*	SEI	Set Interrupt Disable Status
*	DEY	Decrement Index Y by One		STA	Store Accumulator in Memory
				STX	Store Index X in Memory
	EOR	Exclusive-OR Memory with Accumulator		STY	Store Index Y in Memory
	INC	Increment Memory by One			
*	INX	Increment Index X by One	*	TAX	Transfer Accumulator to Index X
*	INY	Increment Index Y by One	*	TAY	Transfer Accumulator to Index Y
			*	TSX	Transfer Stack Pointer to Index X
	JMP	Jump to New Location	*	TXA	Transfer Index X to Accumulator
	JSR	Jump to New Location Saving Return Address	*	TXS	Transfer Index X to Stack Pointer
			*	TYA	Transfer Index Y to Accumulator

*Instructions legal only in the implied addressing mode.

5.7 OPERAND ADDRESSING MODES

5.7.1 Absolute Addressing

The absolute addressing mode is the most common in concept; the data following the machine code is treated as the address of a memory location containing the actual data to be processed during the instruction step. This address is stored in reverse order--as low-byte, then high-byte--to increase processing efficiency during execution

For example:

```
==0230 PIA
       =$4C1F
==0230 LATCH
       =$4DE2
==0230 BUFF1
       =$5400
==0230 START
       =$A000
==0230 EXTRTN
       =$C800
201F4C BIT PIA
CDD7C2 CMP $C2D7
CEBA54 DEC BUFF1+10
4DB054 EOR BUFF1
4C00A0 JMP START
2000C8 JSR EXTRTN
==0242
AD2F6C LDA %11011000
01011111
6ECF11 ROR $11CF
EDCF54 SBC BUFF1+$1F
8DE24D STA LATCH
```

5.7.2 Page Zero Addressing

In practice, the zero page addressing mode (identical in concept to absolute addressing) is the most frequently used. This allows the expression of the instruction to be two bytes instead of three; the low byte of the data address is taken from memory, and the high byte is assumed to be zero. All instructions legal in absolute mode are also legal in zero page mode, with the exception of the JMP and JSR instructions (see Table 5-2); the Assembler automatically generates the shortest possible code. It is good programming practice to reserve page zero (memory locations 0-255) for declaration of variables.

NOTE

Any variables on page zero must be defined before they are referenced.

For example:

```
==024E MODE
       =$06
==024E KEY
       =$0C
==024E COUNTR
       =$37
==024E TTYBUF
       =$6B
6537   ADC COUNTR
247A   BIT $7A
C406   CPY MODE
E638   INC COUNTR+1
A66B   LDX TTYBUF
469A   LSR $21+0171
050C   ORA KEY
86AA   STX TTYBUF+$3
F
```

5.7.3 Immediate Addressing

The immediate mode of addressing is coded by the character "#" followed by a byte expression; the code inserted into memory is treated as the data to be operated upon according to the machine code.

For example:

```
==025E DOLLAR
       =$24
==025E LAB1
6903   ADC #3
29B5   AND #%10111010
1
E024   CPX #DOLLAR
A945   LDA #'E
A05E   LDY #<LAB1
```

5.7.4 Implied Addressing

Twenty-five of the fifty-six instructions, legal only in the implied addressing mode, require no operand--their execution may be completed with no other information than that contained in the opcode. These instructions are preceded by an * in Table 5-2.

For example:

```
00     BRK
D8     CLD
C8     INY
EA     NOP
68     PLA
60     RTS
8A     TXA
```

5.7.5 Accumulator Addressing

Instructions implementing the four shift operations have, in addition to addressing modes referencing memory, a special mode which allows manipulation of the accumulator. Usage of this mode, similar to implied addressing, causes generation of a single byte of machine code.

For example:

```
0A      ASL A
4A      LSR A
2A      ROL A
6A      ROR A
```

5.7.6 Relative Addressing

Eight conditional branch instructions are available to the programmer; normally these immediately follow load, compare, arithmetic, and shift instructions. Branch instructions uniquely use the relative addressing mode. The branch address is a one-byte positive or negative offset, expressed in twos complement notation, from the run-time program counter. At the time the branch address calculation is made, the program counter points to the first memory location beyond the branch instruction code. Hence, the one-byte offset limits access to branch addresses within 129 bytes forward and 126 bytes backward from the beginning of the branch code. (A one-byte twos complement number is limited to the range -128 to 127 inclusive.) An error will be flagged at assembly time if the branch target lies out-of-bounds for relative addressing.

For Example:

```
9000    BCC *-125
==0275 HERE
F0FE    BEQ HERE
30FC    BMI *-2
707F    BVS *+129
```

5.7.7 Indexed Addressing

Indexed addressing (with Index Registers X or Y) facilitates certain types of table processing. The address given as the operand is treated as the base address, to which the contents of either the X or the Y Register is added to arrive at the effective address of the memory location containing the data to be operated upon. All instructions implementing absolute indexed addressing with the X Register also allow the same addressing in the page zero mode; several instructions (LDX, LDY, STX, and STY) allow zero page indexed addressing with the Y Register.

For example:

```
==027B ARRAY
       =$0B
==027B NUMBUF
       =$50
==027B TABLE
       =$2200
794F00 ADC NUMBUF-1,
Y
DDC022 CMP TABLE,X
D60B    DEC ARRAY,X
5DCC22 EOR TABLE+$C,
X
B60B    LDX ARRAY,Y
3622    ROL >TABLE,X
968F    STX NUMBUF+$3
F,Y
```

5.7.8 Indirect Addressing

The concept of indirect addressing constitutes a level of complexity beyond that of absolute addressing. The operand address references not one memory location containing data, but a sequence of two memory locations containing the address--stored in low-byte, high-byte order--of the location containing the actual data to be processed. True indirect addressing is offered only with the JMP instruction; otherwise, indexed indirect addressing with the X Register and indirect indexed addressing with the Y Register are implemented. For indexed indirect addressing, the indexed address is computed before the indirect is taken; the order of evaluation is reversed for indirect indexed addressing.

NOTE

Normal indirect addressing takes place when the Index Register contains zero. The JMP indirect uses an absolute-length (two-byte) operand; others require the operand address to lie in page zero between 0 and 254 inclusive.

For example:

```
==028C INDADR
       =$02
==028C CURSOR
       =$57
==028C OLDPTR
       =$7E
==028C NEXT
       =$D9
2102   AND (INDADR,X
)
D17E   CMP (OLDPTR),
Y
6CD900 JMP (NEXT)
B157   LDA (CURSOR),
```

```
V
E102    SBC (INDADR,X
)
8157    STA (CURSOR,X
)
```

5.8 ASSEMBLER DIRECTIVES

There are nine assembler directives. They are used to set symbol and location counter values (=), reserve and initialize memory locations (.BYTE, .WORD, .DBYTE), and control both assembler input/output (.OPT, .FILE, .END) and assembler listing format (.PAGE, .SKIP). All may be considered as assembly time instructions, rather than as execution-time instructions.

5.8.1 Equate Directive

The equate ("=") directive assigns the value of an expression containing no forward references (symbols defined in a following section of code) to either a symbol or the location counter:

```
==0299
        *=$1800
==1800 TABLE2
        =$C800
==1800 WRDPTR
        =$2A
==1800 NUMPTR
        =WRDPTR+2
```

A label used with an equate directive which increments the location counter will reserve work area memory locations; this is especially useful when consecutively allocating uninitialized memory at the beginning of a program:

```
==1300
        *=0
;EQUATE DIRECTIVE
==0000 EOT
        *=*+1
;RESERVE MEMORY
==0001 EOTADR
        *=*+2
==0003 BUFFER
        *=*+72
==004B ENDFLG
        *=*+1
```

Symbols assigned one-byte values may be programmed as assembler constants--assembly-time values, used consistently throughout a program, which may be changed at a later time when the program is reassembled. Source code is designed so that alteration simply requires reassignment of the corresponding assembler constants. This is considered good programming practice and is a much better alternative to changing each constant as it occurs throughout a program:

```
==004C
        *=0
;EQUATE DIRECTIVE
==0000 STRTCH
        =$26
;ASSEMBLER CONSTANTS
==0000 ENDCH
        =$2B
==0000 DELIM
        =$2C
==0000 LOWCH
        =$41
==0000 HIGHCH
        =$5A
==0000 KEYLEN
        =4
==0000 BUFLEN
        =72
```

5.8.2 .BYTE Directive

The .BYTE directive initializes byte memory locations. Multiple arguments, separated by commas, may be specified in

a single .BYTE command to load consecutive memory locations; either ASCII strings or expressions evaluating to an eight-bit value are legal. ASCII strings in .BYTE direc.ives must not generate more than 20 characters:

```
==0000 ASCII
4142    .BYT 'ABCD','
EFH','JOE''S'
454640
4A4F
00      :BYT <ASCII,>
ASCII+2-*1,<*+>*,2
01
0E
02
```

Note the use of two quotes within an ASCII string to insert a single quote into memory.

5.8.3 .WORD Directive

The .WORD directive is very useful in constructing jump tables and initializing pointers. An operand expression is evaluated as a two-byte address and is stored in low-byte, high-byte sequence, the order in which the microprocessor fetches addresses from memory. As with .BYTE, multiple operand fields, separated by commas, are allowed:

```
==0010 JMPTBL
0409    WOR $0804,$E
489,$F77A
89E4
7AF7
0200    WOR $2,<JMPT
BL,>JMPTBL,*,06371
1000
0000
1C00
F90C
```

5.8.4 .DBYTE Directive

If it is desired to generate a sixteen-bit expression value in normal high-byte, low-byte order, the .DBYTE assembler directive must be used. Its syntax rules are the same as those for .WORD:

```
==0020 DATA
C804    .DBY $C804,$E
489,$F77A
E489
F77A
000E    .DBY $E,<DATA
0>DATA,*,%0101101111
01
0029
0000
002C
05BD
```

5.8.5 .PAGE Directive

The .PAGE directive causes a title line to be printed under a dashed line. A title may be specified as an ASCII string in the operand field, and it may be cleared with a string of one or more blanks. Absence of an operand will also cause the title to be cleared. This command is not printed as entered in the source code--only the results appear. For example, entry of:

```
.PAGE 'ORIGINAL TITL
E'
.PAGE
.PAGE 'NEW TITLE'
.PAGE
```

would cause the following to appear at the top of each page:

```
----------------------
ORIGINAL TITLE

----------------------

----------------------
NEW TITLE

----------------------
```

5.8.6 .SKIP Directive

A blank line may be inserted in the program listing with the .SKIP directive.

```
        *=2
         SKI
CURSOR  *=*+2
EOT     *=*+2
         SKI
TWO     =EOT
```

This causes the following listing to be printed:

```
                *=0

==0000 CURSOR
                *=*+2
==0002 EOT
                *=*+2

==0004 TWO
                =EOT
```

5.8.7 .OPT Directive

The three options of the .OPT directive control generation of output files and expansion of ASCII strings in .BYTE directives. These options are selected by specifying:

```
.OPT  LIST, GENERATE, ERRORS
```

and are eliminated by coding:

```
.OPT  NOLIST, NOGENERATE, NOERRORS
```

Since only the first three characters of each option are scanned, the following may be written:

```
.OPT  LIS, GEN, ERR
.OPT  NOL, NOG, NOE
```

Of these options, only GEN/NOG remains unspecified after entering Pass 2; GEN/NOG has a default value of NOG.

The three options have the following functions:

1. LIST (NOLIST) controls generation of the program listing, which contains assembled source input, generated object code, errors and warnings.

2. GENERATE (NOGENERATE) controls the printing of object code for ASCII strings in the .BYTE directive. Only code for the first two characters is listed if NOG is specified; otherwise, the whole literal will be expanded.

3. ERRORS (NOERRORS) controls the listing of only erroneous program source lines together with the respective messages generated. Fatal assembler table overflows are also messaged in this file.

5.8.8 .FILE Directive

For large programs, it is usually more convenient to divide the source program into logical segments which may be separately loaded into the Editor Text Buffer and edited. After editing, each file is listed from the Text Buffer to a separate file on audio cassette tape. However, when the entire program is to be assembled, it is necessary to tie these files together. This function is performed with the .FILE assembler directive. Each file (except the last) contains, as its last record, a .FILE directive which points to the next file in the chain.

```
.FILE  NAME
```

For example, if the first file is named PRGM, then:

```
.FILE  QARC            is the last statement in file
                       PRGM
.FILE  DEF             IS THE LAST STATEMENT IN FILE
                       QARC
.FILE  PATCH           is the last statement in file
                       DEF
```

5.8.9 .END DIRECTIVE

The last statement of the last file in the source program must be the .END directive. For example:

```
.END
```

is the last statement in a one-file program and the last statement of the last file in a multiple-file program.

5.9 COMMENTS

Comments may be freely inserted into source code following the last field in a line. If preceded by an opcode (and possibly operand) field, the comment may optionally begin with a semicolon (;). Otherwise, the semicolon is necessary. A comment may be the only field on a line.

For Example:

```
==0000
        *=$0300
==0300
A900    LDA #0
;COMMENT FOLLOWING S
EMI-COLON
A900    LDA #0 COMMEN
T NOT FOLLOWING SEMI
-COLON
```

SECTION 6
R6500 PROGRAMMING CONCEPTS

This section provides an introduction to the programming of the R6502 microprocessor at the machine or assembly level. The R6500 Programming Manual and the references listed at the end of the section contain more detailed information. Our intention here is to show you how to perform common tasks with the R6502. You should use this section as a first step in R6502 programming or as a brief overview. The last part of the section discusses the overall problems of software development and lists some useful references.

6.1 PROGRAMMING TASKS

Let us first describe some of the tasks we would like the R6502 to perform. These include:

- Simple calculations or functions, such as addition, subtraction, logical AND, etc.

- Decision making, i.e., determining whether inputs or the results of calculations have certain values or are above or below certain levels. The R6502 should be able to choose a course of action depending on these decisions--after all, that is what intelligence is all about.

- Looping, i.e., repeating tasks a specified number of times or until some condition is satisfied.

- Array handling, i.e., processing groups of data items such as sensor readings, test inputs, control outputs, or strings of characters.

- Code conversion and manipulation, i.e., handling data that is or must be coded in some form such as BCD, ASCII, seven-segment, or Gray. Clearly this task is an essential part of obtaining data from input peripherals and sending data to output peripherals.

- Arithmetic, i.e., performing such functions as multiple precision addition and subtraction, multiplication, division, and square root.

Finally, we need some way of connecting the short programs that perform these tasks to our main program. Of course, this discussion will only touch on how to write and connect short programs--we will not touch here on how to define your problem, design the program, and debug, test, and document it. We will briefly discuss those topics later. Note also that Section 8 describes I/O programming with the R6522 Versatile Interface Adapter.

6.2 R6502 REGISTERS AND FLAGS

We have previously reviewed the R6502 registers and flags in Section 2.6. The registers are: (See Figure 6-1.)

Program counter (PC)

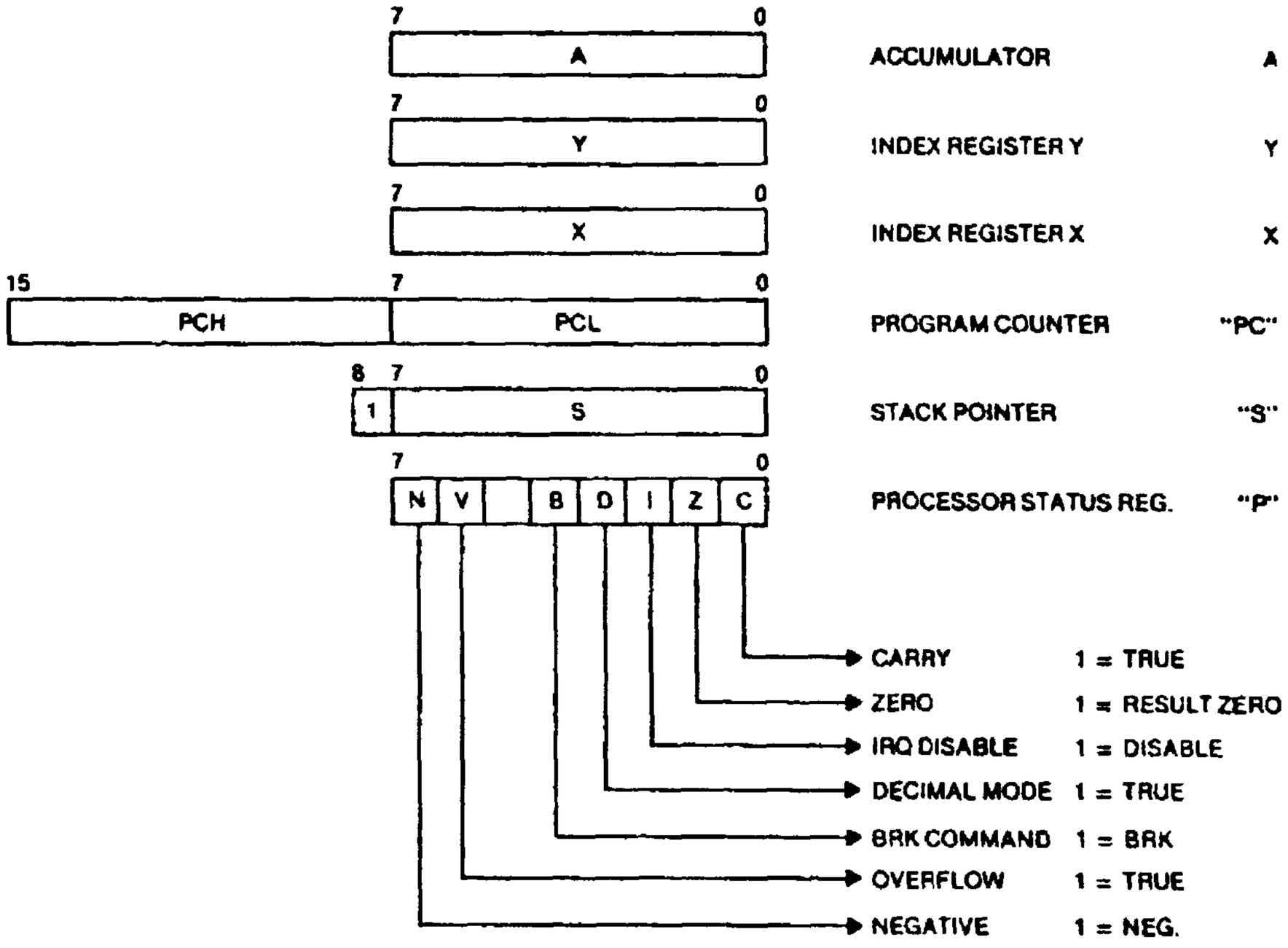


Figure 6-1. R6502 Registers and Flags

Accumulator (A)
Index Registers X and Y
Processor status (P)
Stack pointer (S)

e flags are:

Negative (N) or sign
Overflow (V)
Break command (B)
Decimal mode (D)
Interrupt disable (I)
Zero (Z)
Carry (C)

Ne may describe the uses of the various registers as follows:

PC contains the address of the next instruction to be executed. PC is incremented each time it is used; jump and branch instructions place a new value in this register.

A holds one operand (and the result) in arithmetic and logical operations. It is the center of processing activities.

X and Y generally hold counters for looping or indexes for handling tables or arrays.

S holds the address (on page 1) of the top of the RAM stack used for saving subroutine return and addresses and the previous contents of registers and flags.

The uses of the common flags are described as follows. Note that we generally use these flags by testing their values with conditional jump instructions.

N is used to check status bits and sign values.

Z is used to check for equality (after a subtraction), for a zero value in a counter, and for a zero value in a bit (after a logical AND).

C is used to save carries or borrows in arithmetic operations, to determine the result of numerical comparisons, and to determine the value of a bit (after a shift operation).

6.3 SIMPLE OPERATIONS

Most simple operations take place in the R6502 accumulator. The instructions available include:.

- addition (ADC)
- subtraction (SBC)
- logical AND (AND)
- logical OR (ORA)
- logical EXCLUSIVE OR (EOR)

All of these instructions assume that one operand is in the accumulator and the other is in memory. The result is saved in the accumulator.

Three steps are necessary for the CPU to perform an arithmetic or logical operation on two numbers. Each of these steps corresponds to a separate R6502 instruction:

1. Load the accumulator with one operand.

2. Perform the operation on the accumulator and the other operand.

3. Save the result (from the accumulator) in memory.

Thus, the R6502 works just like a simple hand calculator in which you enter one operand, perform the operation with the other operand, and save the result for later use.

NOTE

All arithmetic and logical operations are performed on two 8-bit numbers at a time. Operations involving more operands or longer operands require a series of instructions. The CARRY bit must be used to transfer carries or borrows between successive additions and subtractions.

Examples:

1. Logically AND the contents of memory locations 40 and 41 place the result in 42.

(R6500 assembly format)

```
LDA     $40     GET FIRST OPERAND
AND     $41     LOGICAL AND
STA     $42     STORE RESULT
BRK
```

Examples (cont.):

(AIM 65 mnemonic format)

```
LDA             40
AND             41
STA             42
BRK
```

Here we use the R6500 Assembler notation in which $ means "hexadecimal". (The $ is not used when you enter instructions with the AIM 65 I command.)

2. Clear the four most significant bits of memory location 40 and place the result in 41.

(R6500 assembly format)

```
LDA             $40     GET OPERAND
AND             #$0F    CLEAR 4 MSB'S
STA             $41     STORE RESULT
BRK
```

Here we use the R6500 Assembler notation in which # means "immediate", i.e., the number that follows is data rather than a memory address.

(AIM 65 mnemonic format)

```
LDA             40
AND             #0F
STA             41
BRK
```

Examples (Cont.)

3. Set the most significant bit of memory location 40 and place the result in 41.

(R6500 assembly format)

```
LDA             $40     GET OPERAND
ORA             #$80    SET MSB
STA             $41
BRK
```

Remember that anything logically ORed with a '1' produces a result of 1.

(AIM 65 mnemonic format)

```
LDA             40
ORA             #80
STA             41
BRK
```

Note that a few simple operations can be performed directly on memory. These operatons are:

Increment (add 1)
Decrement (subtract 1)
Arithmetic shift left (move all bits left by one position and clear LSB)
Logical shift right (move all bits right by one position and clear MSB)

Rotate left or right (move all bits including the CARRY by one position as if the ends of the number were connected through the CARRY).

There are also shift and rotate instructions for the accumulator.

6.4 MAKING DECISIONS

In the simple programs of the previous discussion, the R6502 acts like an inexpensive hand calculator. The conditional branch instructions make the R6502 into an intelligent controller, capable of performing different actions based on input data or the results of calculations. Table 6-1 summarizes these instructions.

The R6502 executes conditional branch instruction as follows:

- If the condition is met, a new value is placed in the program counter and the R6502 executes the instruction at that address next.

- If the condition is not met, the program counter is not changed and the R6502 continues executing instructions sequentially.

Figure 6-2 is a flowchart of the conditional branch.

The common uses of the conditional branch instructions are as follows:

BCC	(BRANCH IF CARRY CLEAR)
BCS	(BRANCH IF CARRY SET)

Table 6-1. R6502 Conditional Branch Instructions

INSTRUCTION	FLAG USED	VALUE ON WHICH BRANCH OCCURS
BCC	CARRY	0
BCS	CARRY	1
BNE	ZERO	0
BEQ	ZERO	1
BPL	NEGATIVE	1
BMI	NEGATIVE	1
BVC	OVERFLOW	0
BVS	OVERFLOW	1

If the specified flag does not have the indicated value, no branch occurs.

1. After a comparison to determine if one value is greater than or less than another. Typical sequences and their effects are as follows:

```
CMP             OTHER
BCC             LARGE
```

causes a branch to address LARGE if (A) < (OTHER).

```
CMP             #THRSH
BCS             ABOVE
```

causes a branch to address ABOVE if (A) ≥ THRSH.

Note that after a CMP instruction, C = 0 if the number in A is less than the other number (unsigned). Table 6-2 shows the states of the major flags (N, Z, C) after a CMP, CPX, or CPY instruction.

Table 6-2. Compare Instruction Results

CONDITION	N	Z	C
A, X, or Y < Memory	1*	0	0
A, X, or Y = Memory	0	1	1
A, X, or Y > Memory	0*	0	1
*N is valid only for two's complement			

2. After an addition or subtraction to determine if a carry or borrow was produced. Typical sequences are:

```
ADC             NEXT
BCS             RIPPLE
```

causes a branch to address RIPPLE if the addition results in a carry.

```
SBC             FACTOR
BCS             DONE
```

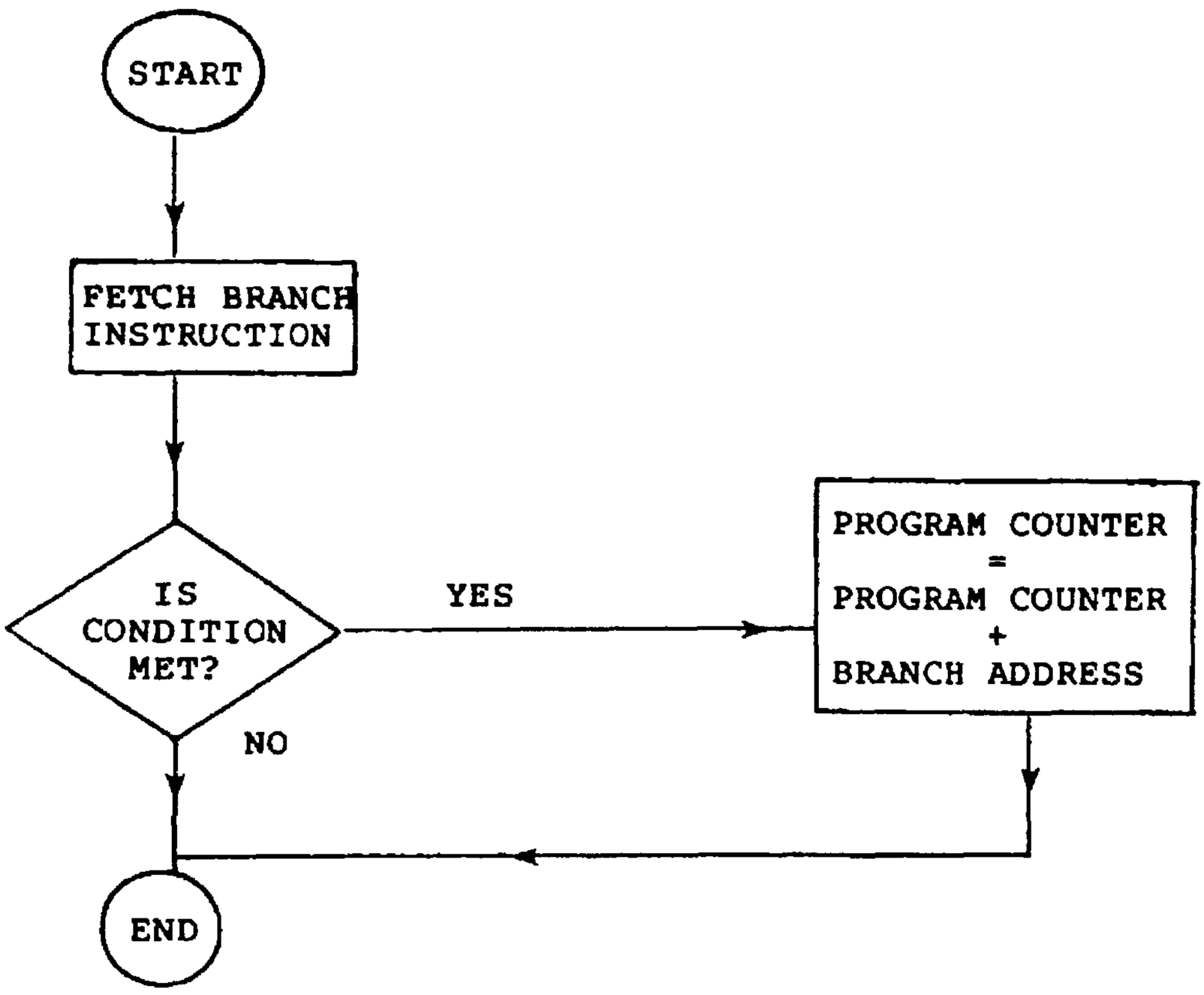


Figure 6-2. Flowchart of a Conditional Branch

causes a branch to address DONE if the subtraction does not require a borrow. Note that a borrow is needed if no CARRY is generated. (The logic for borrows is the opposite of the logic for carries.)

3. After a shift to determine the value of a particular bit. Typical sequences and their effects are:

```
ASL             MULT
BCC             NOADD
```

causes a branch to address NOADD if the most significant bit of the contents of MULT is zero.

```
ROR             INPUT
BCS             LAST
```

causes a branch to address LAST if the least significant bit of the contents of INPUT is one.

```
BEQ             (BRANCH IF Z = 1)
BNE             (BRANCH IF Z = 0)
```

NOTE

Remember, Z = 1 if the previous result was zero and Z = 0 if the previous result was not zero.

1. After a comparison, to determine if two values are equal. Typical sequences are:

```
CMP             OTHER
BNE             DIFF
```

causes a branch to address DIFF if (A) ≠ (OTHER).

```
        CMP             #KEY
        BEQ             FOUND
```

causes a branch to address FOUND if (A) = KEY. Note again the difference between ordinary (or direct) addressing and immediate addressing.

2. After a decrement or increment to determine if a counter has been decremented or incremented to zero. Typical sequences are:

```
        DEX
        BNE             LOOP
```

causes a branch to address LOOP if register X has not been decremented to zero.

```
        INC             LSIG
        BEQ             RIPPL
```

causes a branch to address RIPPL if the result of incrementing LSIG was zero. Note that INC does not affect the C (or CARRY) bit.

3. After a logical AND to determine the value of a particular bit. Typical sequences are as follows:

```
AND             #$40
BEQ             NOT1
```

causes a branch to address NOT1 if bit 6 of the accumulator is zero.

```
AND             #$08
BNE             BITON
```

causes a branch to address BITON if bit 3 of the accumulator is one.

```
BMI             (BRANCH ON N = 1)
BPL             (BRANCH ON N = 0)
```

4. To check the value of the most significant bit. Typical sequences are as follows:

```
LDA             FLAGS
BMI             FOUND
```

causes a branch to address FOUND if the most significant bit of FLAGS is 1.

```
LDA             SWCHS
BPL             CLSD
```

causes a branch to address CLSD if the most significant bit of SWCHS is 0.

The availability of the N bit makes bit position 7 special even if it is not being used for a sign. Since its value can be determined directly (without a shift or logical instruction), bit position 7 is often used for serial I/O and for the most frequently interrogated switches and user flags. Bit positions 0 and 6 are also somewhat special because of the shift instructions.

Let us now look at a few simple examples of decisions:

1. Find the larger value of memory locations 40 and 41, and store the value in 42.

(R6500 assembly format)

```
        LDA     $40     GET FIRST VALUE
        CMP     $41     IS SECOND VALUE LARGER?
        BCS     DONE
        LDA     $41     YES, GET SECOND VALUE
DONE    STA     $42     STORE LARGER VALUE
        BRK
```

(AIM 65 mnemonic format)

```
        LDA     40
        CMP     41
        BCS     02
        LDA     41
        STA     42
        BRK
```

The branch is taken if (40) $\geq$ (41). Note that in comparisons and subtractions the R6502 sets C if no borrow is necessary and clears C if a borrow is required.

2. Set memory location 42 to 1 if the contents of 40 and 41 are equal, and to 0 otherwise.

(R6500 assembly format)

```
        LDX             #0      MARK = ZERO
        LDA             $40     GET FIRST VALUE
        CMP             $41     IS SECOND VALUE THE SAME?
        BNE             DONE
        INX                     YES, MARK = ONE
DONE    STX             $42     STORE MARK
        BRK
```

The branch is taken if (40) $\neq$ (41).

(AIM 65 mnemonic format)

```
LDX             #00
LDA             40
CMP             41
BNE             01
INX
STX             42
BRK
```

3. Set memory location 41 to 1 if bit 5 of memory location 40 is 1 and to 0 otherwise.

(R6500 assembly format)

```
        LDX             #0      MARK = ZERO
        LDA             $40     IS BIT 5 OF DATA 1?
        AND             #$20
        BEQ             DONE
        INX                     YES, MARK = ONE
DONE    STX             $41     STORE MARK
        BRK
```

(AIM 65 mnemonic format)

```
        LDX             #00
        LDA             40
        AND             #20
        BEQ             01
        INX
        STX             41
        BRK
```

The branch is taken if bit 5 of memory location 40 is zero.

4. Set memory location 41 to 1 if bit 7 of memory location 40 is 1 and to 0 otherwise.

(R6500 Assembler format)

```
        LDX             #0
        LDA             $40
        BPL             DONE
        INX
```

```
DONE  STX          $41
      BRK
```

The branch is taken if bit 7 of memory location 40 is zero.

AIM 65 mnemonic format)

```
LDX          #00
LDA          40
BPL          01
INX
STX          41
BRK
```

NOTE

In actual machine language programs, all conditional branches use relative addresses that specify how much to add to the program counter, if the branch is taken. The relative address is an 8-bit two's complement number that allows branches of -128 to +127 locations from the end of the branch instruction. You can get longer branches by using the unconditional JMP instruction and a little inverted logic.

```
        BEQ          NEXT
        JMP          FAR
NEXT
```

causes a branch to memory address FAR if Z ≠ 0.

6.5 LOOPS

The simplest way to have the R6502 microprocessor repeat a section of a program a specified number of times is:

1. Before entering the section to be repeated, initialize a counter to a specified value.

2. After completing the section, decrement the counter.

3. If the result of Step 2 is not zero, return to the start of the section.

Clearly, the branch instruction will be BNE. You can place the counter in register X, in register Y, or in a memory location. Figure 6-3 is a flowchart of the logic for a program loop.

The basic loop structure then is (using register X):

```
        LDX             #COUNT
LOOP    .
        .
        .
        .
        .
        DEX
        BNE             LOOP
```

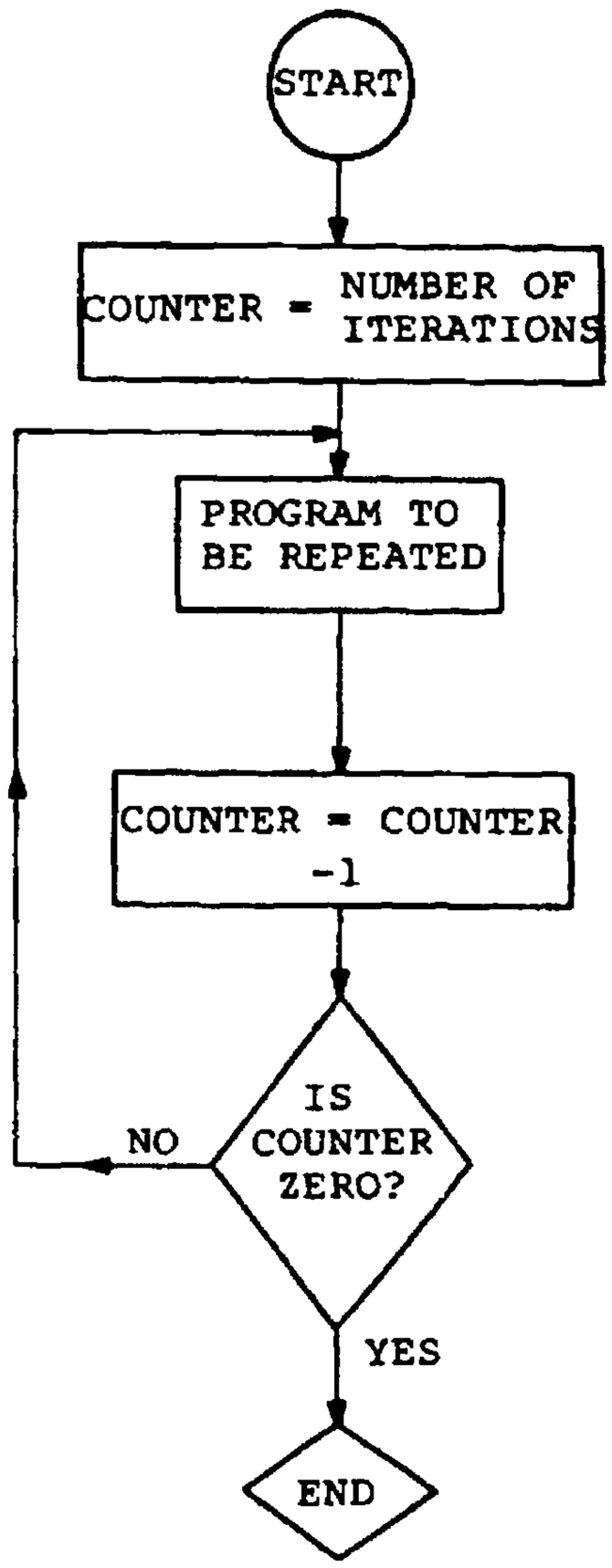


Figure 6-3. Flowchart of a Program Loop

An alternative approach is to count up instead of down, i.e.,

```
        LDX             #0
LOOP    .
        .
        .
        .
        INX
        CPX             #COUNT
        BNE             LOOP
```

One extra instruction (CPX) is necessary, but this method may be more convenient within the program itself.

Note that the looping method is essentially the same as marking off iterations by hand. You must be careful that you only initialize the counter once but that you increment it or decrement it each time through the loop. Of course, you can place loops within loops (called nesting) as long as you keep the counters separate and handle the branches properly.

6.6 DATA ARRAYS

The key to processing data arrays with the R6502 microprocessor is to use an index register to hold the element number. Remember that we need two things to describe an element in any array.

1. The starting or base address of the entire array.

2. The element number or index.

Remember that we can characterize an element in an array as A_i where A is the name of the array and i is the element number.

We can use the R6502's indexed addressing to process arrays. In this addressing mode, the R6502 adds the contents of an index register (X or Y) to the address in the instruction to get the actual data address.

Example:

If (X) = 06, the instruction AND $2000, X results in

(A) = (A) • (2000 +(X))
= (A) • (2006)

Note that X and Y are 8-bit registers so that the offset can never be larger than 256.

An important point is that we can change X or Y (with INX, INY, DEX, or DEY) and then repeat the instruction. Each iteration gets the data from a different address in memory. The actual address of the data is called the effective address. Note that we can change the effective address even if the instruction is in a read-only memory.

Let us now look at some simple examples:

1. Form a checksum in memory location 40 by EXCLUSIVE ORing the contents of memory locations 41 through 48.

(R6500 assembly format)

```
        LDA             #0      CHECKSUM = 0
        LDX             #0      COUNT = 0
CHSUM   EOR             $41,X EXCLUSIVE OR AN ELEMENT
        INX
        CPX             #8      OUT OF ELEMENTS?
        BNE             CHSUM NO, KEEP FORMING CHECKSUM
        STA             $40     YES, SAVE CHECKSUM
        BRK
```

(AIM 65 mnemonic format)

```
        LDA             #00
        LDX             #00
        EOR             41,X
        INX
        CPX             #08
        BNE             F9
        STA             40
        BRK
```

Of course, we could count down just as easily as up. Which do you think is better?

2. Count all the zeroes occurring in memory locations 41 through 48. Place the result in memory location 40.

(R6500 Assembler format)

```
        LDX             #0      ELEMENT COUNT = 0
        LDY             #0      ZERO COUNT = 0
```

```
CHZRO LDA       $41,X IS NEXT ELEMENT ZERO?
      BNE       CTELM
      INY             YES, INCREMENT ZERO COUNT
CTELM INX
      CPX       #8
      BNE       CHZRO
      STY       $40
      BRK
```

(AIM 65 mnemonic format)

```
      LDX       #00
      LDY       #00
      LDA       41,X
      BNE       01
      INY
      INX
      CPX       #08
      BNE       F6
      STY       40
      BRK
```

3. Clear all the memory locations starting with 41 and ending with 48.

(R6500 assembly format)

```
      LDA       #0    GET ZERO
      LDX       #0    COUNT = ZERO
CLR1  STA       $41,X CLEAR A LOCATION
      INX
      CPX       #8
```

```
      BNE             CLR1
      BRK
```

(AIM 65 mnemonic format)

```
      LDA             #00
      LDX             #00
      STA              41,X
      INX
      CPX             #08
      BNE              F9
      BRK
```

Note the use of INX (or DEX) and CPX. Remember that INX and DEX can only increment or decrement X by 1. For larger steps you must either repeat the INX or DEX instruction or perform the required arithmetic in the accumulator. There are no instructions for performing arithmetic in X or Y.

6.7 CODE CONVERSION AND MANIPULATION

You can use the R6502 microprocessor to handle data in any code. The processor itself only handles numerical values. It does not care what the meaning of the data is to peripheral devices or to an operator. So the processor handles control characters, letters, digits, etc. all in the same way. The programmer must provide the intelligence that interprets specific values properly.

Examples:

1. Set memory location 41 to 1 if memory location 40 contains a valid ASCII digit and to 0 otherwise. The ASCII code represents the decimal digits as 30 through 39 hex.

(R6500 assembly format)

```
        LDX             #0
        LDA             $40     GET DIGIT
        CMP             #$30    IS IT BELOW ASCII ZERO?
        BCC             DONE    YES, NOT VALID
        CMP             #$3A    IS IT ABOVE ASCII NINE?
        BCS             DONE    YES, NOT VALID
        INX                     VALID IF IN-BETWEEN
DONE    STX             $41
        BRK
```

(AIM 65 mnemonic format)

```
LDX             #00
LDA             40
CMP             #30
BCC             05
CMP             #3A
BCS             01
INX
STX             41
BRK
```

Remember that the logic for borrows is the inverse of the logic for carries.

2. Count the number of ASCII E's (45 hex) in memory locations 41 through 48. Place the count in memory location 40.

(R6500 assembly format)

```
        LDX     #0      ELEMENT COUNT = 0
        LDY     #0      E COUNT = 0
        LDA     #$45    GET ASCII E
CHKE    CMP     $41,X   IS NEXT ELEMENT AN E?
        BNE     CTELM
        INY             YES, INCREMENT E COUNT
CTELM   INX
        CPX     $8
        BNE     CHKE
        STY     $40
        BRK
```

(AIM 65 mnemonic format)

```
LDX             #00
LDY             #00
LDA             #45
CMP             41, X
BNE             01
INY
INX
CPX             #08
BNE             F6
STY             40
BRK
```

As for code conversions, some (like ASCII to decimal or decimal to ASCII) can be performed with simple arithmetic instructions. More complex conversions involving seven-segment code, Gray codes, etc. can be handled with look-up tables. These tables are just data arrays that are stored in memory in a convenient order. For example, the ith element in the array could simply contain the code corresponding to element i. If the table starts at address CDTAB, the program to convert an element in A into its coded equivalent is:

```
TAX
LDA          CDTAB,X
```

Note that the lookup table procedure does not depend at all on what is in the table. In fact, many of the entries could be the same if you are willing to waste some memory. If the table does not change, you could even make it part of the program or store it in a read-only memory. Lookup tables are discussed more completely in J.B. Peatman, Microcomputer-based Design, McGraw-Hill, 1977, Chapter 7 and in L.A. Leventhal, "Cut Your Processor's Computation Time", Electronic Design, August 16, 1977, pp. 82-88.

6.8 ARITHMETIC

The only explicit arithmetic instructions available on the R6502 are:

ADC - Add with Carry, i.e., (A) = (A) + (M) + C

SBC - Subtract with Borrow, i.e., (A) = (A) - M - 1+C

More complex arithmetic operations, such as multiplication, division, or square root, must be implemented either:

1. As a series of additions, subtractions, and shifts, some algorithms are in

T.C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios, and Square Roots", IBM Journal of Research and Development, July 1972, pp. 380-388.

A.M. Despain, "Fourier Transform Computers Using CORDIC Iterations", IEEE Transactions on Computers, October 1974, pp. 993-1001.

H.C. Schmid, Decimal Computation, Wiley, New York, 1974.1.

2. With lookup tables. For example, see:

T.A. Seim, "Numerical Interpolation for Microprocessor-based Systems", Computer Design, February 1978, pp. 111-116.

3. In special hardware as discussed in S. Waser, "State-of-the-art in High-Speed Arithmetic Integrated Circuits", Computer Design, July 1978, pp 67-75.

Note the following features of R6500 arithmetic instructions:

1. The CARRY is always included. If you do not want its value to affect the operation, you must explicitly

clear it with CLC before an addition or set it with SEC before a subtraction.

2. All additions and subtractions are decimal (BCD) if D = 1. But remember to clear D (CLD) when you are through with the decimal arithmetic.

3. You can use ASL A to add the accumulator to itself.

4. You can use ADC #0 or SBC #0 to add C to the accumulator or subtract 1 minus C from the accumulator.

5. Both ADC and SBC are 8-bit operations in which the result ends up in A.

6. The V bit tells if two's complement overflow has occurred, i.e., if the magnitude of the result has affected its sign bit.

Some simple examples are:

1. Add the contents of memory locations 40 and 41, and place the result in 42.

(R6500 assembly format)

```
        CLC
        LDA     $40     GET FIRST OPERAND
        ADC     $41     ADD SECOND OPERAND
        STA     $42     STORE SUM
        BRK
```

(AIM 65 mnemonic format)

```
CLC
LDA             40
ADC             41
STA             42
BRK
```

2. Add the 16-bit number in memory locations 40 and 41 (MSB's in 41) to the 16-bit number in memory locations 42 and 43 (MSB's in 43), and place the result in 44 and 45 (MSB's in 45).

(R6500 assembly format)

```
CLC
LDA             $40     ADD LSB'S
ADC             $42
STA             $44
LDA             $41     ADD MSB'S
ADC             $43
STA             $45
BRK
```

(AIM 65 mnemonic format)

```
CLC
LDA             40
ADC             42
STA             44
LDA             41
ADC             43
STA             45
BRK
```

Note that we need the carry from the least significant bits to include in the addition of the most significant bits.

3. Add the 64-bit number starting in memory location 40 (LSB's first) to the 64-bit number starting in memory location 50 (LSB's first). Save the sum in memory locations 60 on, starting with the least significant bits.

(R6500 assembly format)

```
      CLC              CLEAR CARRY TO START
      LDY       #08    BYTE COUNTER = 8
      LDX       #00    INDEX = 0
ADDW  LDA       $40,X  ADD 8 BITS
      ADC       $50,X
      STA       $60,X  STORE 8 BITS
      INX
      DEY
      BNE       ADDW
      BRK
```

(AIM 65 mnemonic format)

```
      CLC
      LDY       #08
      LDX       #00
      LDA       40,X
      ADC       50,X
      STA       60,X
      INX
      DEY
```

```
      BNE               F8
      BRK
```

Note that INX and DEY do not affect the CARRY but CPX or CPY would.

6.9 SUBROUTINES

The simplest way to make a routine commonly available from any point in other programs is to make it into a subroutine. This requires:

1. A JSR (Jump To Subroutine) instruction in the main program to transfer control to the subroutine.

2. An RTS (Return From Subroutine) instruction at the end of the subroutine to transfer control back to the main program.

NOTE

Only one copy of the subroutine has to be in memory. Note also that control is automatically transferred back to the instruction immediately following the JSR which sent the processor to the subroutine.

How does the processor know where to return? The answer is that it uses the RAM stack and the stack pointer. JSR and RTS work as follows:

JSR stores the 8 most significant bits of the program counter at the address given by the stack pointer, decrements the stack pointer, similarly stores the 8 least significant bits of the program counter, and decrements the stack pointer again.

Symbolically, what happens is described by:

```
(0100 + (S))  = (PCH)
        (S)   = (S) - 1
(0100 + (S))  = (PCL)
        (S)   = (S) - 1
```

RTS performs the opposite function. It increments the stack pointer and fetches the program counter from those locations, i.e.:

```
  (S)  =  (S) + 1
(PCL)  =  (0100 + (S))
  (S)  =  (S) + 1
(PCH)  =  (0100 + (S))
```

Note the following features of JSR and RTS:

1. The stack is always on page 1. That is, the 8 most significant bits of the address used are always 01 hex, while the 8 least significant bits are given by the contents of register S.

2. The stack grows down in memory, i.e., from higher addresses to lower addresses. Note that the stack area is just ordinary read/write memory; the only thing that moves is the value in the stack pointer.

3. The overall program must initialize the stack pointer and manage the stack.

4. Subroutines can use other subroutines, since the old return addresses are saved in the stack. Note that the first address entered is the last one retrieved.

5. Register contents can also be saved in the stack using PHA and PHP (plus TXA and TYA for X and Y) and restored using PLA and PLP.

6.10 THE TASKS OF PROGRAM WRITING

Of course, developing programs involves far more than just writing short sequences of instructions. The software designer also must:

- Define the problem.
- Design the program.
- Debug, test, and document the program.

These activities typically take far longer than the writing of instructions. A common rule of thumb is that project time is spent as follows:

- 40% definition and design
- 20% writing instruction (or coding)
- 40% debugging testing, and documentation

We do not have the space here to do justice to any of these stages, let alone all of them. If you are serious about programming, we suggest that you consult the following sources:

Chapin, N., Flowcharts, Auerbach, Princeton, N.J., 1971

Hughes, J. K. and J. I. Michtom, A Structured Approach to Programming, Prentice-Hall, Englewood Cliffs, N.J., 1977

Kerninghan, B. W. and P. J. Plauger, The Elements of Programming Style, McGraw-Hill, New York, 1974.

Leventhal, L. A., Introduction to Microprocessors: Software, Hardware, Programming, Prentice-Hall, Englewood Cliffs, N.J., 1978, Chapter 6

McGowan, C. L. and J. R. Kelly, Top-Down Structured Programming Techniques, Petrocelli/Charter, New York, 1975

Schneider, J. M. et. al., Introduction to Programming and Problem-Solving with Pascal, Wiley, New York, 1978

Yourdon, E., Techniques of Program Structure and Design, Prentice-Hall, Englewood Cliffs, N.J., 1976

6.11 REFERENCES ON R6502 PROGRAMMING

Butterfield, J. et. al., The First Book of KIM, Hayden, Rochelle Park, N.J., 1978

Foster, C. C., Programming a Microcomputer: 6502,
Addison-Wesley, Reading, Ma., 1978

Zaks, R., Microcomputer Programming: 6502, Sybex,
Berkeley, Ca., 1978

SECTION 7
AIM 65 SYSTEM DESCRIPTION

This section describes the AIM 65 hardware and software. The hardware is segmented into logical functional areas for ease of description. The AIM 65 Monitor, Editor and Assembler software are also described. User available subroutines are described along with the calling procedures and conditions.

7.1 OVERVIEW

Aim 65 is a complete microcomputer system. It contains an R6502 CPU, programmed instructions in ROM, RAM, and peripheral equipment in the form of a display, a printer and a keyboard. On- and off-board expansion capabilities enhance the usability of AIM 65. True application ease is provided by a user dedicated R6522 Versatile Interface Adapter.

The major components are shown on the AIM 65 block diagram in Figure 7-1.

7.2 FUNCTIONAL AREAS

The hardware functional areas are

Power Distribution

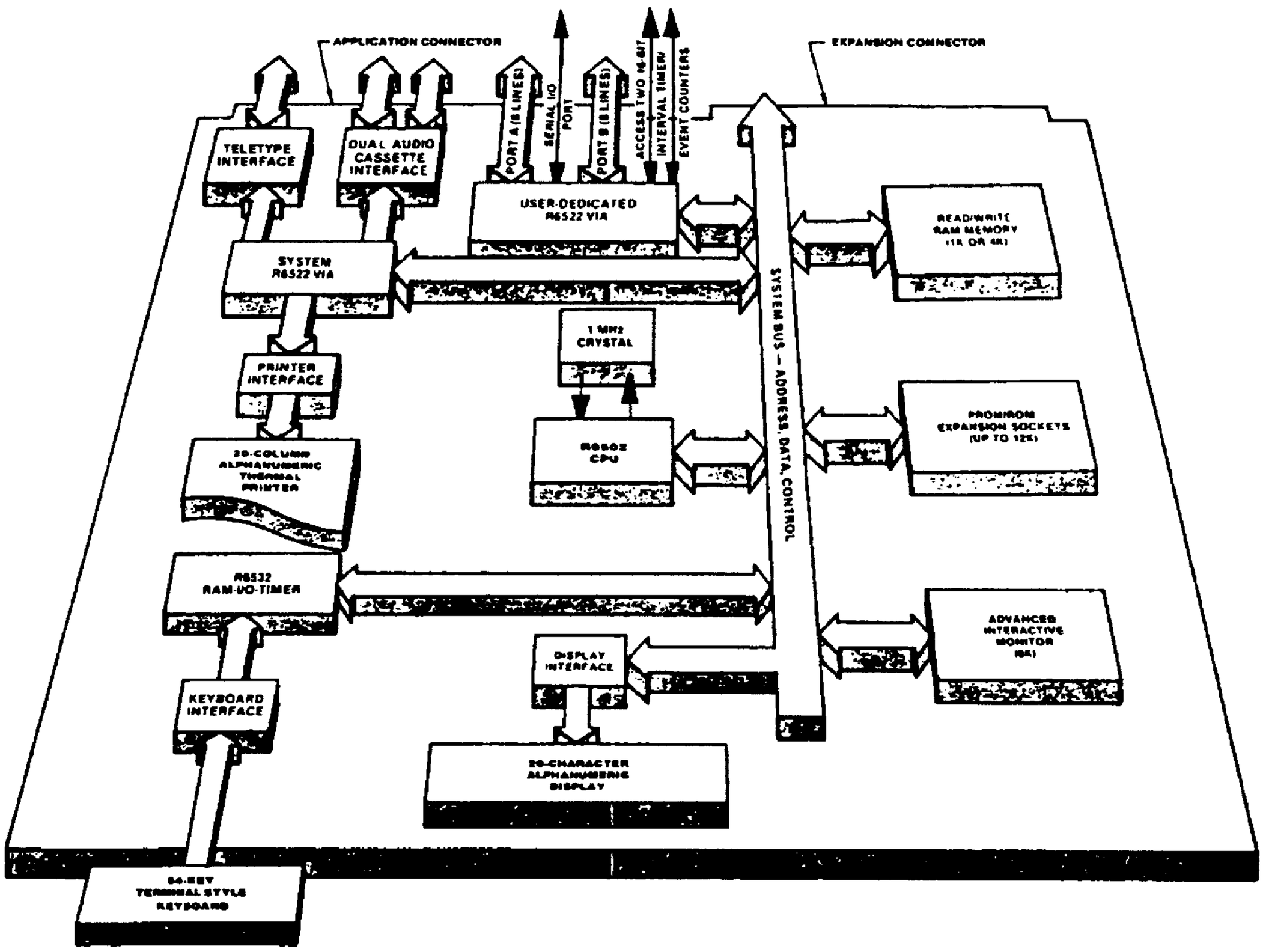


Figure 7-1. AIM 65 Block Diagram

Timing and Control
Chip Select
RAM
ROM
Printer Interface
Display Interface
Keyboard Interface
User R6522 Interface
Audio Cassette Recorder Interface
TTY and Serial Interface

7.2.1 Power Distribution

Power is routed from TB1 terminal strip connections to on-board devices and to interface connectors. Figure 7-2 shows the power distribution. +5V is required for AIM 65, audio, and TTY operation. +5V is also routed from TB1-3 to both the Expansion and Application Connectors.

+24V, required for AIM 65 printer and TTY interface circuitry operation, is routed from TB1-6 to the printer and TTY interface circuitry on the Master Module only. A jumper will extend it to Pin Z on the Application Connector.

+12V and -12V are not required for AIM 65 operation, but are routed from TB1 to the Expansion and Application Connectors for external enhancements. -12V requires a jumper for extension to the Application Connector.

7.2.2 Timing and Control

The Timing and Control area includes the R6502 CPU, the address, data and control bus lines, the R6502 clock

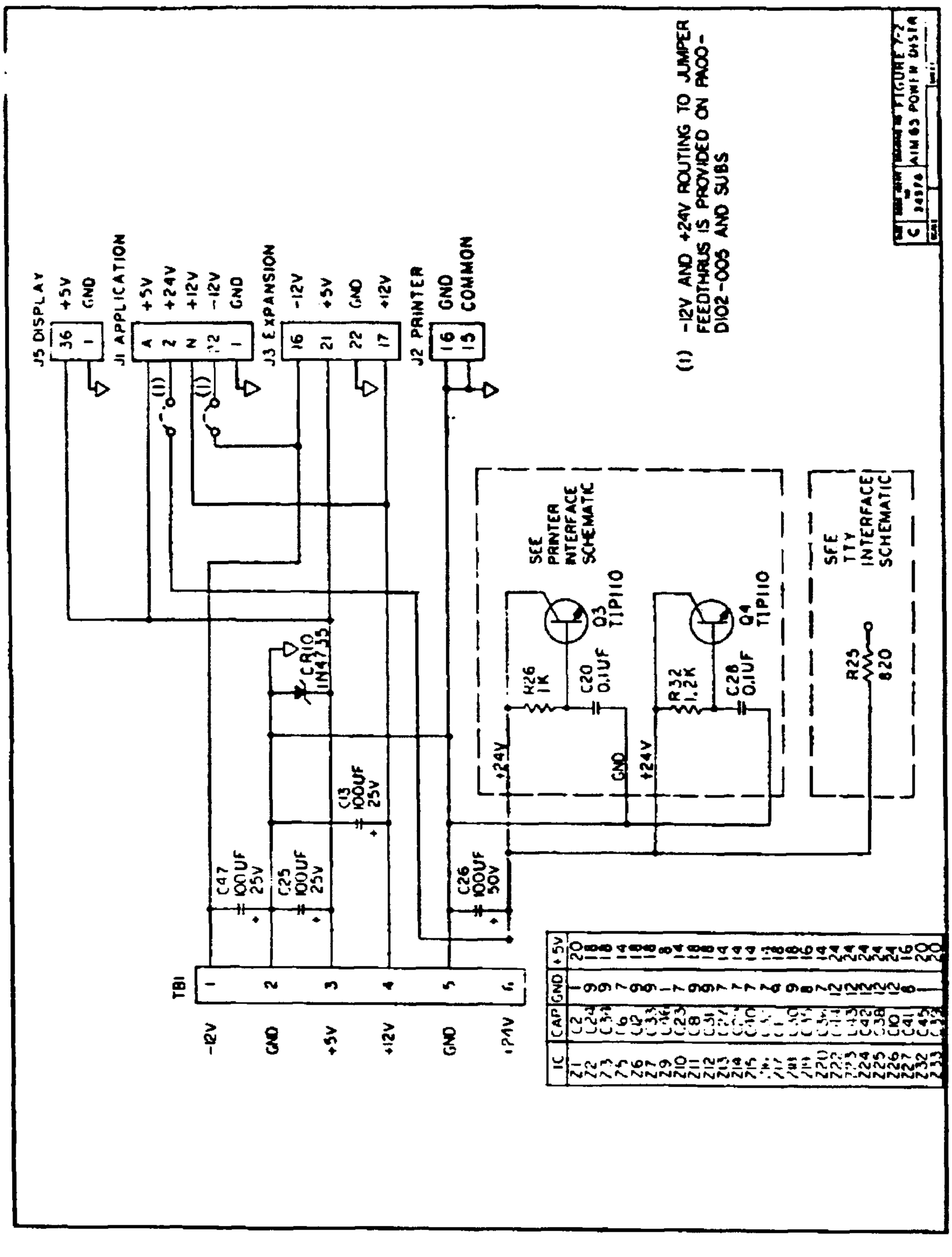


Figure 7-2. AIM 65 Power Distribution Schematic

circuitry, the Monitor R6522 VIA (Z32) interface and the AIM 65 control switches (the RESET pushbutton, the RUN/STEP switch and the KB/TTY switch). Figure 7-3 shows the Timing and Control elements.

R6502

The R6502 8-bit microprocessor, the central processing unit (CPU) of the AIM 65, provides the overall control and monitoring of all AIM 65 operations.

The R6502 communicates with other AIM 65 elements on three separate buses. A 16 bit address bus allows the CPU to directly address 65,536 memory locations. An 8-bit bidirectional data bus carries data from the R6502 CPU to/from memory and interface devices. The control bus carries various timing and control signals between the R6502 CPU and interfacing peripherals, devices and off-board elements.

Section 2 in the R6500 Hardware Manual describes the operation of the R6502 and the bus lines.

R6502 Clock

The R6502 on AIM 65 operates at 1 MHz. The frequency reference is a 4 MHz crystal controlled oscillator. Dual D-type flip-flop Z10 divides the 4 MHz signal by four to drive the R6502 phase 0 (ø0) input with a 1 MHz clock.

The R6502 generates the Phase 1 (ø1) and Phase 2 (ø2) clock outputs based on the Phase 0 input clock. The ø1 (OUT) is routed to J3-3 for external use.

The ø2 (OUT) from the R6502 is routed to J1-C and to inverter J16-9. An ø2 signal provided by J16-8 is routed to J3-Y and Z16-11. A buffered ø2 clock (SYS ø2) generated by Inverter Z16-10 provides the system level timing reference for on-board and expansion use (at J3-U).

R/$\overline{W}$

The Read/Write (R/$\overline{W}$) signal controls the direction of data transfers between the R6502 and interfacing devices. The R/$\overline{W}$ signal is routed to J1-D and Inverter Z16-3. A buffered R/$\overline{W}$ signal from Z16-6 provides the system level R/$\overline{W}$ signal (SYS R/$\overline{W}$) for on-board and expansion use (at J3-V).

CONTROL SWITCHES

RESET

Pushbutton switch S1 initiates RESET of the AIM 65 hardware and software. Timer Z4 holds the $\overline{RES}$ low for at least 15 ms from the time the pushbutton is released. $\overline{RES}$ is routed to the R6502 CPU, the Monitor R6522 (Z32), the Monitor R6532 RIOT (Z33), the user R6522 VIA (Z1), and the display R6520 PIA (U1). To initiate the device RESET function is also routed to the expansion connector for off-board RESET functions. The Monitor performs a software reset when the $\overline{RES}$ line goes high (see Section 7.3.1).

KB/TTY

The position of Switch S3 (KB/TTY) tells AIM 65 to accept commands from either the AIM 65 or the TTY keyboard. This switch is sampled through the Monitor R6522.

STEP/RUN

Switch S2 (STEP/RUN) causes AIM 65 to operate either in the RUN mode or the single STEP mode. In the STEP mode, the $\overline{\text{NMI}}$ interrupt line is driven low when SYNC and ø2 go high during instruction execution if the address lines are outside the A000-FFFF range. The $\overline{\text{NMI}}$ interrupt occurs on the high to low transition of the $\overline{\text{NMI}}$ line. The Monitor software will trace instructions and register, outside the Monitor instruction address range if the trace modes are selected and the NMI Interrupt Routine is not bypassed (see Section 7.7).

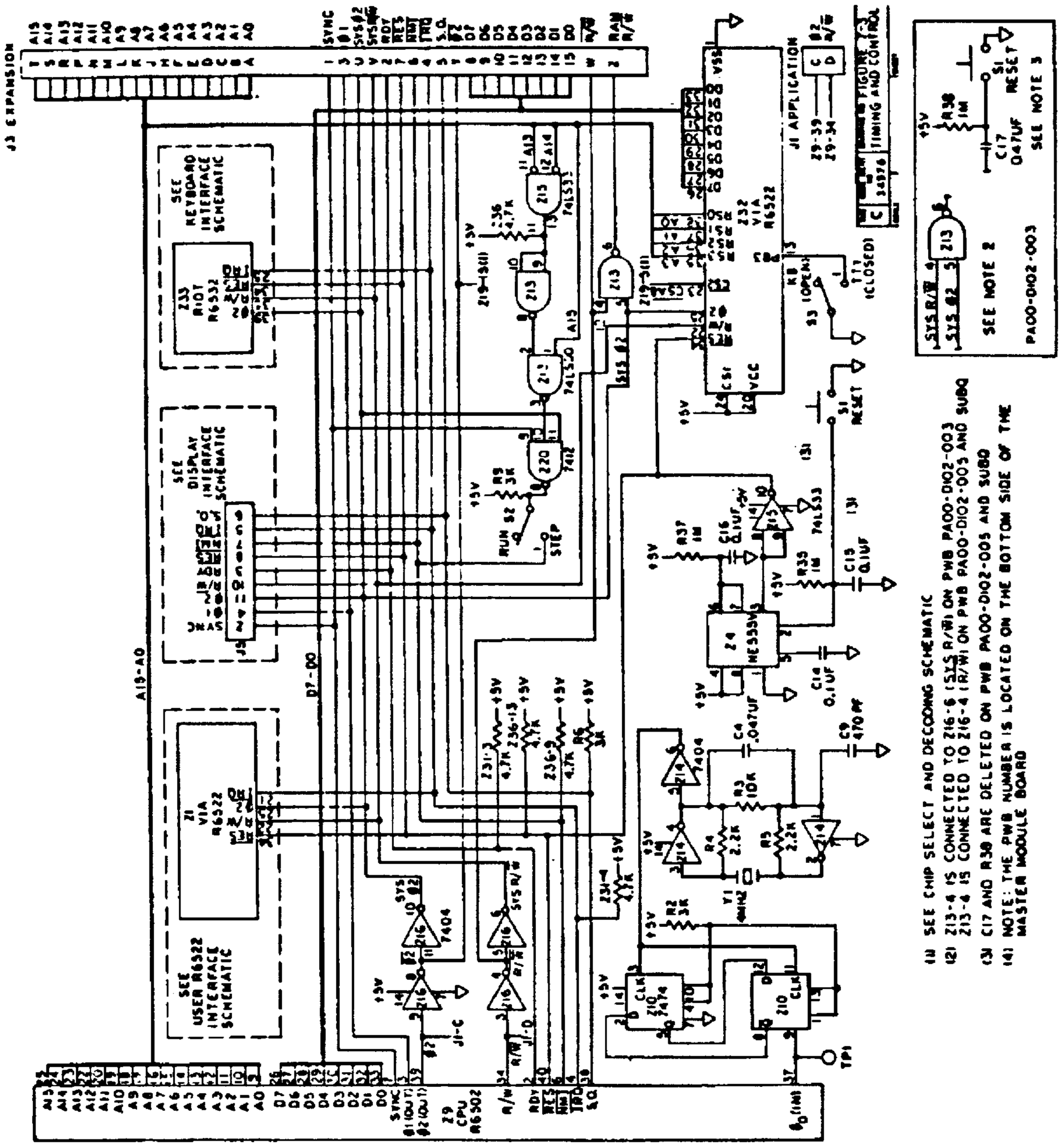


Figure 7-3. Timing and Control Schematic

7.2.3 Chip Select

The chip select function is performed by Decoders Z27 (1 of 8) and Z19 (Dual 2 to 4). See Figure 7-4.

Z27 decodes the upper four address lines (A12-A15) into one of eight possible chip select output lines. Each output line is active over a 4K address range, as shown in Table 7-1. Table 7-2 is the logic table for the Z27 device.

$\overline{CS8}$, $\overline{CS9}$ and $\overline{CSA}$ are routed to the J3 Expansion Connector for off-board use. $\overline{CSA}$ is routed to Z19 to enable the basic Master Module I/O device address select. Any use of $\overline{CSA}$ off-board for additional chip select must not conflict with the on-board use of the A000-AFFF address range (See the AIM 65 Detail Memory Map in Table 7-11.)

$\overline{CSB}$ - $\overline{CSF}$ are routed to the on-board PROM/ROM sockets Z26 - Z22, respectively. Each installed PROM/ROM may use the total 4K address range selected by the respective chip select line. Socket compatible PROM devices with a smaller address range may be directly used, however. In this case the available address range extends from the lowest available address in the range to the upper limit of the installed PROM/ROM.

Z19 performs address decoding for on-board RAM and I/O peripheral chip selection. Z19 contains two independent decoders, each decoding two input lines into one of four possible output lines. Each of the two sides of Z19 decode a selected address range of 1K. Table 7-1 shows the specific address ranges corresponding to the Z19 output chip select lines.

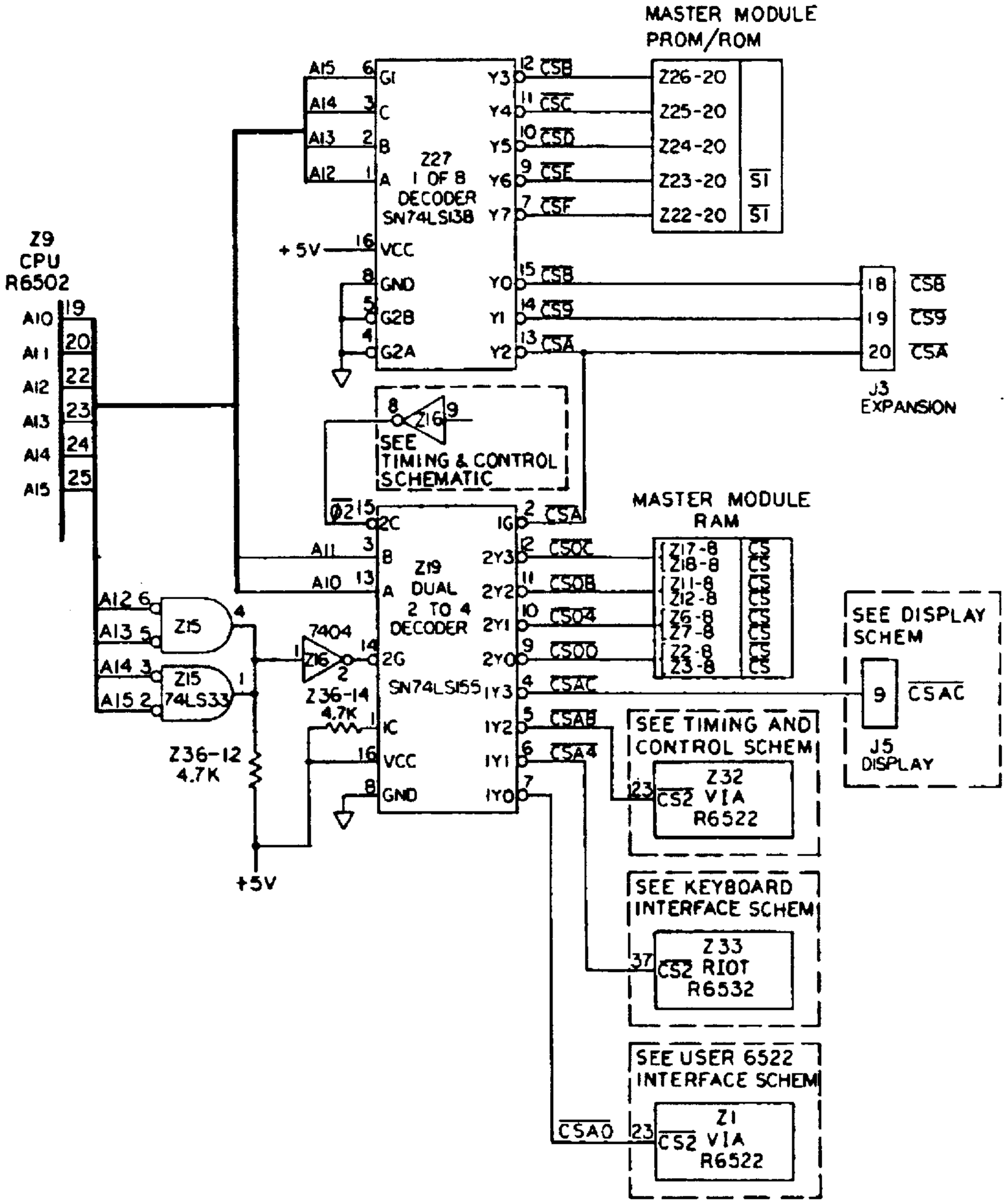


Figure 7-4. Chip Select and Decode Schematic

Side 1 of Z19 performs the individual I/O device select function (see Table 7-3). When $\overline{CSA}$ is low, address lines A10 and A11 are decoded to drive one of the $\overline{CSA0}$-$\overline{CSAC}$ output lines to the active low state. This divides the 4K I/O address range into four 1K segments. Each 1K address segment is allocated to one of the four on-board peripheral I/O devices. See Table 7-7 for the I/O memory allocation.

Side 2 of the Z19 outputs the on-board RAM chip select signals (see Table 7-4). When A12, A13, A14, and A15 are low, A10 and A11 are decoded to drive one of the four $\overline{CS00}$-$\overline{CS0C}$ output lines to the active low state. Each chip select line is routed to two on-board RAM sockets.

Table 7-1. Chip Select Logic Table

ADDRESS RANGE	INPUT ADDRESS LINES																OUTPUT															
																	ON-BOARD RAM SELECT				EXPAND SELECT		ON-BOARD I/O SELECT					ON-BOARD ROM SELECT				
LOW - HIGH	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	$\overline{CS0C}$	$\overline{CS0B}$	$\overline{CS04}$	$\overline{CS00}$	$\overline{CSB}$	$\overline{CS9}$	$\overline{CSA}$	$\overline{CSAC}$	$\overline{CSAB}$	$\overline{CSA4}$	$\overline{CSA0}$	$\overline{CST}$	$\overline{CSE}$	$\overline{CSD}$	$\overline{CSC}$	$\overline{CSB}$
0000-03FF	L	L	L	L	L	L	X	X	X	X	X	X	X	X	X	X	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
0400-07FF	L	L	L	L	L	H	X	X	X	X	X	X	X	X	X	X	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
0800-0BFF	L	L	L	L	H	L	X	X	X	X	X	X	X	X	X	X	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
0C00-0FFF	L	L	L	L	H	H	X	X	X	X	X	X	X	X	X	X	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
8000-8FFF	H	L	L	L	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
9000-9FFF	H	L	L	H	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
A000-A3FF	H	L	H	L	L	L	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	L	H	H	H	L	H	H	H	H	H
A400-A7FF	H	L	H	L	L	H	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	L	H	H	L	H	H	H	H	H	H
A800-ABFF	H	L	H	L	H	L	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	L	H	L	H	H	H	H	H	H	H
AC00-AFFF	H	L	H	L	H	H	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	L	L	H	H	H	H	H	H	H	H
B000-BFFF	H	L	H	H	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
C000-CFFF	H	H	L	L	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
D000-DFFF	H	H	L	H	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
E000-EFFF	H	H	H	L	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
F000-FFFF	H	H	H	H	X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H

H = High L = Low X = Irrelevent

Table 7-2. Z27 SN74LS138 Decode Logic

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2*	C	B	A	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
A15		A14	A13	A12	$\overline{CSF}$	$\overline{CSE}$	$\overline{CSD}$	$\overline{CSC}$	$\overline{CSB}$	$\overline{CSA}$	$\overline{CS9}$	$\overline{CS8}$
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	H	H	H	H	H	H	H	L
H	L	L	L	H	H	H	H	H	H	H	L	H
H	L	L	H	L	H	H	H	H	H	L	H	H
H	L	L	H	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	L	H	H	H	H
H	L	H	L	H	H	H	L	H	H	H	H	H
H	L	H	H	L	H	L	H	H	H	H	H	H
H	L	H	H	H	L	H	H	H	H	H	H	H

H = High Level L = Low Level X = Irrelevant

*G2 = G2A + G2B = always L (GND)

Table 7-3. Z19 SN74LS155 Decode Logic - Side 1 ($\overline{CSA0}$-$\overline{CSAC}$)

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	1G	1C*	1Y3	1Y2	1Y1	1Y0
A11	A10	$\overline{CSA}$		$\overline{CSAC}$	$\overline{CSA8}$	$\overline{CSA4}$	$\overline{CSA0}$
X	X	H	X	H	H	H	H
L	L	L	H	H	H	H	L
L	H	L	H	H	H	L	H
H	L	L	H	H	L	H	H
H	H	L	H	L	H	H	H
X	X	X	L	H	H	H	H

H = High Level L = Low Level X = Irrelevant

* = Wired High

Table 7-4. Z19 SN74LS155 Decode Logic - Side 2 ($\overline{CS00}$-$\overline{CS0C}$)

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	2G*	2C	2Y0	2Y1	2Y2	2Y3
A11	A10		02	$\overline{CS00}$	$\overline{CS04}$	$\overline{CS08}$	$\overline{CS06}$
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

H = High Level L = Low Level X = Irrelevant

*2G = $\overline{A12}$. $\overline{A13}$. $\overline{A14}$. $\overline{A15}$

7.2.4 RAM

The R2114 Static RAM is organized 1024 words by 4-bits. One pair of R2114s are used to provide 1K 8-bit bytes. The I/O lines of one R2114 are connected to data lines D0-D3 to provide the LSD of the bytes (bits 0-3). The I/O lines of the other R2114 are connected to D4-D7 to provide the MSD of the byte (bits 4-7). Figure 7-5 shows the connection to the AIM 65 address, data and control bus lines.

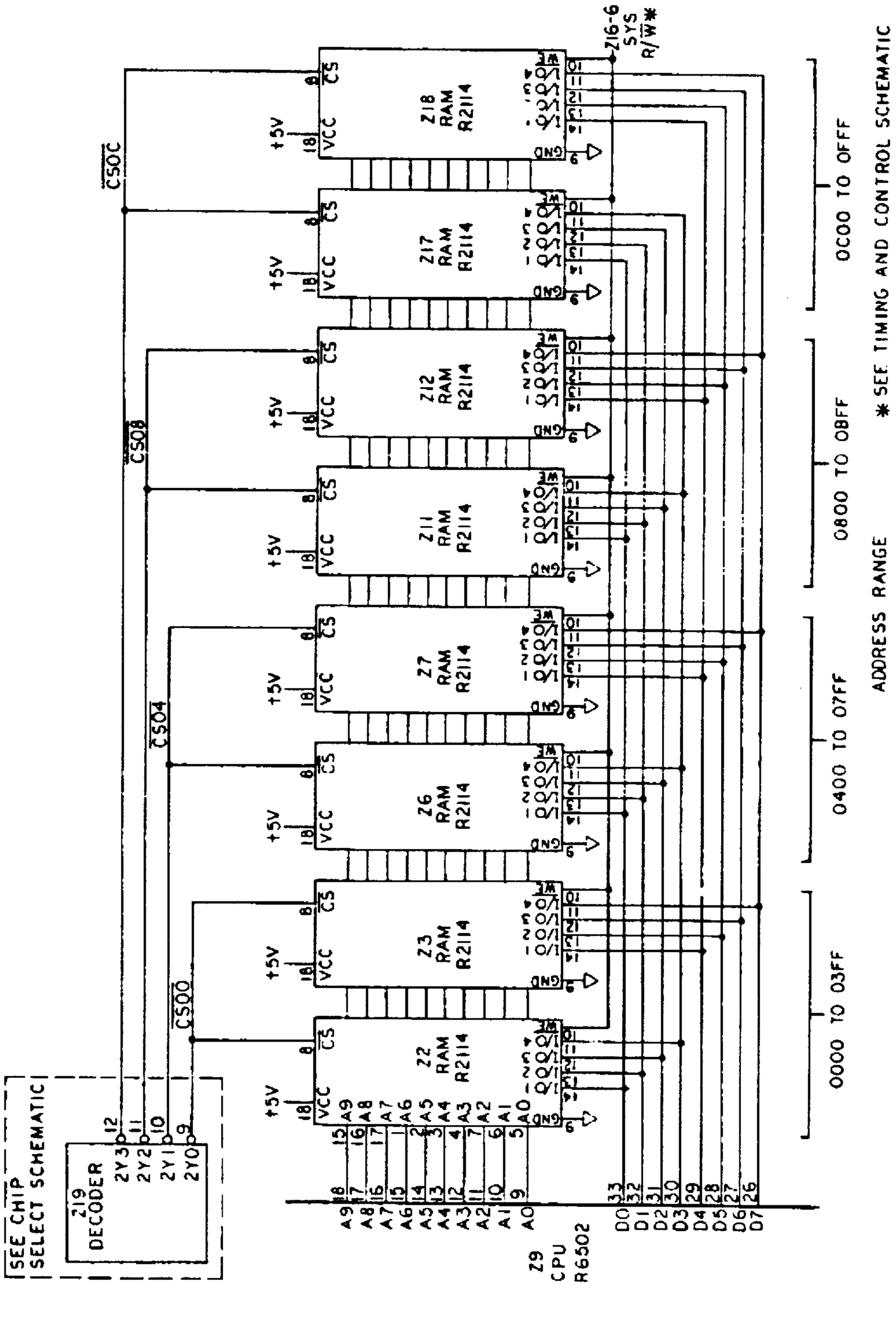


Figure 7-5. RAM Interface Schematic

Each RAM device Chip Select ($\overline{CS}$) line is connected to one of the RAM Chip Select lines ($\overline{CS00}$-$\overline{CS0C}$). When CS is low, the data in the R2114 will be placed on the device I/O lines (I/O1-I/04), per address lines A0-A9.

The R2114 Write Enable ($\overline{WE}$) line is connected to SYS R/$\overline{W}$. When $\overline{WE}$ (and/or $\overline{CS}$) is high, the R2114 data input buffers are inhibited to prevent data from being written into the internal memory. Data within the RAM is changed only when both $\overline{CS}$ and $\overline{WE}$ are both low.

RAM must be installed in Z2 and Z3 sockets to provide Page 0 and Page 1 addresses for the AIM 65 to operate. RAM may be installed optionally in the other sockets for on-board RAM expansion. The R2114 RAM devices must be added in pairs Z6 and Z7, Z11 and Z12, or Z17 and Z18 to provide complete bytes of data.

7.2.5 ROM

Five PROM/ROM sockets are provided on the AIM 65 Master Module. Each socket can accept a 4K R2332 ROM or compatible PROM (see Figure 7-6)

The R2332 is a 32,768-bit static ROM, organized 4,096 words X 8 bits. The two chip select lines, S1/$\overline{S1}$/NC and S2/$\overline{S2}$/NC, are mask programmed to the desired logic level. In AIM 65 the S1/$\overline{S1}$/NC line is masked to the $\overline{S1}$ state, i.e., active low, while the S2/$\overline{S2}$/NC line is masked to the S2 level, i.e.,active high. S2 is wired to +5V to be continuously high. $\overline{S1}$ is wired to one of the AIM 65 chip select lines $\overline{CSA}$-$\overline{CSF}$.

When $\overline{S1}$ is low, the data addressed by lines A0 to A11 is placed on data lines D0-D7.

The AIM 65 Monitor is stored in R2332 devices P/N R3222 (Z22) and in P/N R3223 (Z23). The optional AIM 65 Assembler is masked into R2332 P/N R3224 for installation in Z24. The optional 8K AIM 65 BASIC Interpreter is available in R2332, P/N R3225, and P/N R3226 for installation in Z25 and Z26, respectively.

Pin compatible PROMs may be directly installed in Z24 - Z26 to operate in conjunction with the AIM 65 Monitor. The AIM 65 Monitor ROM may also be replaced, if desired, with a user provided Monitor program. The only restriction is that the interrupt vectors are located at $FFFA-$FFFF. See Section 7.8 for a discussion of the interrupt vectoring.

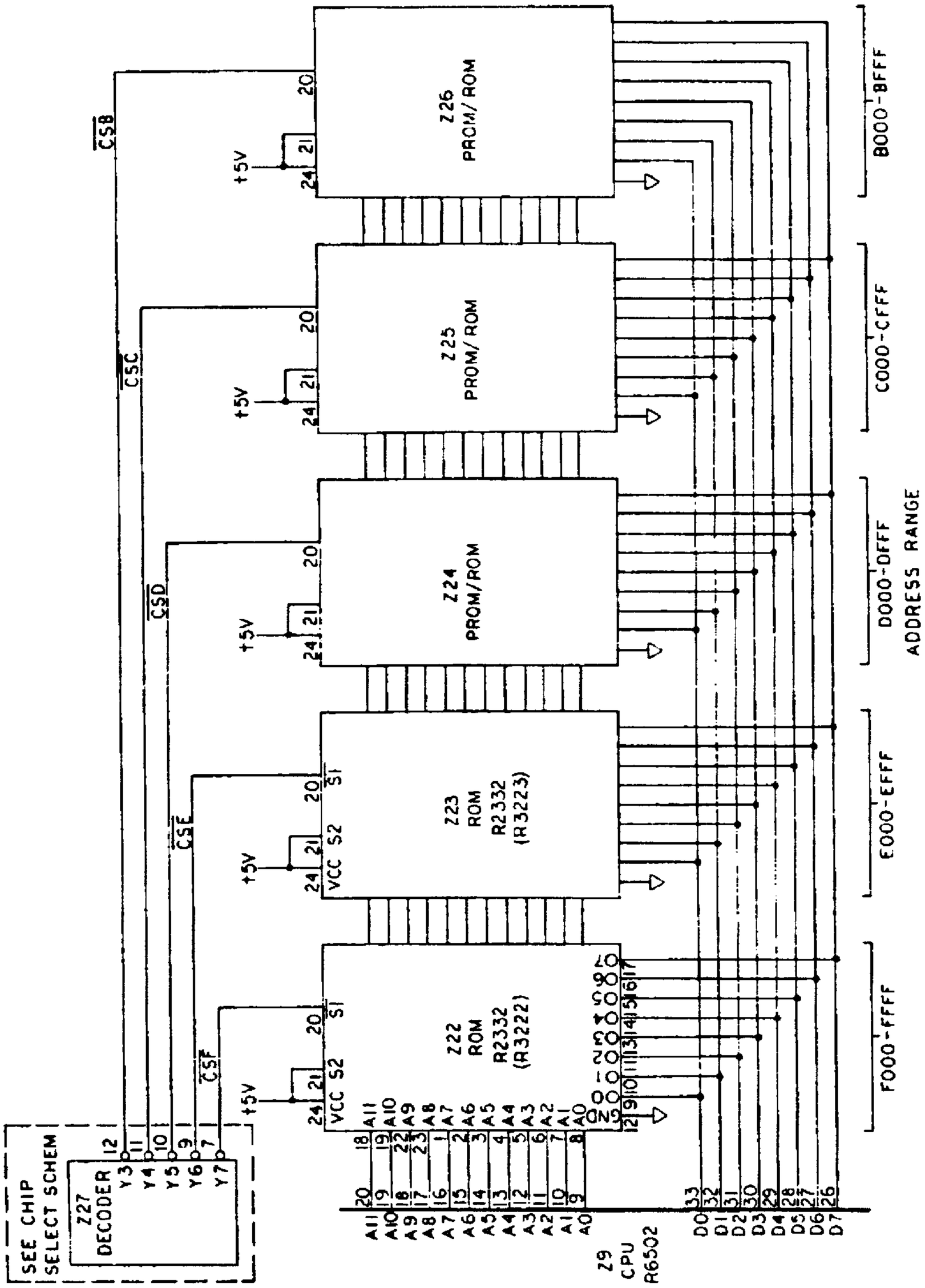


Figure 7-6. PROM/ROM Interface Schematic

7.2.6 Printer Interface

CAUTION

This section is presented for information only. Since improper timing of the print commands may damage the printer thermal head, it is not recommended that user prepared printer interface functions be attempted. The monitor output subroutines described in Table 7-13 may, however, be safely used.

The printer prints on heat sensitive roll paper by means of ten thermal elements, each of which can print two 5 x 7 matrix dot characters. The 10 thermal elements are mounted in fixed positions on a moveable thermal head. During a print cycle, the thermal head is driven back and forth horizontally allowing a row of dots to be printed during movement in each direction. The individual thermal elements are turned on for discrete intervals during the thermal head movement to form partial characters. After a row of dots has been printed, the motor driven platen advances the paper vertically by one dot row. A full line of formed characters is complete after seven dot rows are printed. The printer column layout and dot progression are illustrated in Figure 7-7. The printed characters are formed by dot patterns stored in the AIM 65 Monitor. The print cycle set-up, sequencing and timing is also controlled by the AIM 65 Monitor.

The hardware interface with the printer is provided by a portion of the AIM 65 Monitor R6522 VIA (Z32) and discrete circuitry (see Figure 7-8).

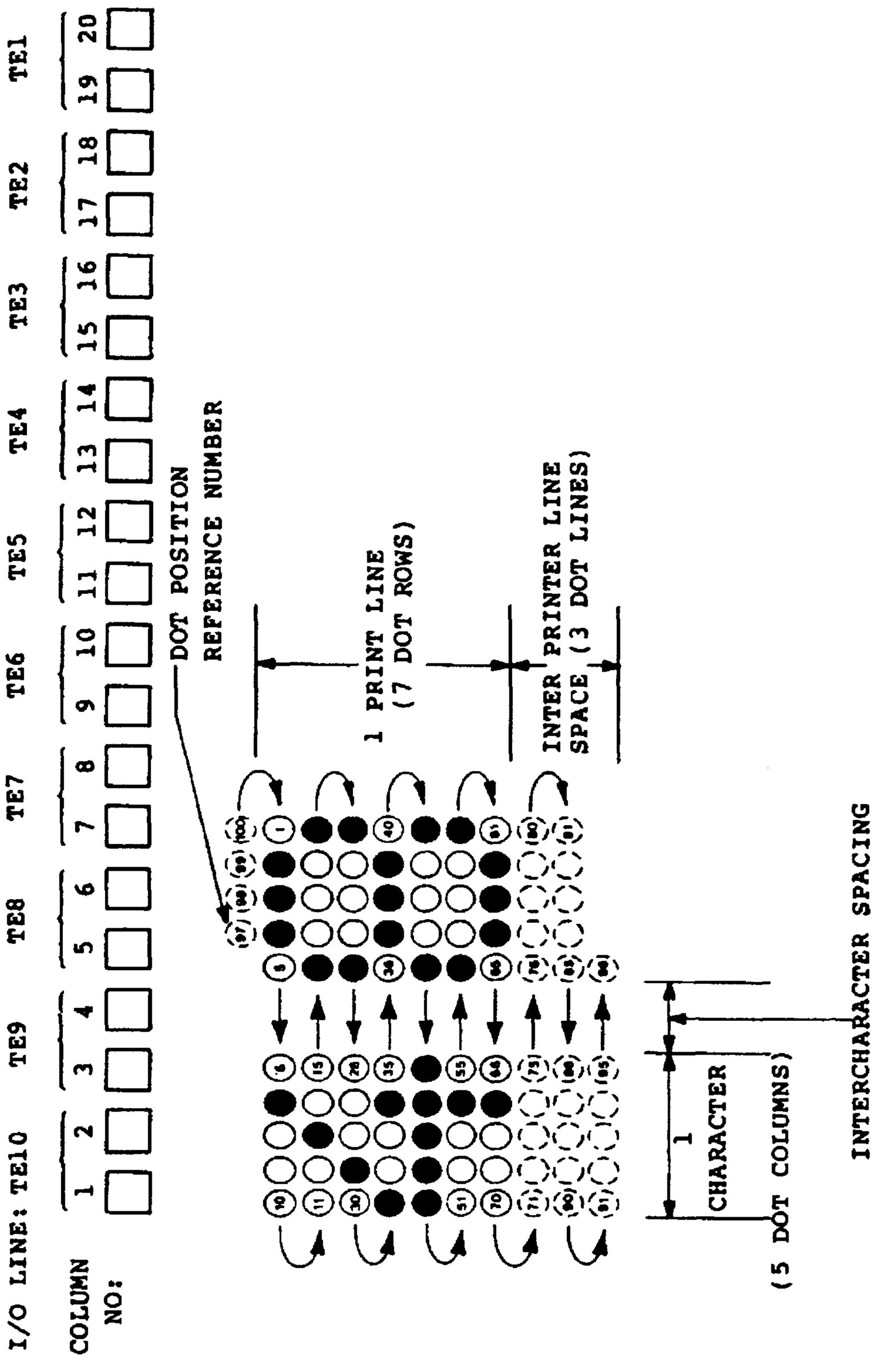


Figure 7-7. Printer Column Layout and Dot Progression

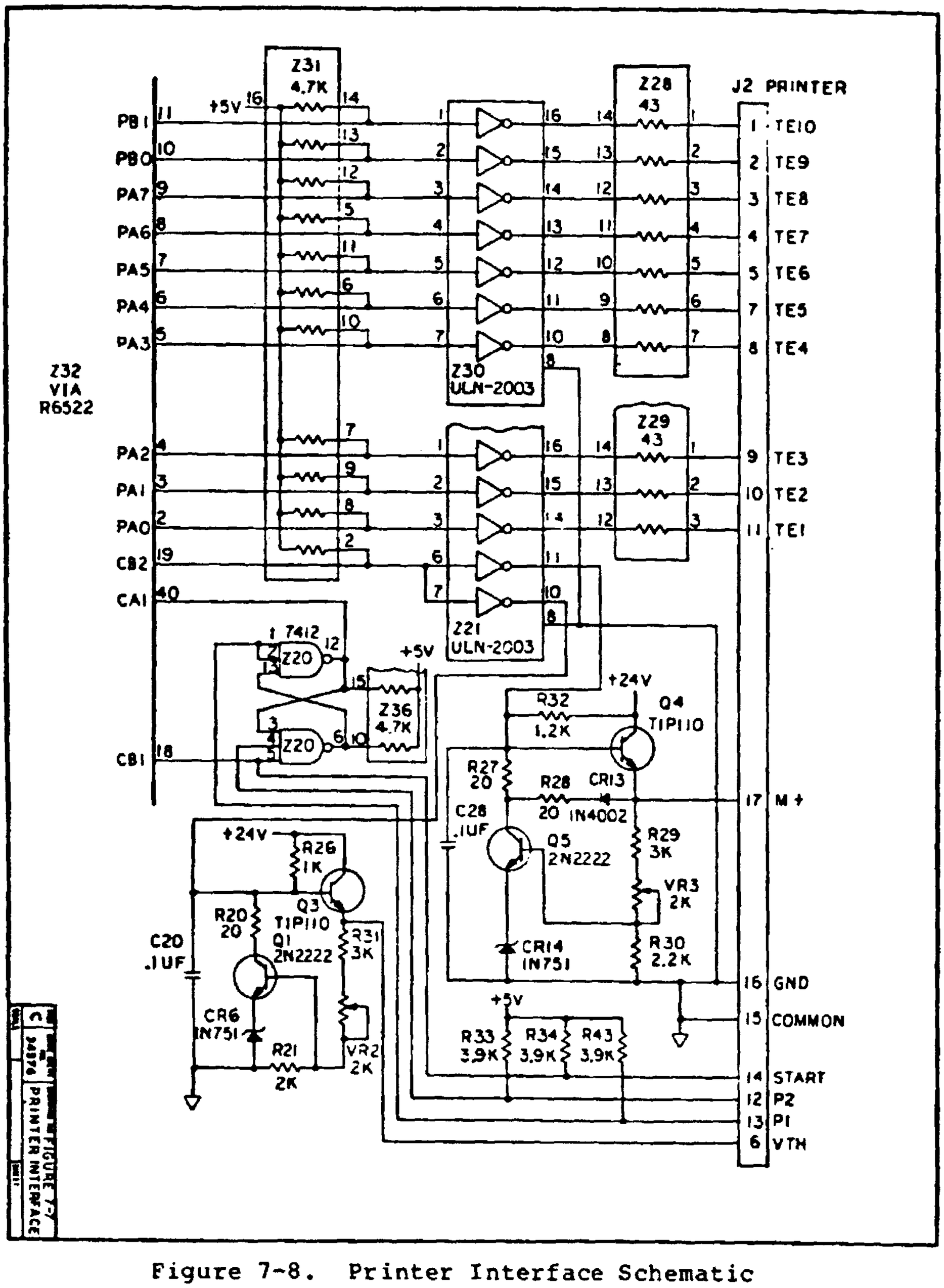


Figure 7-8. Printer Interface Schematic

CB2 operates as a discrete output to drive the motor and thermal head control circuits. A logic 0 to CB2 turns on the control circuits, logic 1 turns them off.

Motor control is provided by the Q4 power driver and Q5 feedback regulator. The output voltage on M+ varies from about 14 to 16 Vdc as adjusted by VR3, which controls the motor speed. The output voltage is zero when CB2 is logic 1. This circuit also dissipates back EMF energy generated by the motor when the motor is turned off, which provides required motor dynamic braking.

The Q3 power driver and Q1 feedback regulator and associated components provide the printer thermal head element voltage. When CB2 is logic 0, the output voltage on VTH varies from about 18 to 20 Vdc as adjusted by VR2, which controls the intensity of the printer. The output voltage is zero when CB2 is logic 1.

When the thermal head is turned on, the printer will print depending on the state of the ten thermal element lines. Thermal element lines TE1-TE10 are controlled by PA0-PA7, PB0 and PB1, respectively. When the I/O line is low (logic 0), the thermal element is turned off and will not print.

The motor operation and thermal head position is monitored by the START, P1 and P2 output lines from the printer. The printer START signal generates a negative pulse when the motor is turned on and at the start of each printed dot row. CB1 is configured to detect a negative transition of the pulse leading edge. The START signal also resets the Z20 P1/P2 debounce flip-flop to the low state. The

CA1 input is initialized to detect a positive transition.

Printer strobe signals P1 and P2 are pulses that request the thermal elements to be turned on to print dots. The P1 pulse is set low to request odd dots to be printed first. When P1 goes low, Z20-12 goes high causing a positive transition on CA1. CA1 is then reconfigured to interrupt a negative transition. The P1 is reset high before the P2 signal goes low to request even dots to be printed. Z20-12 is reset to low when P2 goes low. The CA1 input detects the positive to negative transition. This process is repeated during the print cycle.

7.2.7 Display Interface

The AIM 65 display consists of five four-digit 16 segment alphanumeric displays. Each display (DS1 - DS5) contains internal memory, decoder and driver circuitry. The displays interface with the AIM 65 address, data and control bus lines through the R6520 PIA (U1) mounted on the display module (see Figure 7-9). Each display is controlled by seven data lines (D0 - D6), two address lines (A0 and A1), two control lines ($\overline{W}$ and $\overline{CW}$) and a chip select ($\overline{CE}$).

There are five separate chip select lines ($\overline{CE1}$ - $\overline{CE5}$), one to drive each display.

Table 7-5 shows the display decode logic. To load data, $\overline{CE}$ is held low to the desired display. The desired data code (see Table 7-6) is placed on D0-D6 and the selected digit address (0-3) is placed on A0 and A1. The cursor line ($\overline{CU}$) is held in the high state. The write ($\overline{W}$) line is driven low to store and display the data. After W is returned to high, the data will continue to be displayed until replaced with new data or the cursor is displayed. Data entry may be asynchronous and random.

A display hardware cursor function may be displayed instead of an ASCII character. The cursor function causes all 16 segments of a character to turn on. The cursor is not a character, however, and upon removal the last stored displayed character is restored. The cursor is displayed when $\overline{CU}$ is low while $\overline{W}$ is low and any of the D0-D3 lines are high. The cursor will be displayed in digits 0-3 for each D0-D3 line that is high, respectively (see Table 7-7).

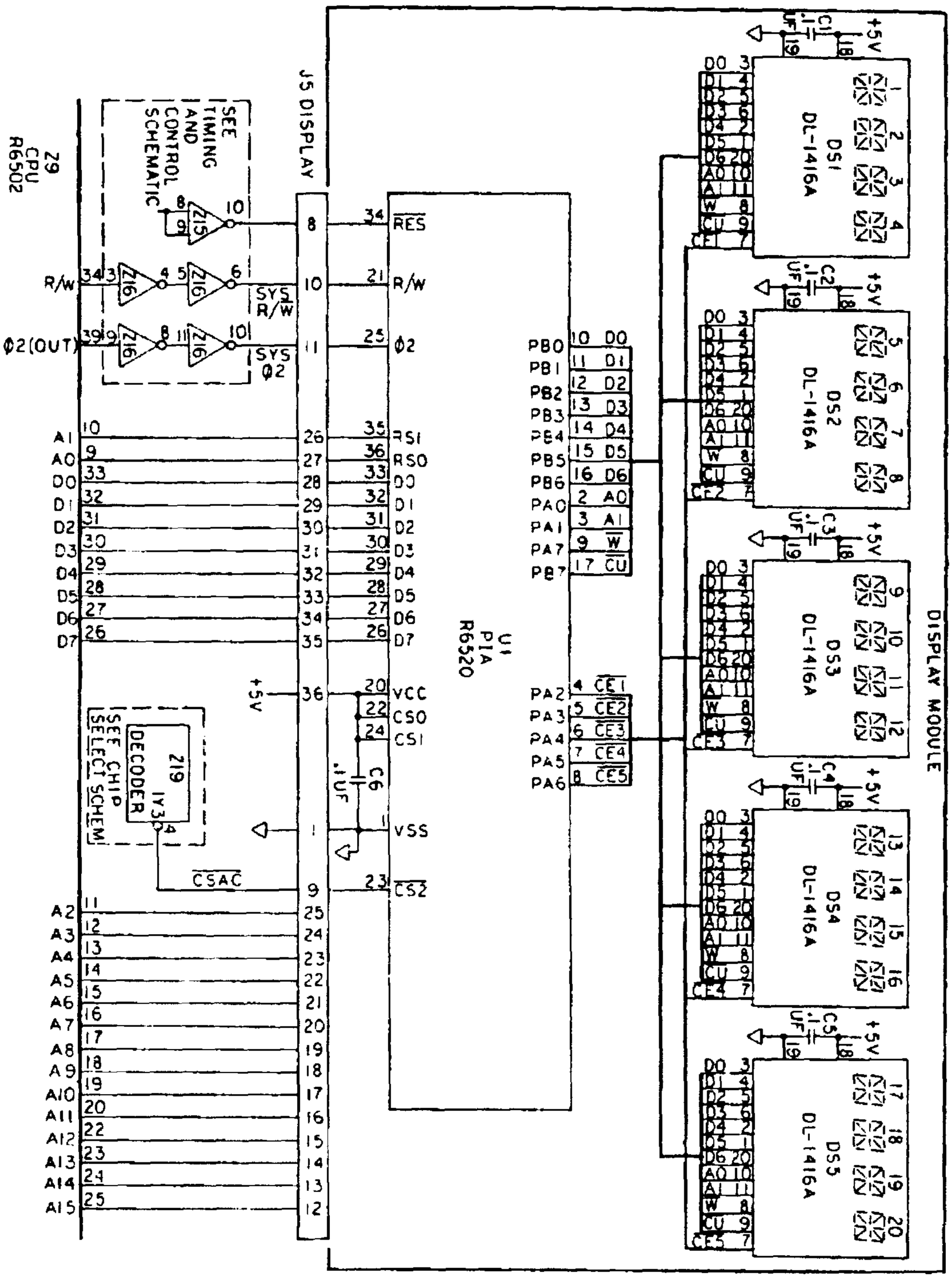


Figure 7-9. Display Interface Schematic

Table 7-5. Display Decode Logic

DISPLAY INPUTS																DISPLAY OUTPUTS																			
																DS1				DS2				DS3				DS4				DS5			
W	CE5	CE4	CE3	CE2	CE1	A1	A0	CU*	D6	D5	D4	D3	D2	D1	D0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
H	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	H	L	L	L	H								N	N	N	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	H	L	L	H	H								N	N	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	H	L	H	L	H								N	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	H	L	H	H	H								T	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	L	H	L	L	H								N	N	N	N	N	N	N	T	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	L	H	L	H	H								N	N	N	N	N	N	T	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	L	H	H	L	H								N	N	N	N	N	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	H	L	H	H	H	H								N	N	N	N	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
L	H	H	L	H	H	L	L	H	Refer to Character Set							N	N	N	N	N	N	N	N	N	N	N	T	N	N	N	N	N	N	N	N
L	H	H	L	H	H	L	H	H	in Table 7-6							N	N	N	N	N	N	N	N	N	N	T	N	N	N	N	N	N	N	N	N
L	H	H	L	H	H	H	L	H								N	N	N	N	N	N	N	N	N	T	N	N	N	N	N	N	N	N	N	N
L	H	H	L	H	H	H	H	H								N	N	N	N	N	N	N	N	T	N	N	N	N	N	N	N	N	N	N	N
L	H	L	H	H	H	L	L	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	T	N	N	N	N
L	H	L	H	H	H	L	H	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	T	N	N	N	N	N
L	H	L	H	H	H	H	L	H								N	N	N	N	N	N	N	N	N	N	N	N	N	T	N	N	N	N	N	N
L	H	L	H	H	H	H	H	H								N	N	N	N	N	N	N	N	N	N	N	N	T	N	N	N	N	N	N	N
L	L	H	H	H	H	L	L	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	T
L	L	H	H	H	H	L	H	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	T	N
L	L	H	H	H	H	H	L	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	T	N	N
L	L	H	H	H	H	H	H	H								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	T	N	N	N

*Note: If CU is low and W is low the cursor symbol (*) will be displayed in the digit rather than the character set symbol shown in Table 7-6.

T = Refer to Table 7-6, Display Character Set

N = No change from previous display

Table 7-6. Display Figure Set

CHARACTER SET

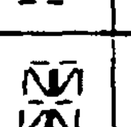
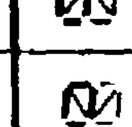
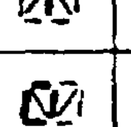
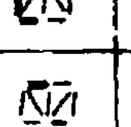
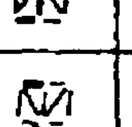
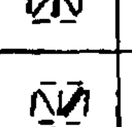
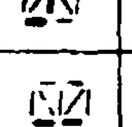
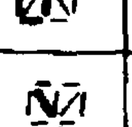
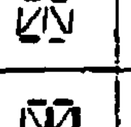
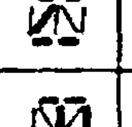
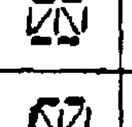
D6 D5 D4 D3 \ D0 D1 D2	L L L	H L L	L H L	H H L	L L H	H L H	L H H	H H H
L H L L								
L H L H								
L H H L								
L H H H								
H L L L								
H L L H								
H L H L								
H L H H								

Table 7-7. Display Data/Cursor Loading

LOADING DATA

$\overline{CE}$	$\overline{CU}$	$\overline{W}$	ADDRESS A_1	ADDRESS A_0	DATA INPUT D6	D5	D4	D3	D2	D1	D0	DIGIT 3	DIGIT 2	DIGIT 1	DIGIT 0
H	X	X	X	X	X	X	X	X	X	X	X	NO CHANGE	NO CHANGE	NO CHANGE	NO CHANGE
L	H	L	L	L	H	L	L	L	L	L	H	NO CHANGE	NO CHANGE	NO CHANGE	A
L	H	L	L	H	H	L	L	L	L	H	L	NO CHANGE	NO CHANGE	B	A
L	H	L	H	L	H	L	L	L	L	H	H	NO CHANGE	C	B	A
L	H	L	H	H	H	L	L	L	H	L	L	D	C	B	A
L	H	L	L	L	H	L	L	L	H	L	H	D	C	B	E
L	H	L	H	L	H	L	L	H	L	H	H	D	K	R	E
L	H	L	–	–	–	–	–	–	–	–	–	SEE CHARACTER SET			

X = DON'T CARE

LOADING CURSOR

$\overline{CE}$	$\overline{CU}$	$\overline{W}$	ADDRESS A_1	ADDRESS A_0	DATA INPUT D6	D5	D4	D3	D2	D1	D0	DIGIT 3	DIGIT 2	DIGIT 1	DIGIT 0
H	X	X	X	X	X	X	X	X	X	X	X	D	K	B	E
L	L	L	X	X	X	X	X	L	L	L	H	D	K	B	D
L	L	L	X	X	X	X	X	L	L	L	L	D	K	B	E
L	L	L	X	X	X	X	X	L	L	H	L	D	K	R	E
L	L	L	X	X	X	X	X	L	H	L	L	D	R	B	E
L	L	L	X	X	X	X	X	H	L	L	L	R	K	B	E
L	L	L	X	X	X	X	X	H	H	H	H	R	R	R	R
L	L	L	X	X	X	X	X	L	L	L	L	D	K	B	E

X = DON'T CARE

7.2.8 Keyboard Interface

The interface to the AIM 65 keyboard is through the R6532 RIOT (see Figure 7-10). Z33 R6532 peripheral I/O lines PA0 through PA7 are assigned to keyboard input lines K11 through K18, respectively. R6532 lines PB0 through PB7 are wired to the keyboard output lines KO1 through KO8, respectively.

When scanning for key depression, a logic 0 is placed in the R6532 Output Register A (ORA) in one bit position at a time corresponding to one KI line. The logic 0 provides a low output to the KI line key switches. Each key depressed of the switches connected to selected KI line will present a closed circuit output from KO1-KO8 causing a logic 0 to be present in the respective bit position of R6532 Output Register B (ORB). Each unpressed key presents an open input circuit to PB0-PB7 causing a logic 1 to be present in the respective R6532 ORB bit position.

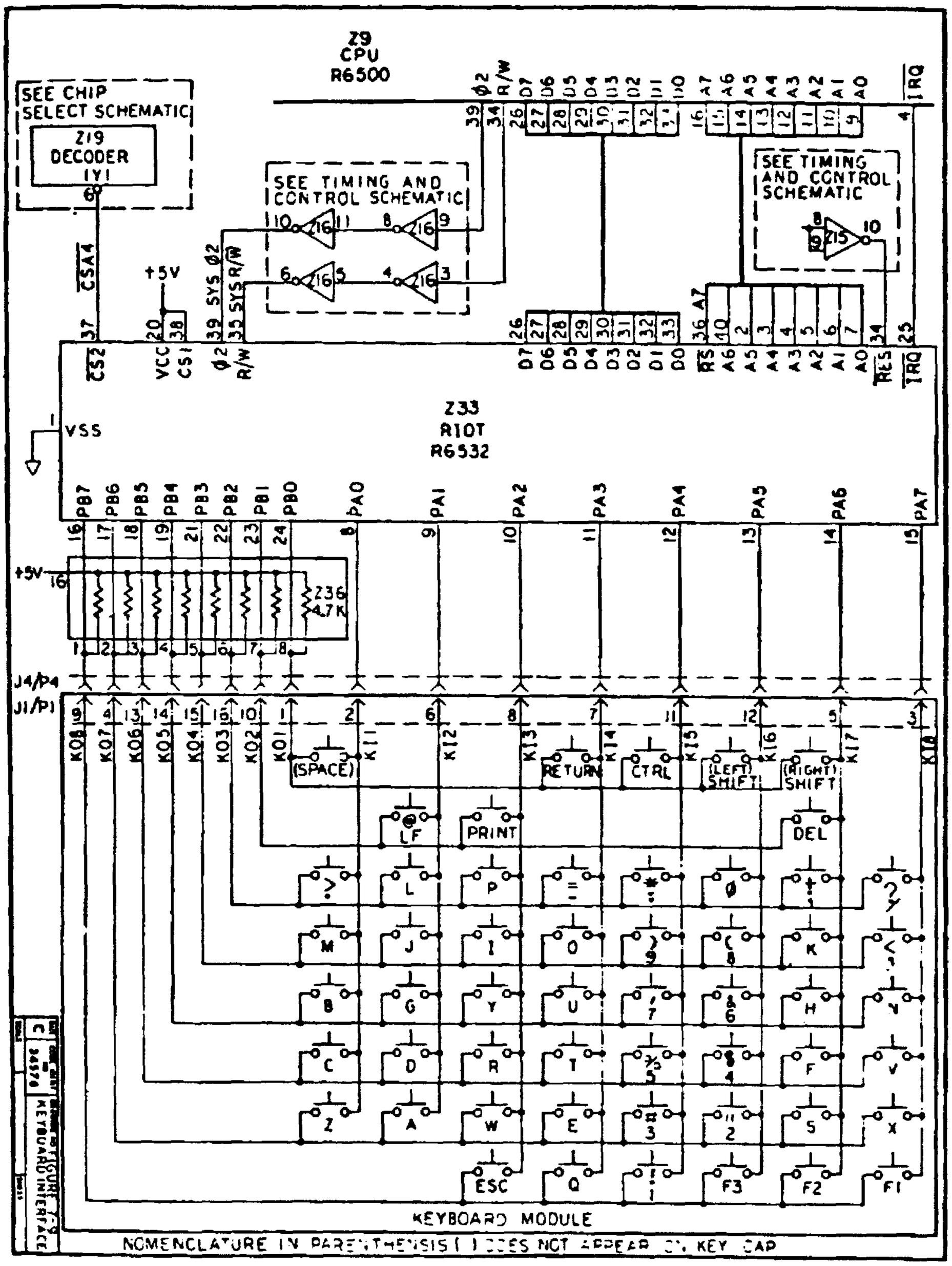


Figure 7-10. Keyboard Interface Schematic

7.2.9 User R6522 Interface

The User R6522 VIA (Z1) interface with the J1 Application Connector is shown in Figure 7-11. The total resources of the R6522 are available for user defined applications. Refer to the R6500 Hardware Manual for a full discussion of R6522 capabilities and application instructions.

The R6522 internal registers are accessible when CS1 is high and $\overline{CS2}$ is low. Since CS1 is wired high to +5V and $\overline{CS2}$ is connected to the chip-select output $\overline{CSA0}$, Z1 is enabled whenever the address is between A000 to A3FF (see Table 7-1). Also, because the Z1 RS0 to RS3 lines are connected to address lines A0 to A3, Z1 will respond to all addresses in the A000-A3FF range. The primary memory map for Z1 is A000-A00F (see Tables 7-9 and 7-10) so these addresses should be used and the addresses between A010 and A3FF avoided.

The Z1 $\overline{IRQ}$ output lines is connected to the Z9 R6502 CPU $\overline{IRQ}$ input for user definition and application. See Section 7.8 for a description of the AIM 65 Monitor IRQ interrupt linkage and handling.

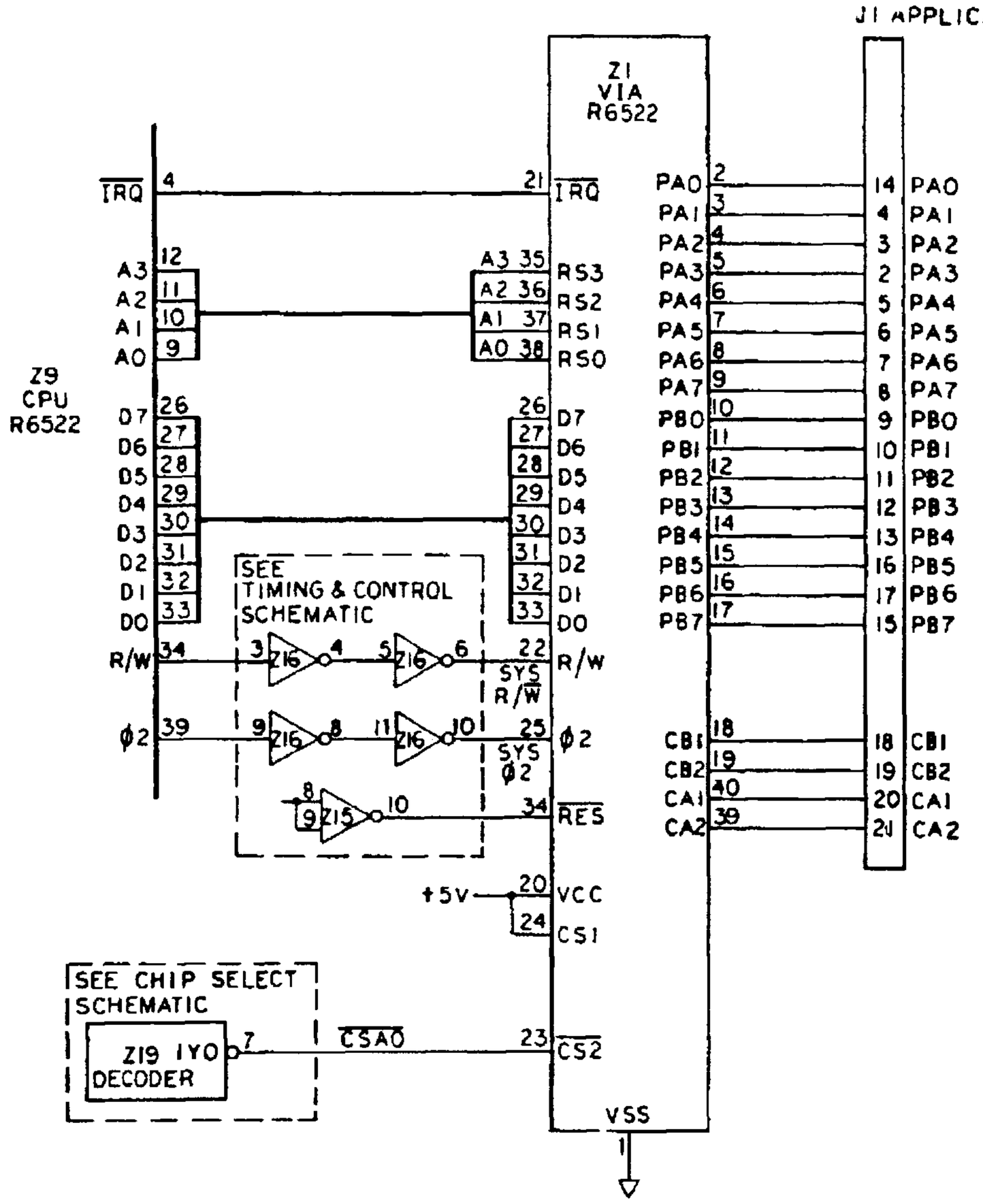


Figure 7-11. User R6522 Interface Schematic

7.2.10 Audio Cassette Recorder Interface

The AIM 65 audio cassette recorder interface provides audio data routing control and waveform shaping as well as recorder remote control circuitry. (See Figure 7-12.).

CA2 operates as a discrete output to control the audio data direction. When CA2 output is set high (logic 1), the audio input data from J1-L (AUDIO IN) is enabled through gate Z5-11 to PB-7, which is configured as an input. Z8 and associated circuitry provides AC coupling and input data compensation and shaping. Note that the audio input data output from Z5-11 is routed back through Z5-8 to J1-P (AUDIO OUT HI) and J1-M (AUDIO OUT LOW).

To output audio data from AIM 65, CA2 output is reset low by logic 0 to gate Z5-11 to inhibit any audio input data or noise from mixing with the audio output data. PB-7 is configured to operate as an output. Audio output data from PB-7 is routed to J1-P (AUDIO OUT HI) and J1-M (AUDIO OUT LO). AC coupling and waveform shaping is performed.

The audio data circuitry supports both AIM 65 and KIM-1 recording formats. The AIM 65 format is described in Appendix F, the KIM-1 format is described in Appendix G.

PB4 and PB5 are outputs that drive the audio cassette recorder remote control circuits. A logic 0 (PB4/PB5 low) turns Q6/Q7 off causing an open circuit in the recorder remote line thereby stopping motor. A logic 1 (PB4/PB5 high) turns Q6/Q7 on causing a closed circuit in the recorder remote line thereby allowing the motor to operate.

Four types of recorder motor remote control circuits are supported. Refer to Section 9.1.1 for a description of the recorder remote control wiring and recorder installation.

In the recorder remote control type PRC and PRS circuits, PB4 opens and closes the recorder No. 1 remote circuit from J1-W (motor low) to GND through Z21-12, while PB5 control recorder No. 2 power circuit from J1-V to GND through Z21-13. The AIM 65 GND must be connected to the record GND through J1-1.

In the type PVC and PVS circuits, PB4 opens and closes the recorder No. 1 remote circuit from J1-F (motor positive voltage supply) to J1-E (motor high) through Q6. PB5 controls the recorder No. 2 remote circuit from J1-J to J1-H through Q7.

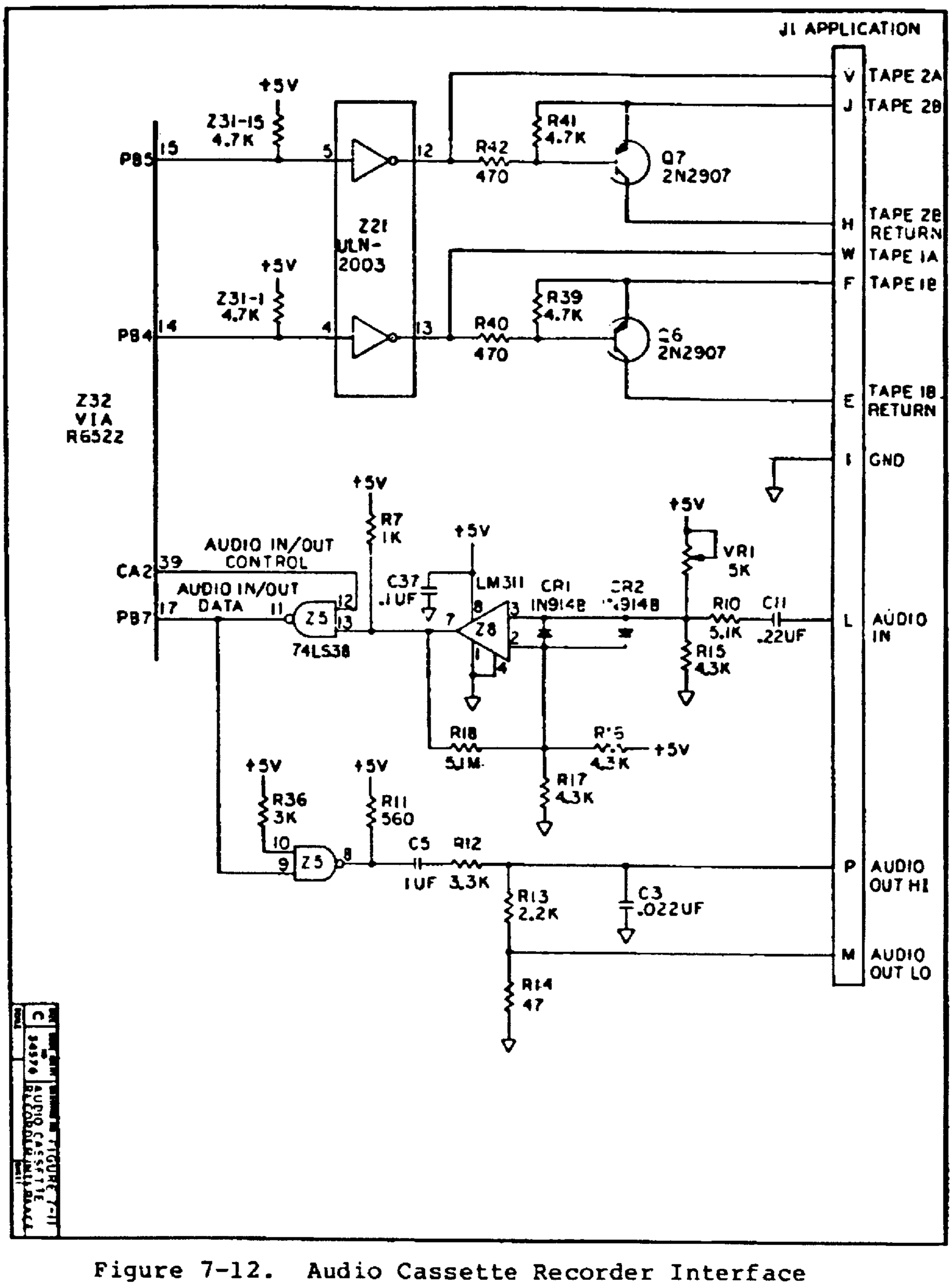


Figure 7-12. Audio Cassette Recorder Interface Schematic

7.2.11 TTY and Serial Interface

AIM 65 provides a 20 MA current loop, full duplex interface with a teleprinter or teletype (TTY). A serial input capability is also included (see Figure 7-13). Refer to Section 9.2 for the TTY installation instructions.

The current source for the TTY printer is provided at J1-S (TTY PTR RTN (+)) from the AIM 65 +5V power supply as limited by R1. The output to the TTY printer originates from PB2, which is configured as an output. The output signal is inverted by permanently enabled gate Z5-3. CR7 clamps the output to GND so only a positive pulse stream is presented to the TTY printer on J1-U (TTY PTR).

The current source for the TTY keyboard is provided at J1-R, TTY KYBD RTN (+), from the AIM 65 +24V power supply as limited by R25. The TTY keyboard input received on J1-T (TTY KYBD) is shaped by Q2 and associated components. The signal is then inverted through permanently enabled gate Z5-6 to the R6522 I/O pin PB6, which is configured as an input.

The ability to input a serial bit stream on J1-Y may be used instead of the TTY keyboard input. The bit transmission rate may be as high as 9600 baud.

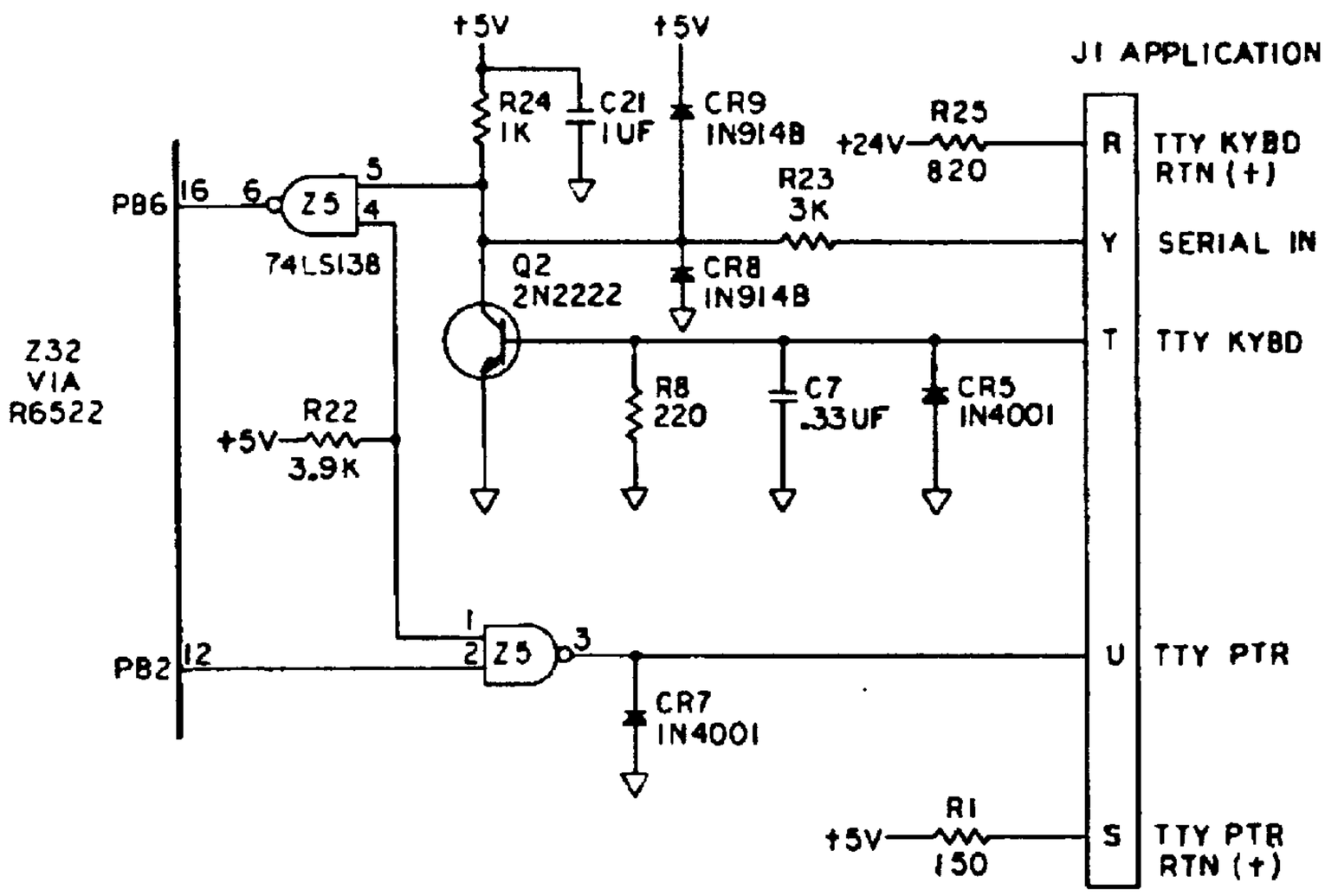


Figure 7-13. TTY and Serial Interface Schematic

7.3 AIM 65 SOFTWARE

The AIM 65 software consists of:

- The AIM 65 8K Monitor (including the Text Editor)
- The Optional AIM 65 4K Assembler
- The Optional AIM 65 8K BASIC

7.3.1 AIM 65 Monitor

The AIM 65 Monitor software provides the overall control of the AIM 65 peripheral devices (except the Z1 user R6522), the control and sequencing of AIM 65 commands and linkage to user functions. Figure 7-13 is a top level flowchart of the Monitor interrupt and command mode processing.

AIM 65 MONITOR ENTRY DUE TO RESET INTERRUPT

The Monitor is initially entered via the RST Interrupt Vector either due to AIM 65 power turn-on or pressing the RESET button. The R6502 decimal mode is cleared and the IRQ interrupt is disabled. The start pointer is initialized to $01FF as is the user saved stack pointer.

The AIM 65 peripheral devices used by the Monitor, i.e., R6522 (Z32), R6532 (Z33), and the R6520 (U1) are initialized to set up the data direction registers, load constant data and establish counter/interrupt modes (if applicable).

The variables required for AIM 65 Monitor operation are initialized. If address $A402 is not $7B or if $A403 is not $EO (the AIM 65 NMI interrupt vector) "cold" reset

variables are initialized. During a "cold" reset, the parameters listed in Table 7-8 are initialized to the stated values. Otherwise, a "warm" reset is performed which initializes only the variables required for AIM 65 Monitor initialization.

If the KB/TTY switch is in the TTY position, the Monitor waits until RUBOUT is typed on the TTY keyboard. The Monitor then measures the TTY to AIM 65 bit transmission rate and stores the value in address $A417 (CNTH30) and $A418 (CNTL30).

The ROCKWELL AIM 65 message is displayed/printed to indicate that AIM 65 reset has been performed. The Monitor command mode is then entered.

AIM 65 MONITOR ENTRY DUE TO IRQ INTERRUPT

The Monitor IRQ Interrupt Processing may be entered due to a BRK instruction being executed. If a BRK instruction was the cause, AIM 65 displays the register contents, disassembles and displays the next instruction and enters the Monitor and Software Command Mode. See Section 7.8 for user IRQ interrupt linkage and the R6500 Hardware Manuals for additional IRQ processing considerations.

AIM 65 MONITOR ENTRY DUE TO NMI INTERRUPT

The Monitor NMI Interrupt Routine may be entered due to execution of an instruction outside the $A000-$FFFF address range while in STEP Mode (RUN/STEP switch in the STEP position). If this is the cause and either an enabled

breakpoint address is encountered (see Monitor B command) or the instruction execution termination criteria is satisfied (see Monitor G command), the register contents are displayed, the next instruction disassembled, and the Monitor Command Mode is entered. See Section 7.8 for User NMI interrupt linkage information and the R6500 Hardware and Software Manuals for additional NMI interrupt processing considerations.

AIM 65 MONITOR COMMAND MODE

The Monitor Command Mode initially displays the Monitor prompt "<" to indicate that a Monitor command may be typed. Upon receipt of a typed command, the typed character is displayed along with the command closing ">" character. If the command is not valid, "?" is displayed and the command mode re-entered. If the command is valid, the Monitor calls the command subroutine. The command sub-routine executes an RTS instruction to return to the Monitor upon the completion of command processing. The Monitor then re-enters the command mode.

7.3.2 AIM 65 Memory Map

Table 7-9 shows the overall AIM 65 memory map. The input/ output portion is further defined in Table 7-10. A complete detailed memory map is listed in Table 7-11.

7.4 USER DEFINED FUNCTIONS LINKAGE

Three user-defined functions may be entered by typing the F1, F2, and F3 keys. Typing one of the user function

keys causes the Monitor to jump to the AIM 65 RAM locations assigned to the first instruction in the user function. That address in turn should be loaded with a three byte JMP instruction to the rest of the user function instructions for that function located in user RAM.

CAUTION

Typing F1, F2, or F3 prior to loading a JMP instruction in the appropriate address may cause AIM 65 to hangup or to operate incorrectly. Press RESET to recover.

LABEL	ADDRESS	DESCRIPTION
KEYF1	$010C	JMP Instruction to Function 1
KEYF2	$010F	JMP Instruction to Function 2
KEYF3	$0112	JMP Instruction to Function 3

7.5 USER DEFINED INPUT/OUTPUT FUNCTION

The input (IN=) and output (OUT=) prompts allow a U to be typed to specify a user-defined I/O function. The Monitor indirectly calls the function pointed to by the user I/O handler vectors:

LABEL	ADDRESS	BYTES	DESCRIPTION
UIN	$0108	2	Vector to User Input Handler
UOUT	$010A	2	Vector to User Output Handler

USER INPUT HANDLER

The user input handler is usually called by the Monitor or a user program from the WHEREI and INALL subroutines. WHEREI is first called and asks for the input device code, i.e., IN= . In response to the typed U, WHEREI clears the carry bit in the processor status register to indicate initial entry into the user input function.

The input function should test the carry bit and conditionally branch on carry clear to input device initialization. Upon completion of input device initialization, the input function should return to the calling program. The calling program should make subsequent calls to the user input function from the INALL subroutine which first sets the carry bit. The user input function should perform the normal input processing on carry bit set.

Example:

Calling Routine

```
JSR WHEREI
JSR INALL
```

User Input Function

```
UIN .WOR INTST  (Vector to input handler subroutine)
INTST            BCC IPINIT
                 :
Perform Input Processing
:
RTS
```

```
IPINIT          Perform Input Initialization
                :
                RTS
```

USER OUTPUT HANDLER

The user output handler is usually called by the Monitor or a user program from the WHEREO and OUTALL subroutines. WHEREO is first called and asks for the output device code, i.e., OUT= . In response to the typed U, WHEREO clears the carry bit in the processor status register to indicate initial entry into the user output function. The output function should test the carry bit and conditionally branch on carry clear to output device initialization, if required. Upon completion of output device initialization, the output function should return to the calling program. The calling program should subsequently call the user program using the OUTALL subroutine which first sets the carry bit. The user output function test should perform normal output processing on carry bit set.

Example:

Calling Routine

```
JSR WHEREO
JSR OUTALL
:
.
```

User Output Function

```
WOUT .WOR OUTTST   (Vector to output handler subroutine)
```

```
OUTTST BCC OTINIT
          :
          Perform Output Processing
          .
          RTS
OTINIT Perform Output Initialization
          :
          RTS
```

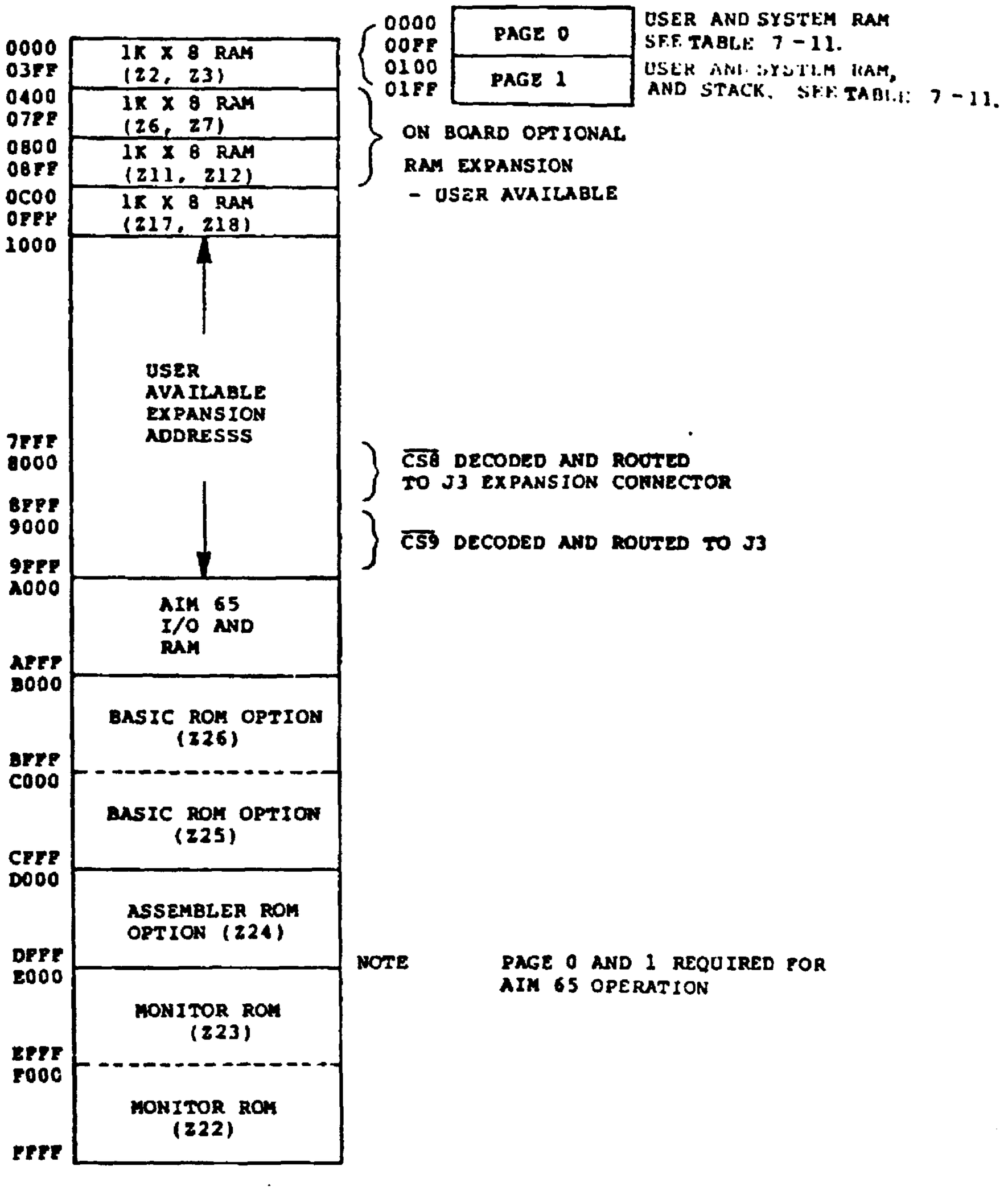
Table 7-8. Parameters Initialized by "Cold" Reset

ADDRESS	LABEL	BYTES	VALUE	DESCRIPTION
A402	NMIV2	2	$7BEO	Vector to $E07B (NMIV3) Monitor NMI Interrupt Processing
A404	IRQV2	2	$54El	Vector to $E154 (IRQV3) - Monitor IRQ Interrupt Processing
A406	DILINK	2	$05EF	Vector to $EF05 (OUTDIS) - Echo Input to AIM 65 Display
A408	TSPEED	1	$C7	$C7 = AIM 65 Format -
A409	GAP	1	$08	$08 = 32 SYN Characters

Table 7-8. Parameters Initialized by "Cold" Reset (Cont.)

ADDRESS	LABEL	BYTES	VALUE	DESCRIPTION
A40A	NPUL	1	$02	
A40B	TIMG	3	$CA038D	
A40E	REGF	1	$00	Turn Register Trace Off
A40F	DISFLG	1	$00	Turn Instruction Trace Off
A410	BKFLG	1	$00	Breakpoint Enable Off
A411	PRIFLG	1	$80	Turn Printer On

Table 7-9. AIM 65 System Memory Map



NOTE PAGE 0 AND 1 REQUIRED FOR AIM 65 OPERATION

Table 7-10. AIM 65 I/O Memory Map (A000-AFFF)

A000	User R6522
A00F	(Z1)
A010	
	Not Available
A3FF	
A400	
	Monitor RAM
A47F	
A480	AIM 65 R6532
	(Z33)
A497	Keyboard I/O
A498	Not Available
A7FF	
A800	AIM 65 R6522
	(Z32)
	Printer, TTY &
A80F	Tape I/O
A810	
	Not Available
ABFF	
AC00	AIM 65 R6520
	(U1)
AC43	Display I/O
AC44	
AFFF	Not Available

NOTE

(1) Unassigned addresses are not available to the user due to address conflict with AIM 65 I/O assignments.

Table 7-11. AIM 65 Detail Memory Map

ADDRESS	AREA	LABEL	BYTES	FUNCTION
0000 00AC	User RAM ↓			Note: Addresses 0004 through 00DE are used by the Assembler.
00AD	Assembler ↓	TABUF2	50	Assembler Tape Output Buffer
00DF	↑	NOWLN	2	Current Line
00E1		BOTLN	2	Last Active, so far
00E3	Editor	TEXT	2	Text Buffer Starting Address
00E5		END	2	Text Buffer Ending Address Limit
00E7		SAVE	2	Used by Replace
00E9		OLDLEN	1	Orig Length
00EA		LENGTH	1	New Length
00EB	↓	STRING	20	Find String
0100	Breakpoints	BKS	8	BRK Locations
0106	Dump	S2	2	Vertical Count
0108	User I/O	UIN	2	User Input Handler Address
010A	Handler	UOUT	2	User Output Handler Address
010C	User	KEYF1	3	JMP Instruction to F1 Function
010F	Functions	KEYF2	3	JMP Instruction to F2 Function
0112		KEYF3	3	JMP Instruction to F3 Function
0115	Tape I/O	BLK	1	Input Block No.
0116	↓	TABUFF	80	Tape Buffer (I/O)
0168		BLKO	2	Output Block No.

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
0116	Disassembler	FORMA	1	
0117	↓	LMNEM	1	
0118		RMNEM	1	
0119		SORN	1	
011A		NUM	2	
011C		SYM	6	
0122		SIZE	2	
0124		STSAVE	2	
0126	Mnemonic	TMASK1	1	
0127	Entry	TMASK2	1	
0126	Mnemonic	MOVAD	2	
0128	Entry	BEGAD	2	
012A	↓	ENDAD	2	
012C		TABEND	2	
012E		TYPE	1	
012F		HASHDA	1	
0130		CH	3	
0133		ADFLD	20	
0169 016F	Reserved for Monitor			
0170	Stack Pointer		144	Available to AIM 65 and User (Addresses 0170 through 0183 are used by the Assembler)
01FF	↓			

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A000	User R6522 VIA (Z1)	UDRB	1	Data Register B
		BIT	SIGNAL	CONN/PIN — AIM 65 SIGNAL NAME
		0	PB0	J1-9 — User Defined
		1	PB1	J1-10
		2	PB2	J1-11
		3	PB3	J1-12
		4	PB4	J1-13
		5	PB5	J1-16
		6	PB6	J1-17
		7	PB7	J1-15
A001		UDRAH	1	Data Register A (Controls Handshake)
		BIT	SIGNAL	CONN/PIN — AIM 65 SIGNAL NAME
		0	PA0	J1-14 — User Defined
		1	PA1	J1-4
		2	PA2	J1-3
		3	PA3	J1-2
		4	PA4	J1-5
		5	PA5	J1-6
		6	PA6	J1-7
		7	PA7	J1-8
A002		UDDRB	1	Data Direction Register B
A003		UDDRA	1	Data Direction Register A
A004		UT1L	1	Write T1L-L; Read T1CL, Clear T1IF
A005		UT1CH	1	Write T1L-H & T1C-H, T1L-L-T1C-L, Clear T1IF; Read T1C-H
A006		UT1LL	1	Write T1L-L; Read T1L-L

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A007	User R6522 VIA (Z1)	UT1LH	11	Write T1L-H, Clear T1IF; Read T1L-H
A008		UT2L	1	Write T2L-L; Read T2C-L, Clear T2IF;
A009		UT2H	1	Write T2C-H, T2L-L-T2C-L Clear T2IF; Read T2C-H
A00A		USR	1	Shift Register (SR)
A00B		UACR	1	Auxiliary Control Register (ACR)
A00C		UPCR	1	Peripheral Control Register (PCR)
		BIT	SIGNAL	CONN/PIN AIM 65 SIGNAL NAME
		0	CA1	J1-19 User Defined
		1-3	CA2	J1-18
		4	CB1	J1-20
		5-6	CB2	J1-21
A00D		UIFR	1	Interrupt Flag Register (IFR)
A00E		UIER	1	Interrupt Enable Register (IER)
A00F		UDRA	1	Data Register A (No Effect on Handshake)
A010 A3FF				Not Available
A400	Interrupt Indirect	IRQV4	2	IRQ after Monitor vector
A402	Jump	NMIV2	2	NMI Vector
A404	Vectors (User Alterable)	IRQV2	2	IRQ Vector

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A406	I/O Devices (User Alter-	DILINK	2	Display Linkage (Echo to Display)
A408	able)	TSPEED	1	Tape Speed
A409	↓	GAP	1	Timing for GAPs between blocks
A40A	Monitor	NPUL	1	No. of Half Pulses
A40B	Parameters	TIMG	3	Tape Timing
A40E	and Flags	REGF	1	Register Trace Flag
A40F		DISFLG	1	Instruction Trace Flag
A410		BKFLG	1	Enable or Disable Breakpoints
A411		PRIFLG	1	Enable or Disable Printer
A412		INFLG	1	Input Device
A413		OUTFLG	1	Output Device
A414		HISTP	1	PC History Trace Pointer
A415		CURPO2	1	Display Pointer
A416		CURPOS	1	Printer Pointer
A417		CNTH30	1	Baud Rate
A418		CNTL30	1	TTY Delay
A419		COUNT	1	0-99
A41A		S1	2	Start Address
A41C		ADDR	2	End Address
A41E	↓	CKSUM	2	Checksum
A41C	Mnemonic Entry	CURAD	2	Current Address
A420	Monitor Saved	SAVPS	1	Processor Status
A421	Registers	SAVA	1	Accumulator

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A422	Monitor	SAVX	1	X Register
A423	Saved	SAVY	1	Y Register
A424	Registers	SAVS	1	Stack Pointer
A425	↓	SAVPC	2	Program Counter
A427	Page Zero	STIY	3	STA NM, Y
A42A	Work Area	CPIY	4	CMP NM, Y or LDA NM, Y and RTS
A42A	Page Zero Work	LDIY	3	LDA NM, Y
A42A	Keyboard	KMASK	1	To Mask Off CTRL or SHIFT
A42B		STBKEY	1	Strobe Key (1-8)
A42E	Cassette	NAME	5	File Name
A433	Tape	GANG	1	Output PB7
A434	↓	TAPIN	1	Input Flag (Tape 1 or 2)
A435		TAPOUT	1	Output Flag (Tape 1 or 2)
A436		TAPTR	1	Tape Buffer Pointer
A437		TAPTR2	1	Tape Output Buffer Pointer
A42E	PC Trace	HIST	10	Four Last Addresses, Plus Next Address
A42E	Mnemonic	FLAG	1	
A42F	Entry	BYTESM	2	
A431		TEMPX	1	

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A432	Mnemonic	TEMPY	1	
A433	Entry	TEMPA	1	
A435	↓	OPCODE	3	
A438		CODFLG	1	
A438	Display Buffer	DIBUFF	40	Display Buffer
A460	Printer	IBUFM	20	Print Buffer
A474	↓	IDIR	1	Direction 0=>+ FF=>-
A475	↓	ICOL	1	Column: 0 = Leftmost 4 = Rightmost
A476	↓	IOFFST	1	Offset: 0 = Left Digit, 1 = Right Dgt.
A477	↓	IDOT	1	No. of Last Dot Encountered
A478	↓	IOUTL	1	Lower 8 Outputs (8 Cols. on Right)
A479	↓	IOUTU	1	Upper 2 Digits
A47A	↓	IBITL	1	1 Bit Mask for Current Output
A47B	↓	IBITU	1	
A47C	↓	IMASK	1	Mask for Current Row
A47D	↓	JUMP	2	Indirect Address of Table for Current Row
A47F	Keyboard	ROLLPL	1	Save Last Strobe for Rollover

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A480	Monitor R6532 RIOT (Z33)	DRA2	1	Data Register A

	BIT	SIGNAL	CONN/PIN	I/O	AIM 65 SIGNAL NAME
	0	PA0	J4-2	I	Keyboard KI1
	1	PA1	J4-6	I	Keyboard KI2
	2	PA2	J4-8	I	Keyboard KI3
	3	PA3	J4-7	I	Keyboard KI4
	4	PA4	J4-11	I	Keyboard KI5
	5	PA5	J4-12	I	Keyboard KI6
	6	PA6	J4-5	I	Keyboard KI7
	7	PA7	J4-3	I	Keyboard KI8

A481	DDRA2	1	Data Direction Register A
A482	DRB2	1	Data Register B

BIT	SIGNAL	CONN/PIN	I/O	AIM 65 SIGNAL NAME
0	PB0	J4-1	O	Keyboard KO1
1	PB1	J4-10	O	Keyboard KO2
2	PB2	J4-16	O	Keyboard KO3
3	PB3	J4-15	O	Keyboard KO4
4	PB4	J4-14	O	Keyboard KO5
5	PB5	J4-13	O	Keyboard KO6
6	PB6	J4-4	O	Keyboard KO7
7	PB7	J4-9	O	Keyboard KO8

A483	DDRB2	1	Data Direction Register B
A484	DNPA7	1	Write Disable PA7 Interrupt, Negative Edge Detect

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A485	Monitor R6532 RIOT (Z33)	DPPA7	1	Write Disable PA7 Interrupt Positive Edge Detect
A486		ENPA7	1	Write Enable PA7 Interrupt Negative Edge Detect
A486		RINT	1	Read Bit 7 = Timer Flag, Bit 6 = PA7 Flag, Clear Int.
A487		EPPA7	1	Write Enable PA7 Interrupt, Positive Edge Detect
A494		DIV1	1	Div. by 0001 (Disable), Add 8 to Enable
A495		DIV8	1	Div. by 0008 (Disable), Add 8 to Enable
A496		DIV64	1	Div. by 0064 (Disable), Add 8 to Enable
A497		DI1024	1	Div. by 1024 (Disable), Add 8 to Enable
A498 A7FF				Not Available
A800	Monitor R6522 VIA (A800-ABFF) (Z32)	DRB	1	Data Register B

BIT	SIGNAL	CONN/PIN	I/O	AIM 65 SIGNAL NAME
0	PB0	J2-2	O	Printer TE9
1	PB1	J2-1	O	Printer TE10
2	PB2	J1-W	I	TTY PTR
3	PB3	-	I	S1 - KB/TTY (1=KB)
4	PB4	J1-W	O	TAPE 1A
		J1-F	O	TAPE 1B
5	PB5	J1-V	O	TAPE 2A
		J1-J	O	TAPE 2B
6	PB6	J1-Y	I	TTY SERIAL IN
7	PB7	J1-L	I	AUDIO IN
7	PB7	J1-M	O	AUDIO OUT LO
7	PB7	J1-P	O	AUDIO OUT HI

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A801	Monitor R6522 VIA (A800-ABFF) (Z32)	DRAH	1	Data Register A (Controls handshake)
		BIT	SIGNAL	CONN/PIN I/O AIM 65 SIGNAL NAME
		0	PA0	J2-11 O Printer TE1
		1	PA1	J2-10 O Printer TE2
		2	PA2	J2-9 O Printer TE3
		3	PA3	J2-8 O Printer TE4
		4	PA4	J2-7 O Printer TE5
		5	PA5	J2-5 O Printer TE6
		6	PA6	J2-4 O Printer TE7
		7	PA7	J2-3 O Printer TE8
A802		DDRB	1	Data Direction Register B
A803		DDRA	1	Data Direction Register A
A804		T1L	1	Write T1L-L; Read T1C-L, Clear T1IF
A805		T1CH	1	Write T1L-H & T1C-H, T1L-L T1C-:, Clear T1IF; Read T1C-H
A806		T1L1	1	Write T1L-L; Read T1L-L
A807		T1LH	1	Write T1L-H, Clear T1IF; Read T1L-H
A808		T2L	1	Write T2L-L; Read T2C-L, Clear T2IF;

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
A809	Monitor R6522 VIA	T2H	1	Write T2C-H, T2L-L T2C-L, Clear T2IF; Read T2C-H
A80A	(Z32)	SR	1	Shift Register (SR)
A80B		ACR	1	Auxiliary Control Register (ACR)
A80C		PCR	1	Peripheral Control Register (PCR)
		BIT	SIGNAL	CONN/PIN I/O AIM 65 SIGNAL NAME
		0	CA1	J2-13 I Printer P1
		1-3	CA2	- O Data IN/Data OUT
		4	CB1	J2-14 I Printer Start
		5-7	CB2	J2-17 O Printer M+ (motor)
A80D		IFR	1	Interrupt Flag Register (IFR)
A80E		IER	1	Interrupt Enable Register (IER)
A80F		DRA	1	Data Register A (No effect on handshake)
A810 ABFF				Not available
AC00	Monitor R6520 PIA Display (Z1)	RA	1	Data Register A
		BIT	SIGNAL	CONN/PIN AIM 65 SIGNAL NAME
		0	PA0	- Display A0
		1	PA1	- Display A1
		2	PA2	- Display DS1 CE
		3	PA3	- Display DS2 CE
		4	PA4	- Display DS3 CE
		5	PA5	- Display DS4 CE
		6	PA6	- Display DS5 CE
		7	PA7	- Display W

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION	
AC01	Monitor	CRA	1	Control Register A	
AC02	R6520	RB	1	Register B	
	PIA				
	Display	BIT	SIGNAL	CONN/PIN	AIM 65 SIGNAL NAME
	(21)	0	PB0	-	Display D0
		1	PB1	-	Display D1
		2	PB2	-	Display D2
		3	PB3	-	Display D3
		4	PB4	-	Display D4
		5	PB5	-	Display D5
		6	PB6	-	Display D6
		7	PB7	-	Display CU
AC03		CRB	1	Control Register B	
AC04		DISP1	1	Display 1 Digit 0 - Char 4	
AC05			1	Display 1 Digit 1 - Char 3	
AC06			1	Display 1 Digit 2 - Char 2	
AC07			1	Display 1 Digit 3 - Char 1	
AC08		DISP2	1	Display 2 Digit 0 - Char 8	
AC09			1	Display 2 Digit 1 - Char 7	
AC0A			1	Display 2 Digit 2 - Char 6	
AC0B			1	Display 2 Digit 3 - Char 5	
AC10		DISP3	1	Display 3 Digit 0 - Char 12	
AC11			1	Display 3 Digit 1 - Char 11	
AC12			1	Display 3 Digit 2 - Char 10	
AC13			1	Display 3 Digit 3 - Char 9	

Table 7-11. AIM 65 Detail Memory Map (Cont.)

ADDRESS	AREA	LABEL	BYTES	FUNCTION
AC20	Monitor	DISP4	1	Display 4 Digit 0 - Char 16
AC21	R6520		1	Display 4 Digit 1 - Char 15
AC22	PIA		1	Display 4 Digit 2 - Char 14
AC23	Display		1	Display 4 Digit 3 - Char 13
AC40	(Z1)	DISP5	1	Display 5 Digit 0 - Char 20
AC41			1	Display 5 Digit 1 - Char 19
AC42	↓		1	Display 5 Digit 3 - Char 18
AC43			1	Display 5 Digit 4 - Char 17
AC44 AFFF				Not Available
B000 CFFF	AIM 65 BASIC		8192	
D000 DFFF	AIM 65 Assembler		4096	
E000 FFFF	AIM 65 Monitor		8192	
FFFA	Monitor Interrupt Vectors	NMIV1	2	NMI Vector to E075
FFFC		RSET	2	RESET Vector to E08F
FFFE		IRQV1	2	IRQ Vector to E078

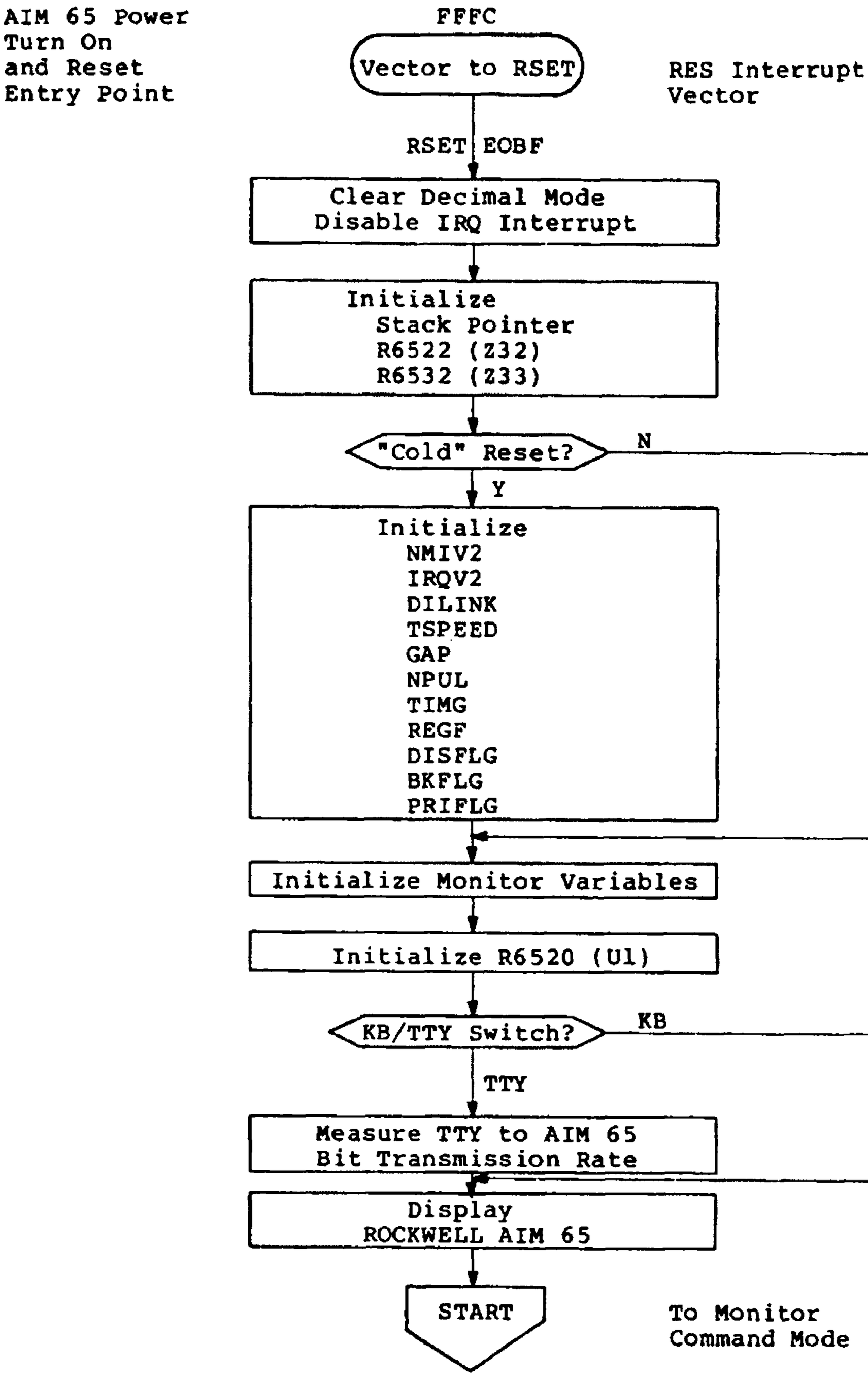


Figure 7-14. AIM 65 Software Flowchart

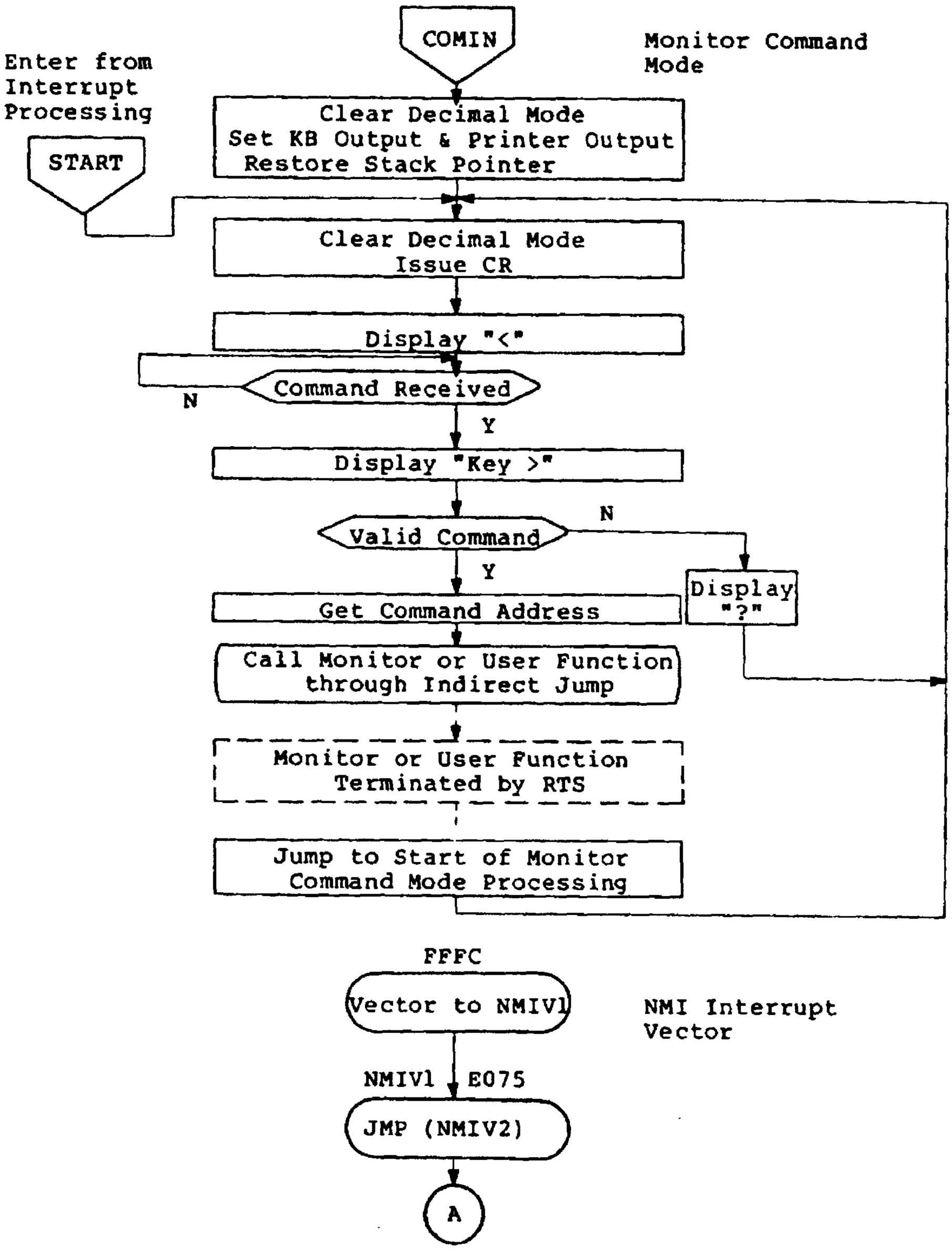


Figure 7-14. AIM 65 Software Flowchart (Cont.)

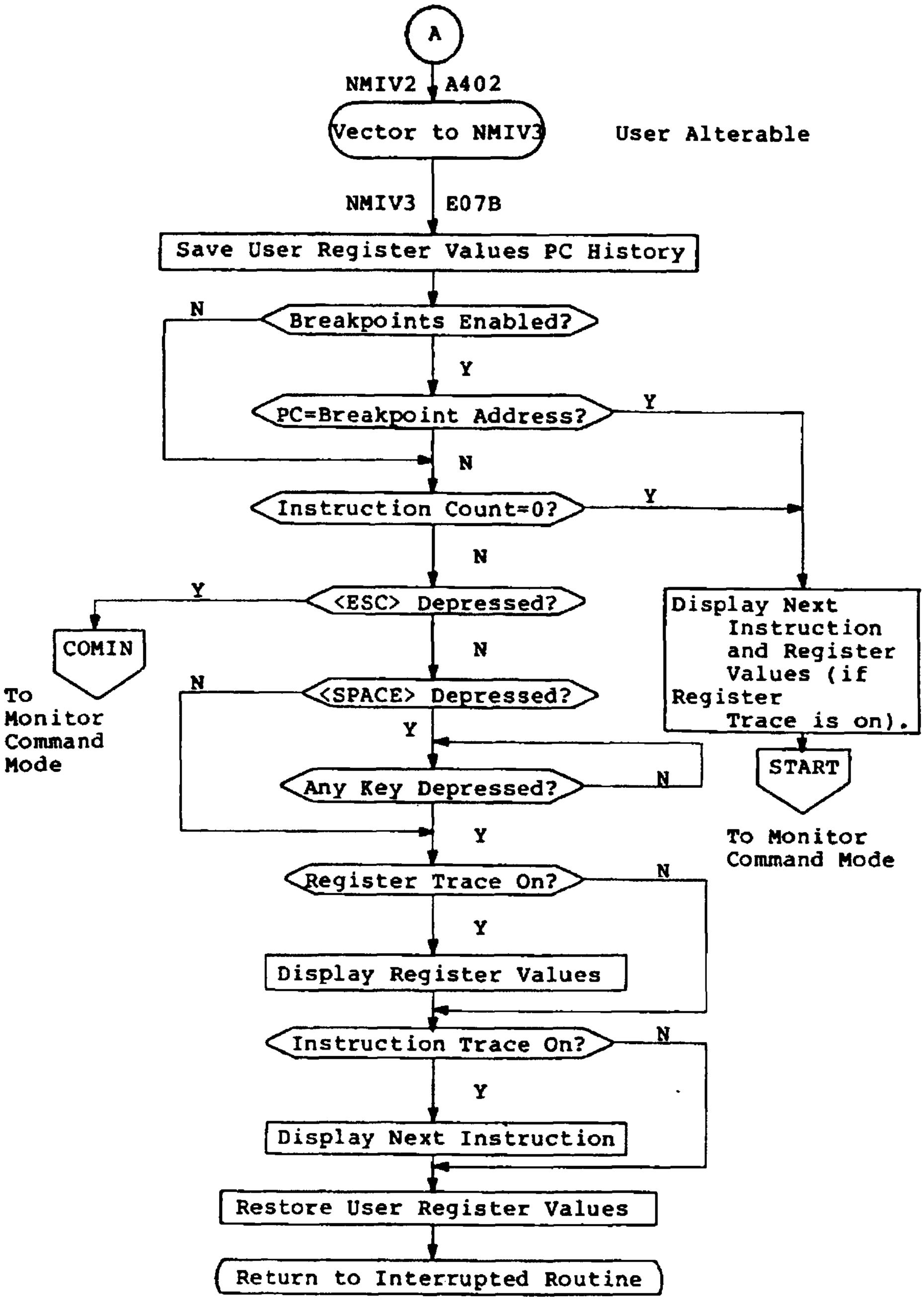


Figure 7-14. AIM 65 Software Flowchart (Cont.)

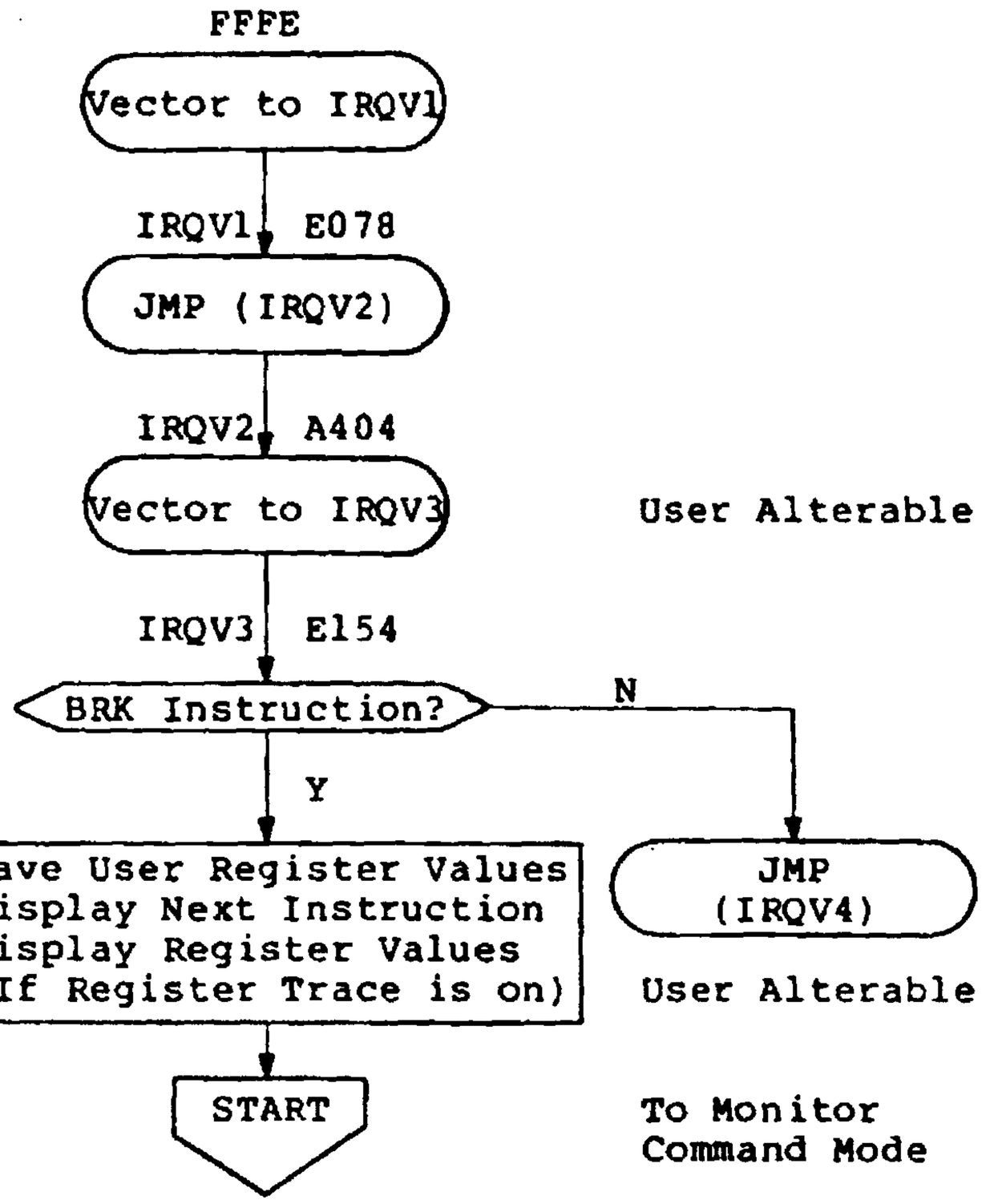


Figure 7-14. AIM 65 Software Flowchart (Cont.)

7.6 USER PARAMETERS

Table 7-12 identifies the user-alterable parameters. Some of the parameters are initialized by a "cold" reset (see Table 7-8).

Table 7-12. User-Alterable Parameters

LOCATION	NAME	BYTES	DESCRIPTION	INITIALIZED
0108	UIN	2	Vector to User Input Handler	No
010A	UOUT	2	Vector to User Output Handler	No
010C	KEYF1	3	JMP Instruction to User Function 1	No
010F	KEYF2	3	JMP Instruction to User Function 2	No
0112	KEYF3	3	JMP Instruction to User Function 3	No
A400	IRQV4	2	Vector to IRQ after Monitor Interrupt Routine	Yes
A402	NMIV2	2	Vector to NMI Interrupt Routine	Yes
A404	IRQV2	2	Vector to IRQ Interrupt Routine	Yes
A406	DILINK	2	Vector to Display Routine	Yes

Table 7-12. User-Alterable Parameters (Cont.)

LOCATION	NAME	BYTES	DESCRIPTION	INITIALIZED
A408	TSPEED	1	Audio Tape Speed Default = $C7 (AIM 65 format) Options: $5A (KIM 1 format x 1) $5B (KIM 1 format x 3)	Yes
A409	GAP	1	Audio Tape Gap Default = $08 = 32 SYN characters Option: $80 = 512 SYN characters for Assembler input and Editor update.	Yes

7.7 USER AVAILABLE SUBROUTINES

Many AIM 65 Monitor subroutines are available for development of application software. An application program designed to operate in AIM 65 while the AIM 65 Monitor ROMs are installed can use the subroutines by calling them with the Jump to Subroutine (JSR) instruction.

Application software designed to operate in hardware separate and independent from AIM 65, or in AIM 65 when the AIM 65 Monitor ROMs are removed, may use the design of these subroutines. In this case, the source instructions for the subroutines may be determined from the AIM 65 program listing.

The user available subroutines are identified and explained in Table 7-13.

AIM 65 subroutines other than those listed in Table 7-13 may be found in the AIM 65 Monitor Listing. These subroutines may be used for application software on an assumed risk basis by the user. Indiscriminate use of some of these subroutines may interfere with the proper operation of the AIM 65 Monitor.

The ENTRY ADDRESS listed in Table 7-13 may be the first address of the subroutine or may be a Jump (JMP) instruction to the first address of the subroutine. The SUBROUTINE NAME is the name used by the AIM 65 Monitor source program. Any data in registers that may be changed by the subroutine are identified in the REGISTERS ALTERED column. Any data in an altered register that is to be retained must be saved before calling the subroutine and restored after return from the subroutine by the application program.

Table 7-13. AIM 65 Monitor Subroutines

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
BLANK	E83E	A	O	Outputs one SP ($20) to Display/Printer.
BLANK2	E83B	A	O	Outputs two SP ($20) to Display/Printer.
CLR	EB44	A	M	Clears display and printer pointers.
CRCK	EA24	A	O	Outputs contents of print buffer if the print pointer is not clear.
CRLF	E9F0	A	O	Outputs one CR ($0D), one LF ($0A) and, if TTY one NUL ($00), to the active output device.
CRLOW	EA13		O	Outputs one CR ($0D) and one LF ($0A) to the display/printer.
CUREAD	FE83	A	I	Reads one character from the keyboard. Returns with the ASCII code in A. Displays cursor upon return.
DUMPTA	E56F		O	Opens a audio tape output file by setting the tape buffer pointer and moving the file name from NAME ($A42E) to the tape buffer. The rest of the tape buffer can be loaded with output data using OUTALL. The recorder 1 or 2 control

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
DUMPTA (Cont.)				will turn on automatically depending on the state of TAPOUT ($A435).
DU11	E50A			Closes a tape block if the audio tape output is active. Turns both recorder controls on. Stores CR ($0D) in INFLG ($A412) and OUTFLG (A714) to indicate input from the keyboard and output to the display/printer.
DISASM	F46C		O	Outputs the disassembly of the current instruction pointed to by SAVPC ($A425) to the active output device.
EQUAL	E7D8	A	O	Outputs one "=" ($3D)to the display/printer.
FROM	E7A3	AXY	O	Outputs "FROM=" to the display/printer and puts the address in ADDR ($A41C) and ADDR+1 ($A41D).
GETTAP	EE29	A Y	I	Reads one character from audio tape into A. The tape must be in sync to read the character properly.

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
HEX	EA7D	A	M	Converts a Hex number (0-9, A-F) in A from ASCII format to hex format and puts the result in the LSD of A and puts zero in the MSD of A. Sets Carry bit if the input character is not a hex number.
INALL	E993	A	I	Reads one ASCII character from the active input device. Returns with the input character in A. The input device code must be in INFLG ($A412) before calling.
INLOW	E8F8	A	M	Puts CR ($0D) in INFLG ($A412) to indicate keyboard input.
LOADTA	E32F		I	Searches for audio tape file with file name as specified in NAME ($A42E). Turns on recorder control 1 or 2 depending on the state of TAPIN ($A434). Loads only the first block of the located file. Each byte of the loaded block can then be read using INALL. Subsequent blocks will be loaded automatically whenever the tape input buffer is empty.

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
LOADTA (Cont.)				Subroutine LOAD1 can be used after LOADTA to load the remaining blocks.
LL	E8FE	A	M	Puts CR ($0D) in INFLG ($A412) to indicate input from the keyboard and puts CR ($0D) in OUTFLG ($A413) to indicate output to the display/printer
NOUT	EA51	A	O	Converts bits 0-3 of A to an ASCII character and outputs it to the active output device.
NUMA	EA46	A	O	Converts two hex numbers in A from hex format to ASCII format (MSD first) and outputs them to the output device.
OUTLOW	E901	A	M	Puts CR ($0D) in OUTFLG ($A413) to indicate output to the display/printer rather than to the active output device.
OUTALL	E9BC		O	Outputs one ASCII character in A to the active output device.

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
OUTDIS	EF05			Outputs an ASCII character in A to the display. If more than 20 characters are displayed, the display is scrolled to the left. Stops scrolling when 60 characters have been displayed.
OUTDP	EEFC			Outputs an ASCII character to the display and to the print buffer (calls OUTPRI). Links to the display subroutine (OUTDIS) indirectly through user alterable vector DILINK ($A406).
OUTPRI	F000			Outputs an ASCII character in A to the print buffer. Prints a line when 20 characters are in the buffer or a CR ($0D) in output.
OUTPUT	E97A		O	Outputs one ASCII character from A to the display/ printer or TTY. If Bit 7 of PRIFLG ($A411) = 1, the character is also output to the printer (if the printer is enabled). If the TTY is active and bit 0 of PRIFLG = 0, the output is to the TTY.

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
OUTTAP	F24A	Y	O	Outputs one character from A to the tape. The SYN ($16) characters must be previously output to tape using TAOSET.
PACK	EA84	A	M	Converts a hex number in A from ASCII format to hexa-decimal format and puts the result in the LSD of A and the result of the last call to PACK in the MSD of A.
PHXY	EB9E		M	Push X and Y register contents onto the stack.
PLXY	EBAC	XY	M	Pull X and Y register contents from the stack.
PSL1	E8E7	A	O	Outputs "/" ($2F) to the display/printer.
QM	E7D4	A	O	Outputs "?" ($3F) to the display/printer.
RBYTE	E3FD	A	I	Reads and displays two char-acters from the active input device. If the input is a hex number in ASCII format, the numbers are converted to hex format and packed into one byte

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
RCHEK	E907	AXY	I	The keyboard is scanned. If no key is depressed return is to the calling routine. If <ESC> is depressed, return is to the Monitor. If <SPACE> is depressed, the subroutine waits until another key is depressed to continue.
RDRUB	E95F	A Y	I	Reads one character from the keyboard. Echos the input code to the display/printer. Allows rubout to delete the character if Y ≠ 0.
READ	E93C	A	I	Reads one ASCII character from the keyboard. Returns with the ASCII code in A.
RED1	FE96	A	I	Reads one ASCII character from the keyboard. Echoes the input code to the display/printer, unless the character is a CR ($0D).
REDOUT	E973	A	I	Reads one ASCII character from the keyboard. Echos the input code to the display/printer if the character is not a CR ($0D). Displays cursor upon return.
SEMI	E9BA	A	O	Outputs one ";" ($3B) to the active output device.
TAISET	EDEA		M	Sets up recorder 1 or 2 input depending on the state of TAPIN ($A434).

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
TAISET	EDEA		M	(Cont.) Checks for SYN ($16) character. Returns to calling routine upon detection of five consecutive SYN characters.
TAOSET	F21D		O	Sets up recorder 1 or 2 output depending upon the state of TAPOUT ($A435). Outputs the number of SYN ($16) characters equal to GAP ($A409) times 4.
TIBYTE	ED3B	A		Loads one input character byte from the audio tape buffer into A. Inputs a block of data from the recorder if the tape buffer is empty.
TIBY1	ED53		I	Loads a block of 80 bytes from audio tape into the tape buffer when BLK ($0115) = 0. The desired tape block number is stored in TABUFF ($0116) before calling TIBY1.
TOBYTE	F18B	A		Stores one output character (byte) in the audio tape buffer. Outputs a block of data to the recorder if the tape buffer is full.
TO	E7A7	AXY	O	Outputs "TO" to the display/ printer and puts the entered address in ADDR ($A41C) and ADDR+1 ($A41D).

Table 7-13. AIM 65 Monitor Subroutines (Cont.)

SUBRTN NAME	ENTRY ADDR.	REG ALT.	I O	FUNCTION
WHEREI	E848	AXY	I	Determines and sets up the active input device from the answer to "IN=" and puts the device code into INFLG ($A412): T ($54) AIM 65 Audio Tape K ($4B) KIM-1 Audio Tape U ($55) User Defined. X ($58) Dummy (no output) OTHER Display/Printer
WHEREO	E871	AXY	O	Determines and sets up the active output device from the answer to "OUT=" and puts the device code into OUTFLG ($A413): T ($54) AIM 65 Audio Tape K ($4B) KIM 1 Audio Tape P ($50) Printer U ($55) User Defined X ($58) Dummy (no output) OTHER Display/Printer

7.7.1 Character Input Subroutine Considerations

The character input subroutines provide different responses to the AIM 65 display and input devices., The following table identifies which subroutines display a cursor (^) before reading the input character and/or echo the input character to the input device:

SUBROUTINE	DISPLAY CURSOR	ECHO CHARACTER	ALLOWS DELETE
CUREAD	Yes	No	No
READ	No	No	No
REDOUT	Yes	Yes	No
RED1	No	Yes	No
RDRUB	Yes	Yes	Yes

7.7.2 CR and LF Output Subroutine Considerations

The CR and LF subroutines output different combinations of CR, LF, and NUL (if TTY) characters depending upon the output device code:

	<RETURN> or <SPACE> (3)						
	S3=KB		S3=TTY				
Subroutine	Display	Printer	TTY	T	K	P	U
CRLF (1)	CR	CR	CR, LF, NUL	CR	CR	CR	CR, LF
CRLOW (2)	CR	CR	CR, LF, NUL				
CRCK (2)		CR	CR, LF, NUL				

Notes: (1) To active device
(2) To input terminal only
(3) Or any character except T, K, P, or U

Example: Display and print a message - entry from F1 and return to monitor.

a. Source listing

```
=<L>
/
OUT=
*=$010C
JMP F1
;
OUTPUT=$E97A
CLR=$E844
*=$0300
;
F1 JSR CLR
LDX #$FF
LOOP INX
LDA MSG1,X
CMP #'/'
BEQ RET
JSR OUTPUT
JMP LOOP
RET RTS
;
MSG1 .BYT 'APPLICATI
ON'
.BYT ' MESSAGE:'
.END
```

b. Disassembly listing

```
<K>*=010C
/01
 010C 4C JMP 0300
<K>*=0300
/
 0300 20 JSR E844
 0303 A2 LDX #FF
 0305 E8 INX
 0306 BD LDA 0314,X
 0309 C9 CMP #2F
 030B F0 BEQ 0313
 030D 20 JSR E97A
 0310 4C JMP 0305
 0313 60 RTS
```

7.8 AIM 65 INTERRUPT LINKAGE AND HANDLING

The AIM 65 Monitor uses the three available interrupts: Reset (RES), Non-Maskable Interrupt (NMI) and Interrupt Request (IRQ). Each will occur under in response to an activation of one of the three input lines of the R6502 CPU; RES, NMI, and IRQ, respectively. Reset is used only by AIM 65. The user may bypass the AIM 65 NMI and IRQ interrupt routines and link directly to user provided interrupt routiness. Alternatively, with the IRQ interrupt, the user may link to a user provided IRQ interrupt routine after interrupt processing by the AIM 65 IRQ interrupt routine.

Before proceeding you should read Section 9 in the R6500 Programming Manual ("Reset and Interrupt Considerations") to gain insight into the R6502 interrupt processing features.

In response to any one of the interrupts, the R6502 CPU will fetch two bytes of data stored at a specific pair of addresses and load the fetched data into the program counter. The two bytes of data constitute the entry address of the corresponding interrupt routine. The R6502 CPU will then continue processing, starting at the new address in the program counter. The three address pairs, called interrupt vectors, contain vectors, or pointers, to the interrupt routine entry addresses. The address of the interrupt vectors are hardware determined and are not under the control of the programmer. The fixed addresses are:

FFFA, FFFB - NMI Interrupt Vector
FFFC, FFFD - RES Interrupt Vector
FFFE, FFFF - IRQ Interrupt Vector

RES Interrupt Handling

The RES vector points to the Reset Interrupt Routine entry point in the Monitor program ($E0BF). Upon AIM 65 power turn-on or depression of the RESET button, AIM 65 Monitor processing will begin at this address.

NMI Interrupt Handling

The NMI vector points to address $E075 (NMIVI), where a jump indirect through address $A402 (NMIV2) is coded. NMIV2 is a user-alterable variable, initialized upon a cold reset, that normally points to the AIM 65 NMI Interrupt Routine at $E07B (NMIV3).

The AIM 65 NMI Interrupt Routine performs single step processing when a user program is being executed and the RUN/STEP switch is in the STEP position. It is possible for the user to change the NMIV2 variable to point to a user provided NMI Interrupt Routine. If this is done, however, the single step processing will not be performed. Also, in this case, RUN/ STEP switch should be left in the RUN position, otherwise an NMI interrupt will be generated each time an instruction of the user program is executed.

IRQ Interrupt Handling

The IRQ interrupt vector points to address $E078 (IRQVI), which contains a jump indirect instruction through address $A404 (IRQV2). IRQV2 is a user-alterable variable, initialized upon a cold reset, that normally points to the AIM 65 IRQ Interrupt Routine.

The AIM 65 Interrupt Routine determines the cause of the interrupt. If the $\overline{IRQ}$ interrupt was caused by a BRK instruction and the RUN/STEP switch is set to RUN, the register values are saved, user program execution is terminated and control is returned to the Monitor. If the $\overline{IRQ}$ interrupt was caused by a BRK instruction and the RUN/STEP switch is set to STEP, user program execution continues.

If the $\overline{IRQ}$ interrupt was not due to a BRK instruction, it must have been caused by a user provided input. The IRQ Interrupt Routine then jumps indirectly to a user IRQ Interrupt Routine through address $A400 (IRQV4).

7.8.1 Monitor Subroutine Examples

The examples in this section show how the Monitor subroutines can be used in either program-controlled or operator-controlled procedures. Some of the Monitor input and output subroutines are similar. Experiment by replacing the given subroutines with other subroutines that appear nearly the same, to see the effect.

The examples are shown in AIM 65 assembly and disassembly form.

The equates to follow were programmed at the beginning of the assembly examples. Not all equates are required with each example, however.

```
==0000 CRCH
       =$EA24
==0000 CRLF
       =$E3F0
==0000 DUPTR
       =$E56F
==0000 DU11
       =$E50A
==0000 OUTDIS
       =$EF05
```

```
==0000 OUTPRI
       =$F000
==0000 OUTPUT
       =$E97A
==0000 INALL
       =$E993
==0000 OUTALL
       =$E95C
==0000 LOAD
       =$E2E6
==0000 LOAD1
       =$E2E9
==0000 LOADTA
       =$E32F
==0000 LL
       =$E8FE
==0000 RDRUB
       =$E95F
==0000 READ
       =$E93C
==0000 REDOUT
       =$E973
==0000 TIBYTE
       =$ED2B
==0000 TOBYTE
       =$F18B
==0000 WHERE1
       =$E848
==0000 WHERE0
       =$E871
==0000 PRIFLG
       =$A411
==0000 INFLG
       =$A412
==0000 OUTFLG
       =$A413
==0000 NAME-
       =$A42E
==0000 TAPIN
       =$A434
==0000 TAPOUT
       =$A435
==0000 DRB
       =$A300
```

1. Display and print an assembled message - under program control. The printer control must be ON in order to print. The message is terminated by a semicolon. Type F1 to enter; the program returns to the Monitor upon completion.

(Aim 65 assembly format)

```
----------------------
FUNCTION 1 LINKAGE

==0200
        *=$010C
4C0002 JMP EX1
----------------------
MAIN PROGRAM

==010F
        *=$0200
==0200 EX1
20F0E3 JSR CRLF
A2FF   LDX #$FF
==0205 LOOP1
E8     INX
BD1702 LDA MSG1,X
C93B   CMP #';'
F006   BEQ PRT
20BCE9 JSR OUTALL
4C0502 JMP LOOP1
==0213 PRT
2024EA JSR CRCK
==0216 EX1R
60     RTS
----------------------
MESSAGE

==0217 MSG1
2055   .BYT ' USER M
ESSAGE;'
```

(AIM 65 disassembly format)

```
<K>*=010C
/
 010C 4C JMP 0200
<K>*=0200
/
 0200 20 JSR E9F0
 0203 A2 LDX #FF
 0205 E8 INX
 0206 BD LDA 0217,X
 0209 C9 CMP #3B
 020B F0 BE0 0213
 020D 20 JSR E96C
 0210 4C JMP 0205
 0213 20 JSR E924
 0216 60 RTS
```

2. Input from keyboard and echo input character and cursor to the display, then output to the printer - under program control. Type F2 to enter. Type ESC return to the Monitor.

(AIM 65 assembly format)

```
----------------------
FUNCTION 2 LINKAGE

==0225
        *=$010F
4C3002 JMP EX2
----------------------
MAIN PROGRAM

==0112
        *=$0230
==0230 EX2
20F0E9 JSR CRLF
A90D   LDA #$0D
8D12A4 STA INFLG
A950   LDA #'P'
8D13A4 STA OUTFLG
==023D LOOP2
2073E9 JSR REDOUT
20BDE9 JSR OUTALL
4C3D02 JMP LOOP2
```

(AIM 65 disassembly format)

```
<K>*=010F
/
 010F 4C JMP 0230
```

```
<K>*=0230
/
 0230 20 JSR E9F0
 0233 A9 LDA #0D
 0235 8D STA A412
 0238 A9 LDA #50
 023A 8D STA A413
 023D 20 JSR E973
 0240 20 JSR E98C
 0243 4C JMP 023D
```

3. Input from keyboard without echoing input character and cursor to the display, followed by output to the display and printer -- under program control.

(AIM 65 assembly format)

```
--------------------
FUNCTION 3 LINKAGE

==0246
        *=$0112
4C5002 JMP EX3
--------------------
MAIN PROGRAM

==0115
        *=$0250
==0250 EX3
20F0E9 JSR CRLF
20FEE9 JSR LL
==0256 LOOP3
207DE9 JSR READOUT
4C5602 JMP LOOP3
```

(AIM 65 disassembly format)

```
<K>*=0112
/
 0112 4C JMP 0250

<K>*=0250
/
 0250 20 JSR E9F0
 0253 20 JSR E848
 0256 20 JSR E871
 0259 4C JMP 025C
 025C 20 JSR E9F0
 025F A9 LDA #00
 0261 8D STA A413
 0264 20 JSR E93C
 0267 20 JSR E9BC
 026A 4C JMP 0264
```

4. Input from a user-specified device and output to a user-specified device -- under operator control, excluding user-defined I/O. Type F1 to enter and ESC to exit. Note that typing RETURN in response to both IN= and OUT= prompts will display each entered digit twice, since the input is echoed to the display on entry as well as displayed to the printer on output.

(AIM 65 assembly format)

```
--------------------
FUNCTION 1 LINKAGE

==0000
       *=$010C
==010C
4C7002 JMP EX4
--------------------
MAIN PROGRAM

==010F
       *=$0270
==0270 EX4
20F0E9 JSR CRLF
2048E8 JSR WHEREI
```

(AIM 65 assembly format) cont.

```
20F0E9 JSR CRLF
20T1E6 JSR WHERE0
==027C LOOP4
20S1E9 JSR INPLL
20T6E9 JSR OUTPUT
4CT102 JMP LOOP4
```

(AIM 65 disassembly format)

```
<K>*=010C
/
 010C 4C JMP 0270
<K>*=0270
/
 0270 20 JSR E9F0
 0273 20 JSR E840
 0276 20 JSR E9F0
 0279 20 JSR E871
 027C 20 JSR E993
 027F 20 JSR E97A
 0282 4C JMP 027C
```

5. Input from the keyboard and output to the display and printer -- under program control, with user function linkage. Enter F2 to enter the program and ESC to return to the Monitor.

(AIM 65 assembly format)

```
--------------------
LINKAGE

;USER DEFINED INPUT
==0285
        *=$0100
ASCI    WOR IN5
;USER DEFINED OUTPUT
==010A
        *=$010A
ASCI    WOR OUT5
;USER DEFINED F2
```

(AIM 65 assembly format) cont.

```
     ==010C
             *=$010F
     4C9002 JMP EX5
     ----------------------
     MAIN PROGRAM

     ==0112
             *=$0290
     ==0290 EX5
     20F0E9 JSR CRLF
     A930   LDA #$30
     8D11A4 STA PRIFLG
     A955   LDA #'U'
     8D12A4 STA INFLG
     8D13A4 STA OUTFLG
     ==02A0 LOOP5
     2033E9 JSR INALL
     20BCE9 JSR OUTALL
     4CA002 JMP LOOP5

     ----------------------
     INPUT PROCESSING

     ==02A9 IN5
     203CE9 JSR READ
     60     RTS

     ----------------------
     OUTPUT PROCESSING

     ==02AD OUT5
     68     PLA
     207AE9 JSR OUTPUT
     60     RTS
```

(AIM 65 disassembly format)

```
     <K>*=010F
     /
      010F 4C JMP 0290
     <K>*=0290
     /
      0290 20 JSR E9F0
      0293 A9 LDA #30
      0295 8D STA A411
      0298 A9 LDA #55
      029A 8D STA A412
      029D 8D STA A413
      02A0 20 JSR E933
      02A3 20 JSR E3BC
```

(AIM 65 disassembly format) cont.

```
0296 4C JMP 0290
0299 20 JSR E91C
029C 60 RTS
029D 68 PLA
029E 20 JSR E97A
02A1 60 RTS
```

6. Input from user-specified device and output to user-specified device -- under operator control, with user function linkage. User-defined I/O linkage is also included. Type F3 to enter. Type U in response to IN= and OUT= prompts, to link to user-defined I/O. Type ESC to return to the Monitor.

(AIM 65 assembly format)

```
----------------------
LINKAGE

;USER-DEFINED INPUT
==0282
        *=$0100
0500    WOZ IN5
;USER DEFINED OUTPUT
==010A
        *=$010A
0508    WOZ OUT5
;USER DEFINED FI
==0100
        *=$0110
4C0300  JMP ENT

----------------------
MAIN PROGRAM

==0115
        *=$0200
==0200 ENT
20F0E9 JSR CRLF
20A3E9 JSR AHEXED
==0206 LOOP5
20B1E9 JSR INALL
20B0E5 JSR OUTALL
```

(AIM 65 assembly format) cont.

```
    4CBC02 JMP LOOPE
    ---------------------
    INPUT PROCESSING

    ==02C5 IN6
    6001   BCS IN6N
    60     RTS
    ==02C8 IN6N
    20ICE9 JSR READ
    60     RTS
    ----------------------
    OUTPUT PROCESSING

    ==02CC OUT6
    6004   BCS OUT6N
    20F0E9 JSR CRLF
    60     RTS
    ==02D2 OUT6N
    68     PLA
    20THE9 JSR OUTPUT
    60     RTS
```

(AIM 65 disassembly format)

```
<K>*=0112
/
 0112 4C  JMP  02C0

<K>*=02C0
/
 02C0 20  JSR  E9F0
 02C3 20  JSR  E848
 02C6 20  JSR  E9F0
 02C9 20  JSR  E871
 02CC 20  JSR  E993
 02CF 20  JSR  E9EC
 02D2 4C  JMP  02CC
 02D5 B0  BCS  02D8
 02D7 60  RTS
 02D8 20  JSR  E93C
 02DB 60  RTS
 02DC B0  BCS  02E2
 02DE 20  JSR  E9F0
 02E1 60  RTS
 02E2 68  PLA
 02E3 20  JSR  E97A
 02E6 60  RTS
```

7. Inputs a file from audio cassette tape -- under program control. Enter with the 5-character file name stored in TAB7, and the tape recorder number stored in IN7 ($0=recorder #1, $1=recorder #2). Insert user-defined processing of input characters. Type F1 to enter. The program returns to the Monitor upon completion of input file processing.

(AIM 65 assembly format)

```
---------------------
FUNCTION 1 LINKAGE

==0000
        *=$010C
==010C
4C0603 JMP EX7
---------------------
MAIN PROGRAM

==010F
        *=$0300
==0300 IN7
        *=*+1
==0301 TAB7
        *=*+5
==0306 EX7
A954    LDA #'T'
8D12A4 STA INFLG
AD0003 LDA IN7
8D34A4 STA TAPIN
A004    LDY #4
==0313 LOOP7
B90103 LDA TAB7,Y
992EA4 STA NAME,Y
88      DEY
10F7    BPL LOOP7
202FE3 JSR LOADTA
==031F LOOP7A
2093E9 JSR INALL
;INSERT USER
;PROCESSING OF INPUT
;CHARACTER---
```

(AIM 65 assembly format) cont.

```
;JMP TO EX7R UPON
;DETECTION OF END
;OF FILE
4C1F03 JMP LOOP7A
==0325 EX7R
A9CF   LDA #$CF
;TURN OFF RECORDERS
2D00A0 AND DRB
8D00A3 STA DRB
60     RTS
```

(AIM 65 disassembly format)

```
<K>*=010C
/
 010C 4C JMP 0306

<K>*=0306
/
0306 A9 LDA #54
0308 8D STA A412
030B AD LDA 0300
030E 8D STA A434
0311 A0 LDY #04
0313 B9 LDA 0301,Y
0316 99 STA A42E,Y
0319 88 DEY
031A 10 BPL 0313
031C 20 JSR E32F
031F 20 JSR E993
0322 4C JMP 031F
0325 A9 LDA #CF
0327 2D AND A800
032A 8D STA A800
032D 60 RTS
```

8. Outputs a file to audio tape -- under program control. Enter with the 5-character file name stored in TAB8 and the tape recorder number stored in OUT8 ($0=recorder #1, $1=recorder #2). Insert user-defined processing to generate output characters. Type F2 to enter. The program returns to the Monitor upon completion of output file processing.

(AIM 65 assembly format)

```
-------------------
FUNCTION 2 LINKAGE

==012E
       *=$010F
4C4601 JMP EX8
-------------------
MAIN PROGRAM

==0112
       *=$0140
==0140 OUT8
       *=*+1
==0141 TAB8
       *=*+5
==0146 EX8
A954   LDA #'T'
8D13A4 STA OUTFLG
AD4001 LDA OUT8
8D35A4 STA TAPOUT
A004   LDY #4
==0153 LOOP8
B94101 LDA TAB8,Y
992EA4 STA NAME,Y
88     DEY
10F7   BPL LOOP8
206FE5 JSR DUMPTA
==015F LOOP8A
;INSERT USER
;PROCESSING TO PUT
;OUTPUT CHARACTER
;IN A ---
;JMP TO EX8A UPON
;END OF OUTPUT DATA
20BCE9 JSR OUTALL
```

(AIM 65 assembly format) cont.

```
405F03 JMP LOOP6A
;CLOSE SOURCE FILE
;WITH TWO CR'S
==0165 ENDA
20F0E5 JSR CRLF
20F0E9 JSR CRLF
200AE5 JSR DU11
A9CF   LDA #$CF
;TURN OFF RECORDER
2000A6 AND DRB
8D00A8 STA DRB
==0176
60     RTS
```

(AIM 65 disassembly format)

```
<K>*=010F
/
 010F 4C JMP 0346

<K>*=0346
/
 0346 A9 LDA #54
0348  8D  STA  A413
034B  AD  LDA  0340
034E  8D  STA  A435
0351  A0  LDY  #04
0353  B9  LDA  0341,Y
0356  99  STA  A43E,Y
0359  88  DEY
035A  10  BPL  0353
035C  20  JSR  E56F
035F  20  JSR  E95C
0362  4C  JMP  035F
0365  20  JSR  E9F0
0368  20  JSR  E9F0
036B  20  JSR  E50A
036E  A9  LDA  #CF
0370  2D  AND  A000
0373  8D  STA  A000
0376  60  RTS
```

SECTION 8
THE R6522 VERSATILE INTERFACE ADAPTER

8.1 MICROCOMPUTER INPUT/OUTPUT SECTIONS

Input/output is a major consideration in almost all microcomputer applications. The greatest difficulty is that each input and output device has unique features. Devices differ in the following ways:

1. Data rates

2. Codes

3. Response time

4. Data format, e.g., serial or parallel

5. Control signals required to synchronize transfers and determine operating modes

6. Status signals that reflect the progress of transfers and the state of the device

7. Error detection and correction facilities

Thus frequently the I/O section of a microcomputer is more complicated and more expensive than all the other sections. It is also different for each application. The I/O section

may contain latches, flip-flops, buffers, drivers, shift registers, counters, timers, decoders, and multiplexers. These components may easily occupy one or several circuit boards.

Even then, a complex I/O section may not be well-suited to a given application. It may not offer the required numbers of inputs and outputs, the proper signal polarities, or the desired combinations of status and control signals. Furthermore, a complex I/O board requires extra space and power and reduces system reliability.

An alternative is to build an I/O section on a chip. This chip should contain a variety of I/O circuit elements and connections with the precise choice of features determined by the contents of accessible registers. The user then, has the equivalent of a designer's casebook at his or her command. The program simply has to establish the values of the various registers and thus choose the desired logic connections.

Typical choices include:

1. Whether ports (or individual pins) are to be inputs or outputs.

2. The polarity of signals (i.e., active-high or active-low) and active transitions (i.e., rising edge or falling edge).

3. Polling or interrupt-driven modes of operation.

4. The sequence of status and control signals.

5. Methods for deactivating status and control signals.

Note the many advantages of this approach:

1. One board can handle a variety of applications.

2. Changes and corrections can be made in software rather than in hardware.

3. The chip will take less room, dissipate less power, and be more reliable than a circuit board full of components.

4. The chip can include all the circuitry required to communicate with the microprocessor.

The problems for the user are to learn:

1. What features the chip offers.

2. How a particular set of logic connections can be implemented.

3. How to take full advantage of the chip's capabilities.

8.2 FEATURES OF THE VERSATILE INTERFACE ADAPTER

The R6522 Versatile Interface Adapter used in the AIM 65 microcomputer is an example of an entire I/O section on a chip. It contains the following:

1. Two 8-bit I/O ports (A and B). Each pin can be individually selected to be either an input or an output.

2. Four status and control lines (two associated with each port).

3. Two 16-bit counter/timers which can be used to generate or count pulses. These timers can produce single pulses or a continuous series of pulses.

4. An 8-bit shift register which can convert data between serial and parallel forms.

5. Interrupt logic so that I/O can proceed on an interrupt-driven basis. This logic includes an interrupt flag register that tells whether particular interrupts have occurred and an interrupt enable register which determines whether particular interrupts are allowed.

The Versatile Interface Adapter occupies 16 memory locations as shown in Table 8-1. The addresses are those of the user VIA on the AIM 65 board. The way that it operates is determined by the contents of 4 registers.

1. Data Direction Register A (DDRA) determines whether the pins on port A are inputs or outputs.

2. Data Direction Register B (DDRB) determines whether the pins on port B are inputs or outputs.

3. The Peripheral Control Register (PCR) determines which polarity of transition (i.e., rising edge or falling

edge) is recognized on the input status lines (CA1 and CB1) and how the other status lines (CA2 and CB2) operate.

4. The Auxiliary Control Register (ACR) determines whether the data ports are latched and how the timers and shift register operate.

Table 8-1. R6522 Memory Assignments

Location	Function		
A000	Port B Output Data Register (ORB)		
A001	Port A Output Data Register (ORA)		**Controls handshake**
A002	Port B Data Direction Register (DDRB)		0 = Input
A003	Port A Data Direction Register (DDRA)		1 = Output
	Timer	**R/$\overline{W}$ = L**	**R/$\overline{W}$ = H**
A004	T1	Write T1L-L	Read T1C-L Clear T1 Interrupt Flag
A005	T1	Write T1L-H & T1C-H T1L-L → T1C-L Clear T1 Interrupt Flag	Read T1C-H
A006	T1	Write T1L-L	Read T1L-L
A007	T1	Write T1L-H Clear T1 Interrupt Flag	Read T1L-H
A008	T2	Write T2L-L	Read T2C-L Clear T2 Interrupt Flag
A009	T2	Write T2C-H T2L-L → T2C-L Clear T2 Interrupt Flag	Read T2C-H
A00A	Shift Register (SR)		
A00B	Auxiliary Control Register (ACR)		
A00C	Peripheral Control Register (PCR)		
A00D	Interrupt Flag Register (IFR)		
A00E	Interrupt Enable Register (IER)		
A00F	Port A Output Data Register (ORA)		***No effect on handshake***

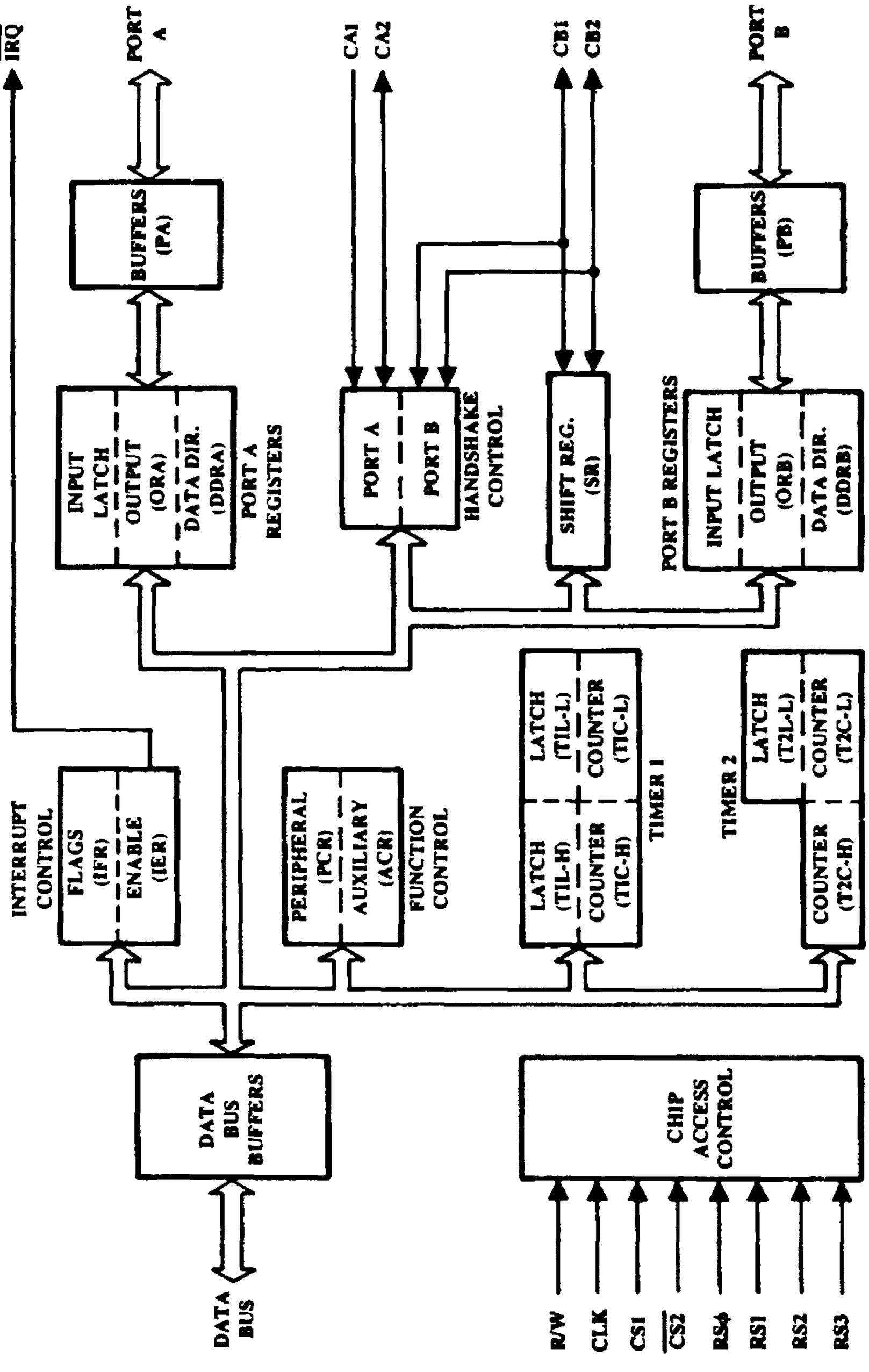


Figure 8-1. Versatile Interface Adapter

Note that there is a data direction register for each side but only one pair of control registers. Ports A and B are almost identical. One important difference is that port B can handle Darlington transistors which are used to drive solenoids and relays. We will generally use port A for input and port B for output in our example.

Figure 8-1 is a block diagram of the R6522 Versatile Interface Adapter.

8.3 SIMPLE I/O WITH THE VERSATILE INTERFACE ADAPTER

Since RESET clears all the VIA registers, disabling all interrupts and clearing all control lines, we can discuss simple I/O referring only to the data registers and the data direction registers. So simple I/O can be performed with the R6522 VIA as follows:

1. Establish the directions of the pins by storing the proper values in the data direction registers.

2. Transfer data by moving it to or from the data registers.

Note that most programs only have to execute Step 1 once since the directionality of most input and output devices is fixed (i.e., you never want to read data from a display or printer or write data to a switch or paper tape reader).

You can establish directions as follows:

1. A '0' in a bit in the data direction register makes the corresponding pin an input.

For example, a '0' in bit 4 of data direction register A makes pin PA4 into an input.

2. A '1' in a bit in the data direction register makes the corresponding pin an output.

   For example, a '1' in bit 6 of data direction register B makes pin PB6 into an output.

As for transferring data, remember that the R6502 microprocessor has no specific I/O instructions. Storing data in a VIA port that has been designated for output is equivalent to sending the data to the attached output device. Loading data from a VIA port that has been designated for input is equivalent to reading the data from the attached input device. But any instruction that acts on memory can serve as an I/O instruction if the specified address is actually an I/O device. You must be careful of the exact significance of such instructions in writing, reading, and documenting R6502 programs.

EXAMPLES

In these examples, we use the assembler designations (U in front of the register name) for the user VIA addresses in the assembler format. We use the actual addresses in the AIM 65 mnemonic format. Note that the labels for the user R6522 variables correspond to the labels in the AIM 65 memory map (Table 7-11) and Monitor program listing. The following equates were established at the beginning of the assembly source code:

```
==0000 UDRB
       =$A000
==0000 UDRAH
       =$A001
==0000 UDDRB
       =$A002
==0000 UDDRA
       =$A003
==0000 UT1L
       =$A004
==0000 UT1CH
       =$A005
==0000 UT1LL
       =$A006
==0000 UT1LH
       =$A007
==0000 UT2L
       =$A008
==0000 UT2H
       =$A009
==0000 USR
       =$A00A
==0000 UACR
       =$A00B
==0000 UPCR
       =$A00C
==0000 UIFR
       =$A00D
==0000 UIER
       =$A00E
==0000 UDRA
       =$A00F
```

1. Fetch data from a simple input port (e.g., from a set of switches or a keypad) and store it in memory location 40.

(AIM 65 assembly format)

```
==0200 EX1
A500   LDA #0
;MAKE SIDE A INPUTS
A502A0 STA UDDRA
A501A0 LDA UDF=H
;GET AND STORE DATA
8540   STA $40
```

(AIM 65 disassembly format)

```
0200 A9 LDA #00
0202 8D STA A003
0205 AD LDA A001
0208 85 STA 40
```

2. Send data to a simple output port (e.g., to a set of displays or relays) from memory location 40.

(AIM 65 assembly format)

```
==020A EX2
A9FF    LDA #$FF
;MAKE SIDE B OUTPUTS
8D03A0 STA UDDRB
A540    LDA $40
;GET AND SEND DATA
8D00A0 STA UDRB
```

(AIM 65 disassembly format)

```
020A A9 LDA #FF
020C 8D STA A002
020F A5 LDA 40
0211 8D STA A000
```

You can mix inputs and outputs on a single port by establishing the directions of individual pins appropriately. Note that you can read the states of data pins (e.g., with LDA UDRA or LDA UDRB) even if they have been designated as outputs. The B side is buffered so that it can always be

read correctly, however, the A side is not buffered so that it can only be read correctly if it is lightly loaded (or designated as inputs).

8.4 RECOGNIZING STATUS SIGNALS

If the I/O device is more complex, we cannot simply transfer data to or from it at will. In the input case, the processor must know when new data is available (e.g., a key has been pressed on a keyboard or a tape reader has read another character). In the output case, the processor must know whether the device is ready to receive data (e.g., a printer has finished printing the last character or a modem has completed the previous transmission).

Normally, the input or output device provides a status signal. A transition on that line indicates the availability of data or the readiness of the device. The microcomputer I/O section must recognize the transition and allow the processor to determine that it has occurred.

You can handle this kind of I/O with the R6522 Versatile Interface Adapter as follows:

1. Attach the peripheral status input to input CA1 or CB1.

2. Determine which edge on the status line will be recognized by assigning a value to control register bit 0 (CA1) or 4 (CB1). A value of zero in that bit position means that the interrupt flag will be set by a high-to-low transition (or falling edge). A value of

one means that the interrupt flag will be set by a low-to-high transition (or rising edge).

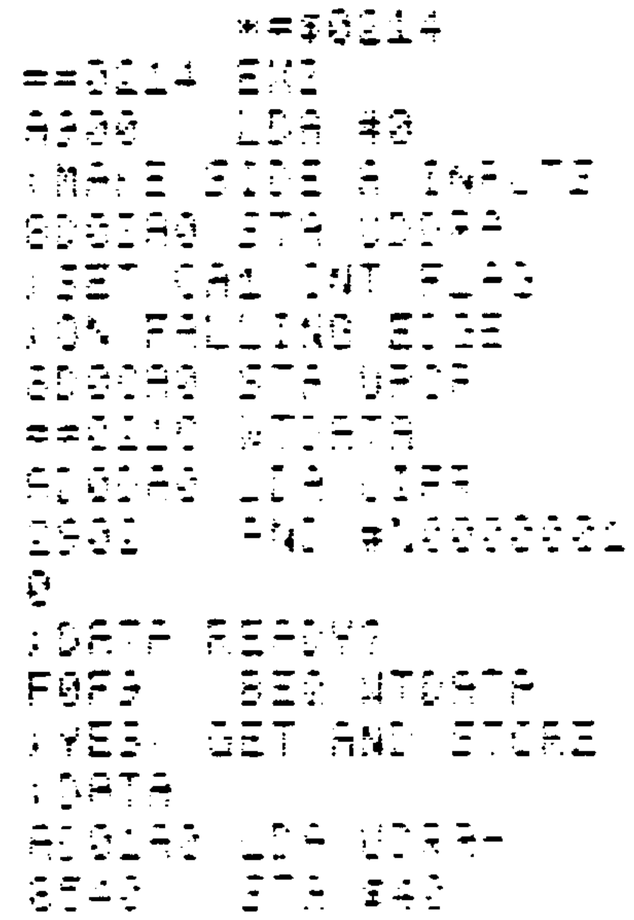
3. Determine whether a transition has occurred by examining bit 1 (CA1) or 4 (CB1) of the interrupt flag register. The bit will be one if a transition has occurred.

4. Reset the interrupt flag by reading or writing the corresponding data register. The flag is then ready to be used in the next operation.

Let us now look at some examples:

1. Fetch data from an input port with an active high-to-low DATA READY strobe and place the data in memory location 40.

(AIM 65 assembly format)



(AIM 65 disassembly format)

```
0214 A9 LDA #30
0216 8D STA A001
0219 8D STA A003
021C A9 LDA #00
021E 29 AND #32
0220 F0 BEQ 021C
0222 AD LDA A291
0225 85 STA 40
```

Clearing the Peripheral Control Register is unnecessary if the routine is starting from a reset. Note that reading the Output Data Register with LDA UDRAH clears the interrupt flag so that it is available for the next DATA READY signal.

2. Send data to an output port with an active high-to-low PERIPHERAL READY strobe. Get the data from memory location 40 and send it when the peripheral is ready.

(AIM 65 assembly format)

```
==0220 EM4
A9FF   LDA #$FF
;MAKE SIDE B OUTPUTS
8002A0 STA UDDRB
A500   LDA #0
;SET CB1 INT FLAG
;ON FALLING EDGE
8000CA STA UPCR
==0232 WTRDY4
A020A0 LDA UIFR
2910   AND #%00001000
0
;PERIPHERAL READY?
F0F8   BEQ WTRDY4
;YES.GET AND SEND
;DATA
A540   LDA $40
8D00A0 STA UDRB
```

(AIM 65 disassembly format)

```
0213 A9 LDA #FF
0214 8D STA A002
0217 A0 LDA #10
0219 8D STA A001
0218 AD LDA A000
021F 29 AND #10
0221 F0 BEQ 0211
0223 A5 LDA 40
0225 8C STA A000
```

Note that sending the data to the Output Data Register (STA UDRB) clears the interrupt flag so that it is available for the next PERIPHERAL READY signal.

3. Fetch data from an input port with an active low-to-high DATA READY strobe and place the data in memory location 40.

(AIM 65 assembly format)

```
==023E EXS
A900    LDA #0
;MAKE SIDE A INPUTS
8D01A0 STA UDDRA
A901    LDA #1
;SET CA1 INT FLAG
;ON RISING EDGE
8D0CA0 STA UPCR
==0243 WTDATS
AD0DA0 LDA UIFR
2902    AND #%00000001
0
;DATA READY?
F0F9    BEQ WTDATS
;YES, GET AND STORE
;DATA
AD01A0 LDA UORAH
8540    STA $40
```

(AIM 65 disassembly format)

```
023E A9 LDA #00
0240 8D STA A001
0243 F9 LDA #01
0245 8D STA A001
0248 AD LDA A000
024B 29 AND #02
024D F0 EB BEQ 0248
024F AD LDA A001
0252 85 STA 40
```

4. Send data to an output port with an active low-to-high PERIPHERAL READY strobe. Get the data from memory location 40 and send it when the peripheral is ready.

(AIM 65 assembly format)

```
==0254 EX6
A9FF    LDA #$FF
;MAKE SIDE B OUTPUTS
8D02A0 STA UDDRB
;SET CB1 INT FLAG ON
;RISING EDGE
A910    LDA #%00001000
0
8D0CA0 STA UPCR
==025E WTRDY6
AD0DA0 LDA UIFR
2910    AND #%00001000
0
;PERIPHERAL READY?
F0F3    BEQ WTRDY6
;YES,GET & SEND DATA
A540    LDA $40
8D00A0 STA UDRB
```

```
0254 A9 LDA #FF
0256 8D STA A002
0259 A9 LDA #10
025B 8D STA A00C
025E AD LDA A00D
0261 29 AND #10
0263 F0 BEQ 025E
0265 A5 LDA 40
0267 8D STA A000
```

Note that the VIA has both input and output latches. The output latches are always enabled; output data is latched when it is stored in an output data register. The input latches, if they are needed, can be enabled by setting Bit 0 (side A) or Bit 1 (side B) of the Auxiliary Control Register. The input data will then be latched by the active transition on CA1 or CB1.

8.5 PRODUCING OUTPUT STROBES

The peripheral may also require information about when a transfer has occurred or whether the port is ready to receive data. For example, devices such as digital-to-analog converters commonly require a LOAD pulse to enter data into the converter. A multiplexed display requires an output signal that directs the next output properly. A communications device may need a signal to indicate that an input buffer is available or that an output buffer is full. Output signals may also be needed to turn devices on or off, activate operator displays, or control operating modes.

You can handle this kind of I/O with the R6522 Versatile Interface Adapter as follows:

1. Attach the control output to CA2 or CB2.

2. Make CA2 (CB2) into an output by setting control register bit 3 (7).

3. Make CA2 (CB2) into a pulse by clearing control register bit 2 (6) or into a level by setting that bit.

4. If CA2 (CB2) is a pulse, make it into a handshake signal (low from the time the Output Register is read or written until the next active transition on CA1 (CB1) by clearing control register bit 1 (5) or into a single-cycle strobe by setting that bit.

5. If CA2 (CB2) is a level, determine its value by clearing or setting bit 1 (5).

So the options are:

1. CA2 goes low when the processor transfers data to or from Output Register A and goes high when the next active transition occurs on CA1. The signal can indicate that the port is ready for more data or that output data is available. The peripheral's response then indicates that it has sent more data or has processed the previous data.

2. CA2 goes low when the processor transfers data to or from Output Register A and goes high after one clock cycle. This signal indicates that an input or output operation has occurred and can be used for multiplexing.

3. CA2 is a level controlled by the value of control register bit 1. This signal can provide an active-high or low pulse of arbitrary length. It can be used to load registers, turn devices on or off, or control operating modes.

Let us now look at some examples:

1. Fetch data from an input device that requires a hand-shake signal and that produces an active high-to-low DATA READY strobe. Place the data in memory location $40.

(AIM 65 assembly format)

```
==026A EX7
A900   LDA #0
;M4KE SIDE A INPUTS
8D03A0 STA UDDRA
;SET CA2 HANDSHAKE
;OUTPUT MODE WITH
;CA1 INT FLAG SET
;ON FALLING EDGE
A908   LDA #%00000100
0
8D0CA0 STA UPCR
==0274 WTDAT7
AD0DA0 LDA UIFR
2902   AND #%00000001
0
;DATA READY?
F0F9   BEQ WTDAT7
;YES, GET AND STORE
;DATA AND SEND DATA
;TAKEN ON CA2
AD01A0 LDA UDRAH
8540   STA $40
```

(AIM 65 disassembly format)

```
0268 A9 LDA #00
026A 8D STA A003
026F A9 LDA #08
0271 8D STA A00C
0274 AD LDA A00D
0277 29 AND #02
0279 F0 BE0 0274
027B AD LDA A001
027E 85 STA 40
```

The Peripheral Control Register bits are:

bits 4-7 = 0 since CB1 and CB2 are not used

bit 3 = 1 to make CA2 an output

bit 2 = 0 to make CA2 a pulse

bit 1 = 0 to make CA2 a handshake acknowledgement that remains low until the next active transition on CA1

bit 0 = 1 to make the active transition on CA1 a falling edge (high-to-low transition).

2. Fetch data from an input device that requires a brief DATA ACCEPTED strobe for multiplexing or control purposes. Place the data in memory location $40.

(AIM 65 assembly format)

```
==0160 EX2
A900   LDA #0
;MAKE SIDE A INPUTS
8D03A0 STA UDDRA
;SET CA2 PULSE
;OUTPUT MODE WITH
;CA1 INT FLAG SET
;ON FALLING EDGE
A90A   LDA #%00000101
0
8D0CA0 STA UPCR
==028A WTRDYS
AD0DA0 LDA UIFR
2302   AND #%00000001
0
;DATA READY?
F0F9   BEQ WTRDYS
;YES, GET AND STORE
;DATA AND SEND DATA
;TAKEN STROBE ON CA2
AD01A0 LDA UDRAH
8540   STA $40
```

(AIM 65 disassembly format)

```
0210 A9 LDA #00
0212 8D STA A003
0215 A9 LDA #0A
0217 8D STA A00C
021A AD LDA A00D
021D 29 AND #02
021F F0 BE0 028A
0221 AD LDA A001
0224 85 STA 40
```

Here bit 1 of the Peripheral Control Register is set to 1 to make CA2 a brief strobe lasting one cycle after the reading of Port A Output Data Register.

3. Send data to an output device that requires a handshake signal and that produces an active low-to-high PERIPHERAL READY strobe. Get the data from memory location $40 and send it when the peripheral is ready.

(AIM 65 assembly format)

```
==0296 EX3
A3FF   LDA #$FF
;MAKE SIDE B OUTPUTS
8D02A0 STA UDDRB
;SET CB2 HANDSHAKE
;OUTPUT MODE WITH
;CB1 INT FLAG SET
;ON RISING EDGE
A930   LDA #%10001000
0
8D0CA0 STA UPCR
==02A0 WTRDY9
AD0DA0 LDA UIFR
2910   AND #%00001000
0
;PERIPHERAL READY?
F0F9   BEQ WTRDY9
;YES. GET AND SEND
;DATA AND SET
;ACKNOWLEDGE
A940   LDA $40
8D00A0 STA UDRB
```

(AIM 65 disassembly format)

```
0256 A9 LDA #FF
0258 8D STA A002
025B A9 LDA #90
025D 8D STA A00C
0260 AD LDA A00D
0263 29 AND #10
0265 F0 BE0 02A0
02F7 A5 LDA 40
02F9 8D STA A000
```

The Peripheral Control Register bits are:

bit 7 = 1 to make CB2 an output

bit 6 = 0 to make CB2 a pulse

bit 5 = 0 to make CB2 a handshake acknowledgement that remains low until the next active transition on CB1

bit 4 - 1 to make the active transition on CB1 a rising edge (low-to-high transition)

bits 0-3 = 0 since CA1 and CA2 are not used.

4. Send data to an output device that requires a brief OUTPUT or DATA READY strobe for multiplexing or control purposes. Fetch the data from memory location $40.

(AIM 65 assembly format)

```
==0280 EX10
A9FF   LDA #$FF
;MAKE SIDE B OUTPUTS
8D02A0 STA UDDRB
;SET CB2 PULSE
;OUTPUT MODE
A9A0   LDA #%10100000
0
8D0CA0 STA UPCR
;GET AND SEND DATA
;AND DATA STROBE
A540   LDA $40
8D00A0 STA UORB
```

(AIM 65 disassembly format)

```
0280 A9 LDA #FF
0282 8D STA A002
0285 A9 LDA #A0
0287 8D STA A00C
028A A5 LDA 40
028C 8D STA A000
```

Here bit 5 of the Peripheral Control Register is set to 1 to make CB2 a brief strobe lasting one cycle after the writing of Port B Output Data Register.

5. Fetch data from an input device that requires an active-high START pulse. The device produces an active high-to-low DATA READY strobe. Place the data in memory location $40.

(AIM 65 assembly format)

```
==02BF EM11
A900   LDA #0
;MAKE SIDE A INPUTS
8D03A0 STA UDDRA
;RESET START PULSE
A90C   LDA #%00000110
0
8D03A0 STA UPCR
;SET START PULSE
;HIGH ON CA2
A90E   LDA #%00000111
0
8D03A0 STA UPCR
;RESET START PULSE
A90C   LDA #%00000110
0
==02D0
8D03A0 STA UPCR
==02D3 WTDT11
AD01A0 LDA UIFR
2901   AND #%00000001
0
;DATA READY?
F0F9   BEQ WTDT11
;YES, GET AND STORE
;DATA
AD01A0 LDA UDRAH
8540   STA $40
```

(AIM 65 disassembly format)

```
020F A9 LDA #00
0211 8D STA A003
0214 A9 LDA #0C
0216 8D STA A00C
0219 A9 LDA #0E
021B 8D STA A00C
021E A9 LDA #0C
0220 8D STA A00C
0223 AD LDA A00D
0226 29 AND #02
0228 F0 BE0 0203
022A AD LDA A001
022D 85 STA 40
```

Here bit 2 of the Peripheral Control Register is set to 1 to make CA2 a level with the value given by bit 1 of the Peripheral Control Register. This mode can be used to produce pulses of any length and polarity; it is called the manual output mode because there is no automatic pulse information.

In a typical application, an analog-to-digital converter or data acquisition system usually needs a START CONVERSION pulse to begin operations.

6. Send data to an output device that must be turned on before the data is sent and turned off after the data is sent (a logic 1 on a control line turns the device on). The peripheral produces an active low-to-high PERIPHERAL READY strobe. Get the data from memory location $40 and send it when the peripheral is ready.

(AIM 65 assembly format)

```
==3DF EX12
A9FF   LDA #$FF
;MAKE SIDE B OUTPUTS
8D02A0 STA UDDRB
A9F8   LDA #%1111000
0
;TURN PERIPHERAL ON
;BY SETTING CB2 HIGH
;AND SET CB1 INT
;FLAG ON RISING EDGE
8C0CA0 STA UPCR
==3IE9 WTRD12
AD0DA0 LDA UIFR
2910   AND #%0001000
0
;PERIPHERAL READY?
F0F9   BEQ WTRD12
;YES, GET & SEND DATA
A540   LDA $40
8D00A0 STA UDRB
;TURN PERIPHERAL OFF
A9D0   LDA #%1101000
0
8D0CA0 STA UPCR
```

(AIM 65 disassembly format)

```
02FF A9 LDA #FF
0301 8D STA A002
0304 A9 LDA #F0
0306 8D STA A00C
0309 AD LDA A00D
030C 29 AND #10
030E F0 EE0 02E9
0310 A5 LDA 40
0312 8D STA A000
0315 A9 LDA #D0
0317 8D STA A00C
```

In many applications, such as portable equipment, the output peripheral is only turned on when data is to be sent to it. In other applications, the processor must issue an OUTPUT REQUEST and receive an acknowledgement before sending the data.

8.6 VIA INTERRUPTS

You can easily use the R6522 Versatile Interface Adapter in an interrupt-driven mode. Figure 8-2 shows the Interrupt Enable Register (IER). Any of the various interrupt sources can be enabled by setting the corresponding enable bit. Note that the most significant bit controls how the other enable bits are affected:

If IER7 = 0, each '1' in a bit position clears an enable bit and thus disables that interrupt.

If IER7 = 1, each '1' in a bit postion sets an interrupt bit and thus enables that interrupt.

Zeroes in the enabling bit positions always leave the enable bits as they were.

Some examples should help you see how this works.

1. Enable CA1 interrupt, disable all others.

(AIM 65 assembly format)

```
==DIFA EM13
ABTD   LDA #%01111110
1
;DISABLE OTHER
;INTERRUPTS
9D9EA0 STA UIER
ABA2   LDA #%10000001
0
;ENABLE CA1
;INTERRUPT
9D9EA0 STA UIER
```

(AIM 65 disassembly format)

```
02FA A9 LDA #7D
02FC 80 STA A00E
02FF A9 LDA #82
0301 80 STA A00E
```

The first operation clears all the interrupt enables except CA1. The second operation sets the CA1 interrupt enable.

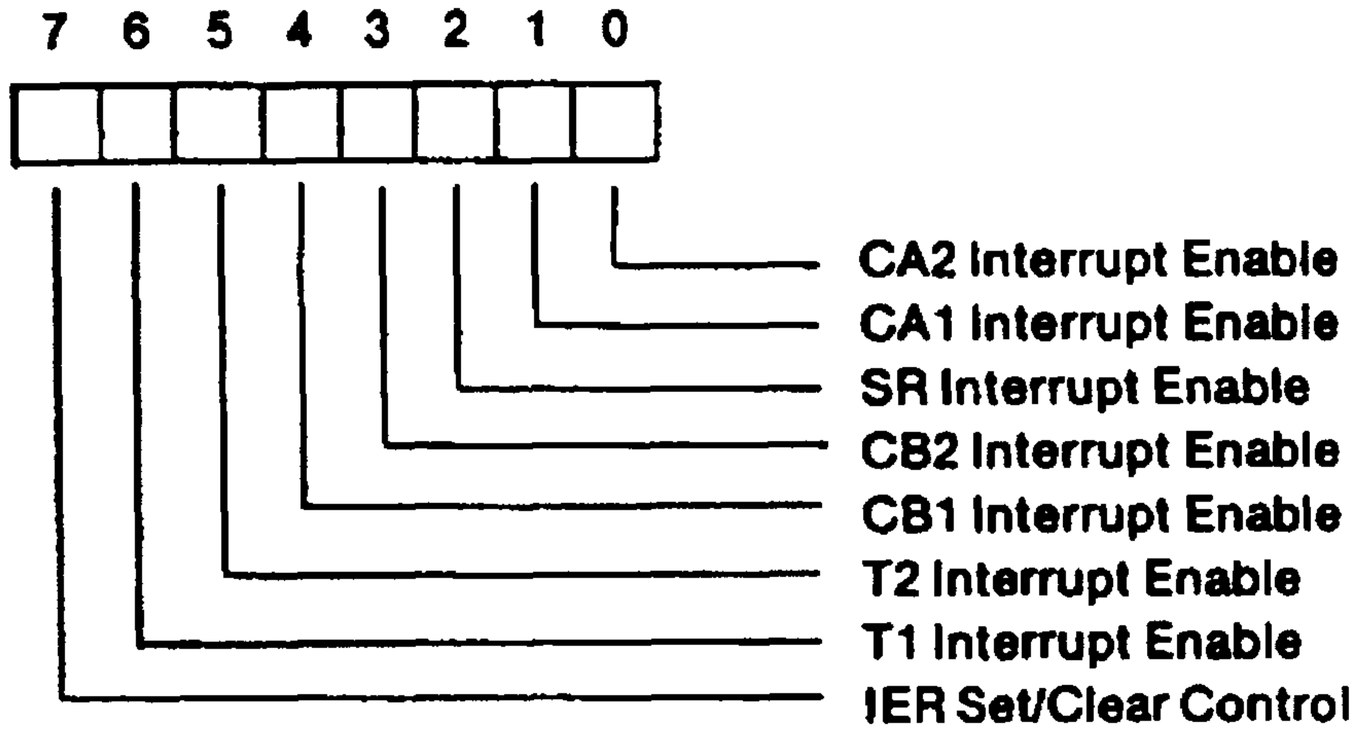
2. Enable CB1 and CB2 interrupts, disable all others.

(AIM 65 assembly format)

```
==0304 EX14
A967   LDA #%01110011
1
;DISABLE OTHER
;INTERRUPTS
6D9EA0 STA UIER
A936   LDA #%1001100
0
;ENABLE CB1 AND CB2
;INTERRUPTS
0D9EA0 STA UIER
```

(AIM 65 disassembly format)

```
0304 A9 LDA #67
0306 0D STA A00E
0309 A9 LDA #98
030B 9D STA A00E
```



INTERRUPT ENABLE BITS (IER0-6)

IERn = 0 Disable interrupt
= 1 Enable interrupt

IER SET/CLEAR CONTROL (IER7)

IER7 = 0 For each data bus bit set to logic 1, clear corresponding IER bit
= 1 For each data bus bit set to logic 1, set corresponding IER bit.

Note: IER7 is active only when R/$\overline{W}$ = L; when R/$\overline{W}$ = H, IER7 will read logic 1.

Figure 8-2. R6522 Interrupt Enable Register (IER)

Note that we could disable all interrupts in the first step.

3. Disable CA1 interrupt, leave others as they were.

(AIM 65 assembly format)

```
==010E EX15
A902   LDA #%00000001
0
;DISABLE CA1
;INTERRUPT
8D0EA0 STA UIER
```

(AIM 65 disassembly format)

```
010E A9 LDA #02
0110 8D STA A00E
```

4. Disable CB1 and CB2 interrupts, leave others as they were.

(AIM 65 assembly format)

```
==0118 EX16
A918   LDA #%00001100
0
;DISABLE CB1 AND CB2
;INTERRUPTS
8D0EA0 STA UIER
```

(AIM 65 disassembly format)

```
0210 A9 LDA #10
0214 8C STA A00E
```

The processor can determine which interrupt has occurred by examining the interrupt flag register (Figure 8-3). Note that examining bit 7 determines if any interrupts have occurred on the VIA. Note also the conditions for clearing the interrupt flags.

A typical polling sequence would be:

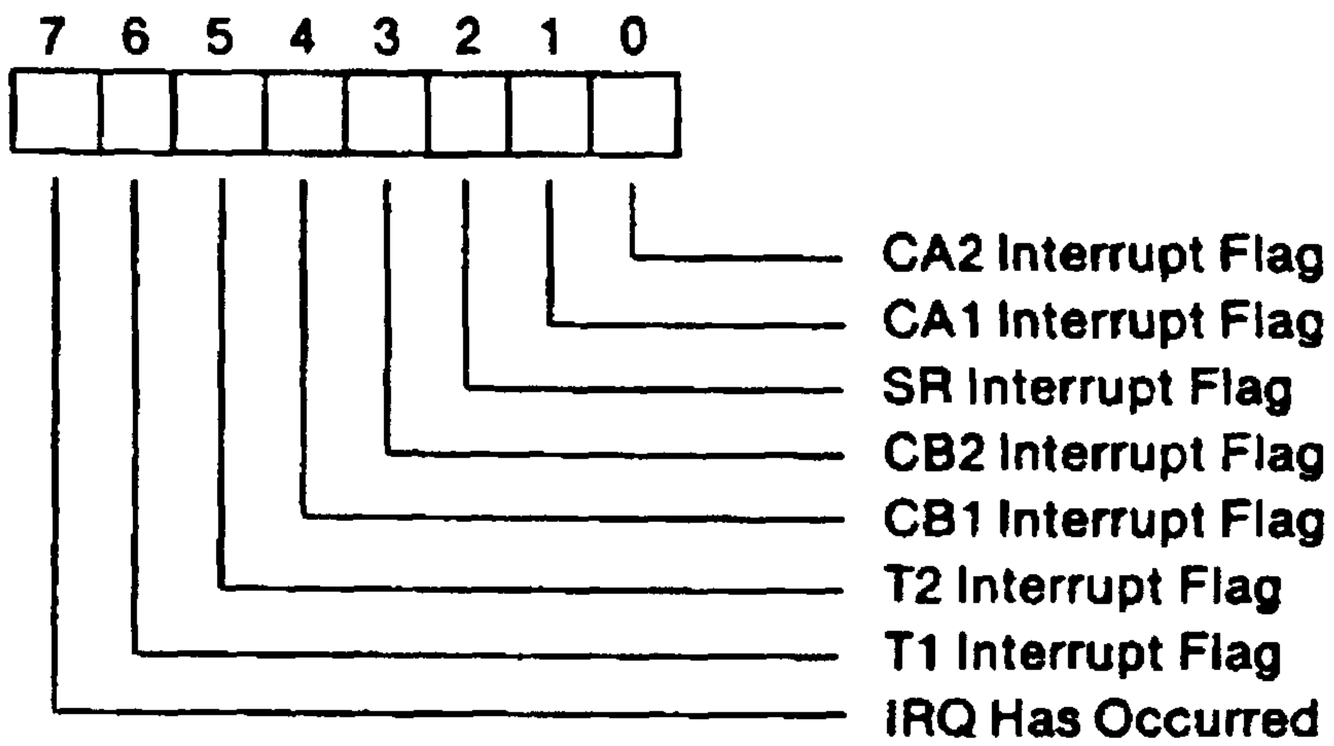
```
LDA     UIFR            ;ANY INTERRUPTS ON THIS VIA?
BPL     NEXTS           ;NO, LOOK AT NEXT SOURCE
ASL     A               ;IS INTERRUPT FROM T1?
BMI     TIM1            ;YES, GO SERVICE T1 INTERRUPT
ASL     A               ;IS INTERRUPT FROM T2?
BMI     TIM2            ;YES, GO SERVICE T2 INTERRUPT
ASL     A               ;IS INTERRUPT FROM CB1?
BMI     CB1             ;YES, GO SERVICE CB1 INTERRUPT
.
.
.
```

8.7 VIA TIMERS

The two 16-bit timer/counters in the R6522 Versatile Interface Adapters can be used to:

1. Generate a single time interval. The timer must be loaded with the number of clock pulses.

R6522 INTERRUPT FLAG REGISTER (IFR)



IFR Bit	Set By	Cleared By
0	Active transition on CA2	Reading or writing the ORA ($A001 or $A00F)
1	Active transition on CA1	Reading or writing the ORA ($A001 or $A00F)
2	Completion of eight shifts	Reading or writing the SR ($A00A)
3	Active transition on CB2	Reading or writing the ORB ($A000)
4	Active transition on CB1	Reading or writing the ORB ($A000)
5	Time-out of Timer 2	Reading T2C-L ($A008) or writing T2C-H ($A009)
6	Time-out of Timer 1	Reading T1C-L ($A004) or writing T1L-H ($A005 or $A007)
7	Any IFR bit set with its corresponding IER bit also set	Clearing IFR0-IFR6 ($A00D) or IER0-IER6 ($A00E)

Figure 8-3. R6522 Interrupt Flag Register (IFR)

2. Count pulses on pin PB6 (timer 2 only). The timer must be loaded with the number of pulses to be counted.

3. Generate continuous time intervals (timer 1 only). The timer must be loaded with the number of clock pulses per interval.

4. Produce a single pulse or a continuous series of pulses on pin PB7 (timer 1 only). The timer must be loaded with the number of clock pulses per interval.

Let us first look at timer 2, which can only be used to generate a single time interval (the so-called one-shot mode) or to count pulses on PB6. Bit 5 of the Auxiliary Control Register selects the mode:

Bit 5 = 0 for one-shot mode, 1 for pulse-counting mode

Note that 16-bit timer 2 occupies two memory locations (see Table 8-1). The first address (A008) is used to read or write the 8 least significant bits; reading this address also clears the timer 2 interrupt flag (Figure 8-3). The second address (A009) is used to read or write the 8 most significant bits; writing this address loads the counters, clears the timer 2 interrupt flag, and starts the timing operation. The completion of the operation sets the timer 2 interrupt flag.

EXAMPLES:

1. Generate a delay of 2048 (0800 hex) clock pulses.

(AIM 65 assembly format)

```
==031D EX17
A900   LDA #0
;SET T2 ONE-SHOT
;INTERVAL TIMER MODE
8008A0 STA UACR
;DELAY = 0300 CLOCKS
8009A0 STA UT2L
A903   LDA #3
;LOAD & START TIMER
800BA0 STA UT2H
A910   LDA #%00010000
0
;T2 COUNTED DOWN?
==032C WAIT17
2008A0 BIT UIFR
F0FB   BEQ WAIT17
;YES,CLR T2 INT FLAG
AD08A0 LDA UT2L
```

(AIM 65 disassembly format)

```
031D  A9  LDA  #00
031F  8D  STA  A00B
0322  8D  STA  A00A
0325  A9  LDA  #00
0327  8D  STA  A009
032A  A9  LDA  #20
032C  2C  BIT  A00D
032F  F0  BEQ  032C
0331  AD  LDA  A008
```

Note the final LDA UT2L. The sole purpose of this instruction is to clear the timer 2 interrupt flag so that it will be available for the next operation.

2. Count 5 input pulses on PB6.

(AIM 65 assembly format)

```
==0134 EX18
A900   LDA #0
;MAKE SIDE B INPUTS
9033A0 STA UDDR3
;SET T2 PULSE COUNT
;MODE
A920   LDA #%00100000
0
9036A0 STA UACR
;DELAY = 0009 CLOCKS
A905   LDA #5
9038A0 STA UT2L
A900   LDA #0
==0145
;START T2
903BA0 STA UT2H
A920   LDA #%00100000
0
;T2 COUNTED DOWN?
==014A WAIT18
200DA0 BIT UIFR
F0FB   BEQ WAIT18
;YES.CLR T2 INT FLAG
AD08A0 LDA UT2L
```

(AIM 65 disassembly format)

```
0134 A9 LDA #00
0136 8D STA A002
0139 A9 LDA #20
013B 8D STA A00B
013E A9 LDA #05
0140 8D STA A008
0143 A9 LDA #00
0145 8D STA A009
0148 A9 LDA #20
014A 2C BIT A00D
014D F0 EB0 014A
014F AD LDA A008
```

Note that you must load the least significant bits of the timer first, since loading the most significant bits loads the counters and starts the timing operation.

Timer 1 is somewhat more complex than timer 2 because it has four operating modes (see Table 8-2). It can be used to generate a single time interval (one-shot mode) or a continuous series of intervals (free-running mode). Furthermore, each loading operation can generate an output pulse on PB7. Bits 6 and 7 of the Auxiliary Control Register determine timer 1 mode.

Bit 7 = 1 to generate output pulses on PB7, 0 to disable such pulses (in the free-running mode, PB7 is inverted each time the counter reaches zero).

Bit 6 = 1 for free-running mode, 0 for one-shot mode.

Table 8-2. Timer 1 Control

ACR7	ACR6	Mode
0	0	T1 one-shot mode — Generate a single time-out interrupt each time T1 is loaded. Output to PB7 disabled.
0	1	T1 free-running mode — Generate continuous interrupts. Output to PB7 disabled.
1	0	T1 one-shot mode — Generate a single time-out interrupt and an output pulse on PB7 each time T1 is loaded.
1	1	T1 free-running mode — Generate continuous interrupts and a square wave output on PB7.

Timer 1 occupies four memory locations (see Table 8-1). The first two addresses (A004 and A005) are used to read or write the counters. Writing the second address loads the counters, clears the timer 1 interrupt flag, and starts the timing operation. The next two addresses (A006 and A007) are used to read or write the latches without affecting the counters. This allows the generation of complex waveforms in the free-running mode. Writing the most significant bits of the latches also clears the timer 1 interrupt flag.

EXAMPLES:

1. Generate a delay of 1024 (0400 hex) clock pulses, no output to PB7.

   (AIM 65 assembly format)

```
==0152 EX13
A900   LDA #0
;SET T1 ONE-SHOT +
;PBT DISABLED MODE
8D0BA0 STA UACR
;DELAY = 0400 CLOCKS
8D04A0 STA UT1L
A904   LDA #4
;START T1
8D05A0 STA UT1CH
A940   LDA #%01000000
0
;T1 COUNTED DOWN?
==0161 WAIT19
2C0DA0 BIT UIFR
F0FB   BEQ WAIT19
;YES,CLR T1 INT FLAG
AD04A0 LDA UT1L
```

(AIM 65 disassembly format)

```
0352 A9 LDA #00
0354 8D STA A00B
0357 8D STA A004
035A A9 LDA #04
035C 8D STA A005
035F A9 LDA #40
0361 2C BIT A00D
0364 F0 EE0 0361
0366 AD LDA A004
```

This program is the same as the one shown previously for timer 2 except for the changes in the addresses and the interrupt flag bit position.

2. Generate an interrupt every 4096 (1000 hex) clock pulses and produce a pulse output on PB7 with a pulse width of 4096 clock pulses. Note that timer 1 is loaded with the desired clock count minus two. e.g. ($0FFE).

(AIM 65 assembly format)

```
;SET IRQ VECTOR
==0000
        *=$A400
==A400
0000    .WOR IRQINT
;
==A402
        *=$0370
==0370 EX20
A9FF    LDA #$FF
;MAKE SIDE B OUTPUTS
8D02A0 STA UDDRB
;SET T1 FREE RUN -
;SET ACTIVE MODE
A9C0    LDA #%11000000
0
8D0BA0 STA UACR
;ENABLE T1 IRQ INT
8D0EA0 STA UIER
;DELAY =1000 CLOCKS
;       =0FFE T1 LOAD
```

(AIM 65 assembly format) cont.

```
A9FE    LDA #$FE
8D04A0 STA UT1L
==0362
;START T1
A90F    LDA #$0F
8D05A0 STA UT1CH
;ENABLE IRQ INT
58      CLI
;CONTINUE
----------------------
IRQ INT PROCESSING

==0368 IRQINT
A940    LDA #%01000000
0
2C0DA0 BIT UIFR
;T1 CAUSE IRQ INT?
D003    BNE INTRET
;YES CLR T1 INT FLAG
AD04A0 LDA UT1L
;CONTINUE T1 INT
;PROCESSING
==0392 INTRET
40      RTI
;RETURN
```

(AIM 65 disassembly format)

```
0370 A9 LDA #FF
0372 8D STA A002
0375 A9 LDA #C0
0377 8D STA A00B
037A 8D STA A00E
037D A9 LDA #F2
037F 8D STA A004
0382 A9 LDA #0F
0384 8D STA A005
0387 58 CLI
```

(AIM 65 disassembly format) cont.

```
0338 A9 LDA #40
033A 2C BIT A00D
033D D0 BNE 0392
033F AD LDA A004
0342 40 RTI

.M:A400 A9 00 7B E0
```

This program will cause the processor to be interrupted every 4096 clock cycles. The level on PB7 will be inverted at the end of each interval (it will go low when the first interval starts). Note that T1 runs continuously; whenever the counters reach zero, they are reloaded with the values in the latches.

8.8 VIA SHIFT REGISTER

The VIA also has a shift register which can be used to convert data between serial and parallel forms. Auxiliary control register bits 2, 3, and 4 control this register as shown in Table 8-3. Loading the register starts the shifting process and an interrupt flag (bit 2 of the interrupt flag register) is set after the completion of eight shifts.

Table 8-3. Shift Register Control

ACR4	ACR3	ACR2	Mode
0	0	0	Shift Register Disabled.
0	0	1	Shift in under control of Timer 2.
0	1	0	Shift in under control of ø2.
0	1	1	Shift in under control of external clock.
1	0	0	Free-running output at rate determined by Timer 2.
1	0	1	Shift out under control of Timer 2.
1	1	0	Shift out under control of ø2.
1	1	1	Shift out under control of external clock.

EXAMPLES:

1. Shift in 8 bits under control of timer 2 and store the data into memory location $40. Subtract 2 from the desired time value (in microseconds) to determine the times 2 count value.

(AIM 65 assembly format)

```
        *=0390
==0390 EX21
A590    LDA #0
;DISABLE SR
8D8BA0 STA UACR
A594    LDA #%00000010
0
;SET SHIFT IN BY T2
8D8BA0 STA UACR
;DELAY = 0040 CLOCKS
==039A ST21
A540    LDA #40
8D88A0 STA UT2L
A500    LDA #0
8D89A0 STA UT2H
;START SHIFT IN
AD8AA0 LDA USR
==03A7 WTSH21
AD0DA0 LDA UIFR
2904    AND #%00000010
0
```

(AIM 65 assembly format) cont.

```
;SHIFT IN COMPLETE?
F0F9   BEQ WT5H21
;YES, DISABLE SR
A500   LDA #0
8D5EA0 STA UACR
;GET AND STORE DATA
A005A0 LDA USR
8540   STA $40
```

(AIM 65 disassembly format)

```
0190 A5 LDA #00
0192 8D STA A00B
0195 A5 LDA #04
0197 8D STA A00B
019A A9 LDA #26
019C 8D STA A008
019F A9 LDA #00
01A1 8D STA A009
01A4 AD LDA A00A
01A7 AD LDA A00D
01AA 29 AND #04
01AC F0 BEQ 01A7
01AE A9 LDA #00
01B0 8D STA A00B
01B3 AD LDA A00A
01E6 85 STA 40
```

2. Shift in 8 bits under control of phase 2 (ø2) and store the data into memory location $40.

(AIM 65 assembly format)

```
==0159 EX22
A500   LDA #0
;DISABLE ACR
0162A0 STA UACR
A502   LDA #%00000100
0
;SET SHIFT IN BY
;PHASE 2 MODE
0164A0 STA UACR
;START SHIFT IN
A506A0 LDA USR
;DELAY 18 CYCLES
A204   LDX #4
==016C WTSH22
CA     DEX
;SHIFT IN COMPLETE?
D0FC   BNE WTSH22
;YES, DISABLE SR
0170A0 STA UACR
;GET AND STORE DATA
AD41A0 LDA USR
8540   STA $40
```

(AIM 65 disassembly format)

```
0180 A9 LDA #00
0182 8D STA A00B
0185 A9 LDA #08
0187 8D STA A00B
018A AD LDA A00A
018D A2 LDX #04
018F CA DEX
0190 D0 BNE 018F
0192 8D STA A00B
0195 AD LDA A00A
0198 85 STA 40
```

3. Shift in 8 bits under control of an external clock and store the data into memory location $40.

(AIM 65 assembly format)

```
==0302 EX23
A300   LDA #0
;DISABLE SR
0305A0 STA UACR
A302   LDA #%00000110
0
;SET SHIFT IN BY
;EXT CLOCK MODE
0307A0 STA UACR
==030C WTSH23
A309A0 LDA UIFR
2904   AND #%00000010
0
;SHIFT IN COMPLETE?
F0F9   BEQ WTSH23
;YES, GET AND STORE
;DATA
A30CA0 LDA USR
2540   STA $40
```

(AIM 65 disassembly format)

```
0302 A9 LDA #00
0304 8D STA A00B
0307 A9 LDA #0C
0309 8D STA A00B
030C AD LDA A00D
030F 29 AND #04
0311 F0 BEQ 030C
0313 AD LDA A00A
0316 85 STA 40
```

4. Continually shift out 8 bits from memory location $40 at timer 2 rate.

(AIM 65 assembly format)

```
==03E8 EX24
A900   LDA #0
;DISABLE SR
8D0BA0 STA UACR
A910   LDA #%00010000
0
;SET SHIFT OUT
;CONTINUOUSLY BY T2
8D0BA0 STA UACR
;DELAY =0060
A960   LDA #$60
8D09A0 STA UT2L
A900   LDA #0
==03F9
8D09A0 STA UT2H
;LOAD SR WITH DATA
;AND START SHIFT OUT
A940   LDA #$40
8D0AA0 STA USR
```

(AIM 65 disassembly format)

```
03D2  A9 LDA #00
03D4  8D STA A00B
03D7  A9 LDA #0C
03D9  8D STA A00B
03DC  AC LDA A00A
03DF  AD LDA A00D
03E2  29 AND #04
03E4  F0 EB 03DF
03E6  AD LDA A00A
03E9  85 STA 40
```

5. Shift out 8 bits from memory location $40 under control of timer 2. Subtract 2 from the desired time value (in microseconds) to determine the times 2 count value.

(AIM 65 assembly format)

```
==0401 EX25
A900   LDA #0
;DISABLE SR
8D0BA0 STA UACR
A914   LDA #%00001010
0
;SET SHIFT OUT BY
;T2 MODE
8D0BA0 STA UACR
;DELAY =0020
A920   LDA #$20
8D08A0 STA UT2L
A900   LDA #0
==0412
8D09A0 STA UT2H
;LOAD DATA INTO SR
;AND START SHIFT OUT
A540   LDA $40
8D0AA0 STA USR
```

(AIM 65 disassembly format)

```
0401 A9 LDA #00
0403 8D STA A00B
0406 A9 LDA #14
0408 8D STA A00B
040B A9 LDA #20
040D 8D STA A008
0410 A9 LDA #00
0412 8D STA A009
0415 A5 LDA 40
0417 8D STA A00A
```

6. Shift out 8 bits from memory location $40 under control of phase 2 (ø2).

(AIM 65 assembly format)

```
==041A EX26
A900   LDA #0
;DISABLE SR
8D0BA0 STA UACR
A918   LDA #%00011000
0
;SET SHIFT OUT BY
;PHASE 2 CLOCK MODE
8D0BA0 STA UACR
;LOAD SR WITH DATA
;AND START SHIFT OUT
A540   LDA $40
8D0AA0 STA USR
```

(AIM 65 disassembly format)

```
041A A9 LDA #00
041C 8D STA A00B
041F A9 LDA #18
0421 8D STA A00B
0424 A5 LDA 40
0426 8D STA A05A
```

7. Shift out 8 bits from memory location $40 under control of an external clock with IRQ interrupt handling.

(AIM 65 assembly format)

```
 SET IRQ VECTOR
==0000
        *=$A400
==A400
4504     WOR IRQINT
;
==A402
        *=$0420
==0420 EA27
A000    LDA #0
;DISABLE SR
8D3EA0  STA UACR
A911    LDA #%00001110
0
;SET SHIFT OUT BY
;EXT CLOCK MODE
8D3EA0  STA UACR
A984    LDA #%10000010
0
;ENABLE SR IRQ INT
8D3EA0  STA UIER
;LOAD SR WITH DATA
;AND START SHIFT OUT
A540    LDA $40
==0441
8D3FA0  STA USR
;ENABLE IRQ INT
58      CLI
;CONTINUE
```

(AIM 65 assembly format) cont.

```
    IRQ INT PROCESSING

    ==0445 IRQINT
    A304   LDA #%00000010
    0
    2C03A0 BIT UIFR
    ;SR CAUSE IRQ INT?
    D003   BNE INTRET
    ;YES.CLR SR INT FLAG
    AD0AA0 LDA USR
    ;CONTINUE SR INT
    ;PROCESSING
    ==044F INTRET
    40     RTI
    ;RETURN
```

(AIM 65 disassembly format)

```
0430 A9 LDA #00
0432 8D STA A00B
0435 A9 LDA #1C
0437 8D STA A00B
043A A9 LDA #84
043C 8D STA A00E
043F A5 LDA 40
0441 8D STA A00A
0444 58 CLI
0445 A9 LDA #04
0447 2C BIT A00D
044A D0 BNE 044F
044C AD LDA A00A
044F 40 RTI

<M:=A400 45 04 7B E0
```

8.9 INTERFACING REFERENCES

Microprocessor input/output and interfacing are discussed in:

Lesea, A. and R. Zaks, Microprocessor Interfacing Techniques, Sybex, Berkeley, Ca., 1977

Peatman, J., Microcomputer-based Design, McGraw-Hill, New York, 1977

Wakerly, J.F., "Microprocessor Input/Output Architecture", Computer, February, 1977, pp. 26-33

8.9 INTERFACING REFERENCES

Microprocessor input/output and interfacing are discussed in:

Lesea, A. and R. Zaks, Microprocessor Interfacing Techniques, Sybex, Berkeley, Ca., 1977

Peatman, J., Microcomputer-based Design, McGraw-Hill, New York, 1977

Wakerly, J.F., "Microprocessor Input/Output Architecture", Computer, February, 1977, pp. 26-33

SECTION 9

INTERFACING WITH AUDIO CASSETTE RECORDERS AND TELETYPE

AIM 65 provides inerfaces to two types of external peripherals: audio cassette recorder and teletype. These peripherals allow permanent storage of source and object programs as well as program data of general text. There are two major reasons why permanent storage is desirable:

- The AIM 65 RAM is volatile memory--its memory contents are altered to an unpredictable state when RAM power is removed.

- A permanent storage medium is desired to save source programs, object programs, and data currently in RAM, to make room for other programs and data.

9.1 INTERFACING WITH AUDIO CASSETTE RECORDERS

AIM 65 can interface and operate with one or two audio cassette recorders. Information recorded on cassette may be read by AIM 65 as many times as desired. In addition, data can be transferred between recorders.

The AIM 65 audio cassette interface hardware and software is designed to provide flexible and useful AIM 65 operation. The capability to record and read both source and object data allows program and data development and usage at either or both levels. A binary data recording technique for object data allows rapid and efficient program dumping and loading.

A blocked, multi-record, object code format enables recording of program or data segments from different portions of memory. Since the starting address is provided with each record, program segments from different portions of memory can be recorded on the same file.

The block format, in conjunction with I/O data buffering allows processing to be performed between block read operations. This supports such functions as assembling from cassette and reading new data into a partially filled Editor Text Buffer.

Only a single recorder is needed to record or to read source or object data. Two recorders are required to assemble input source code from one cassette while directing output object code to the other cassette.

Remote control capability allows the recorders to be set up for reading or recording, then letting AIM 65 intitiate the recorder operation at the proper time during command execution. Note that remote control is a convenient feature, but not required for recording data from Monitor dump, load, or verify commands, Editor List or initial Read commands. Remote control is required, however, in order to assemble from cassette or to read source data into a partially loaded Editor Text Buffer.

A dual remote control feature along with the blocked format structure allows cassette-to-cassette assembly. This feature maximizes use of on-board RAM during both assembly and subsequent loading of object code.

The audio tape interface allows low cost audio cassette recorders to be used. It is recommended, however, that the highest quality recorders and cassette tapes be used to obtain maximum performance and reliability.

9.1.1 Recorder Requirements

The audio cassette recorders used should be equipped with the following features:

- An earphone (EAR) jack. The AIM 65 audio input line will use it to read cassette data into AIM 65.

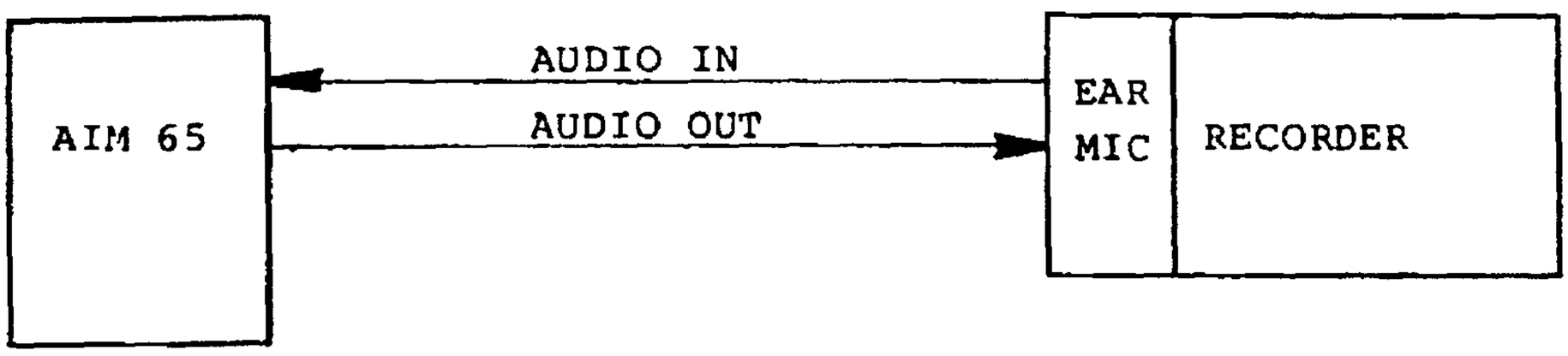
- A microphone (MIC) jack. The AIM 65 audio output line will use it to record cassette data from AIM 65.

- A remote (REM) jack. An AIM 65 output recorder control line will use it to turn the recorder on or off automatically or by user command. This line is not required for most operations.

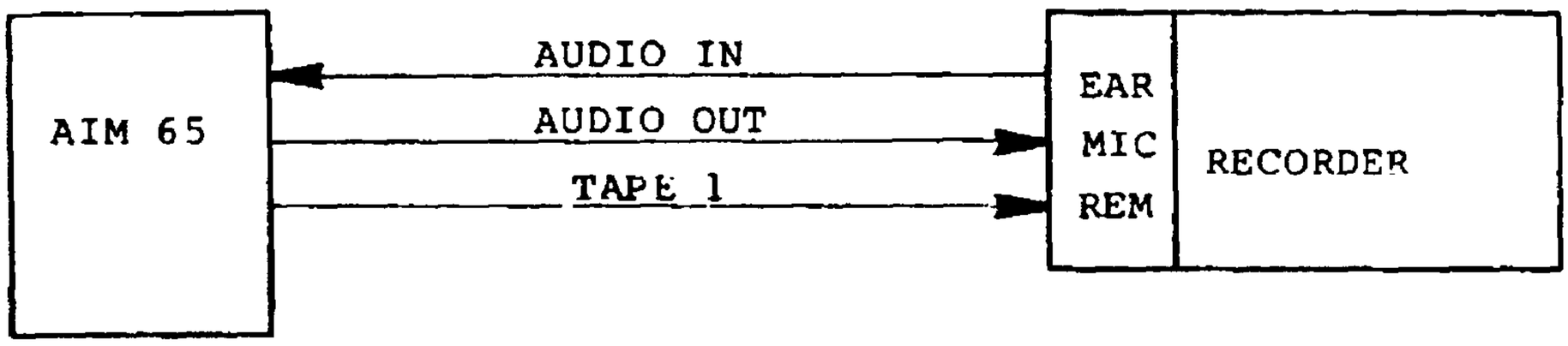
A tape counter, while not required for operation, provides a convenient reference for location programs written on cassette.

9.1.2 Cassette Recorder Installation

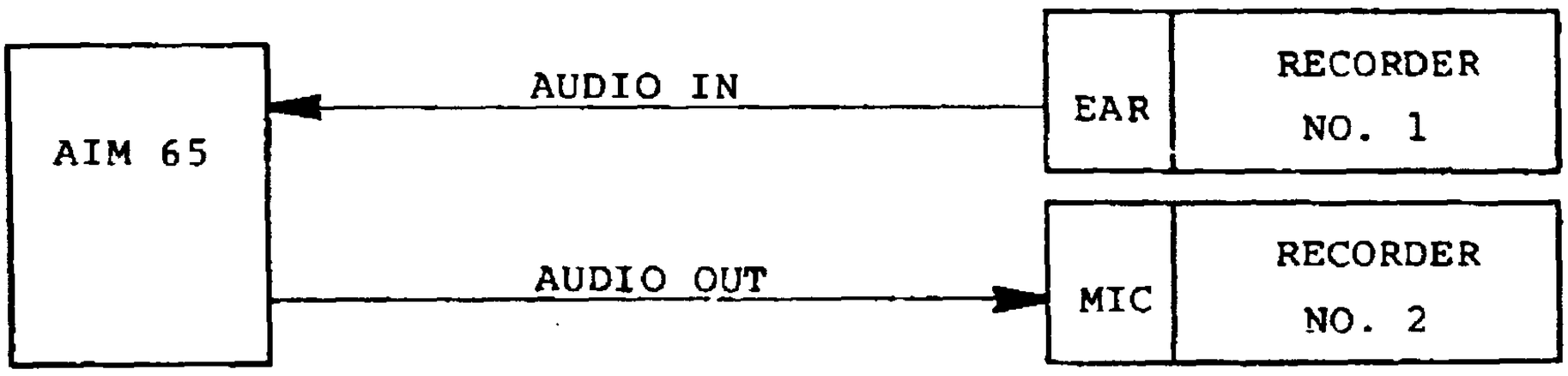
AIM 65 contains a built-in interface for one or two audio cassette recorders, with or without remote control. Figure 9-1 illustrates the connections required with each of the four possible combinations.



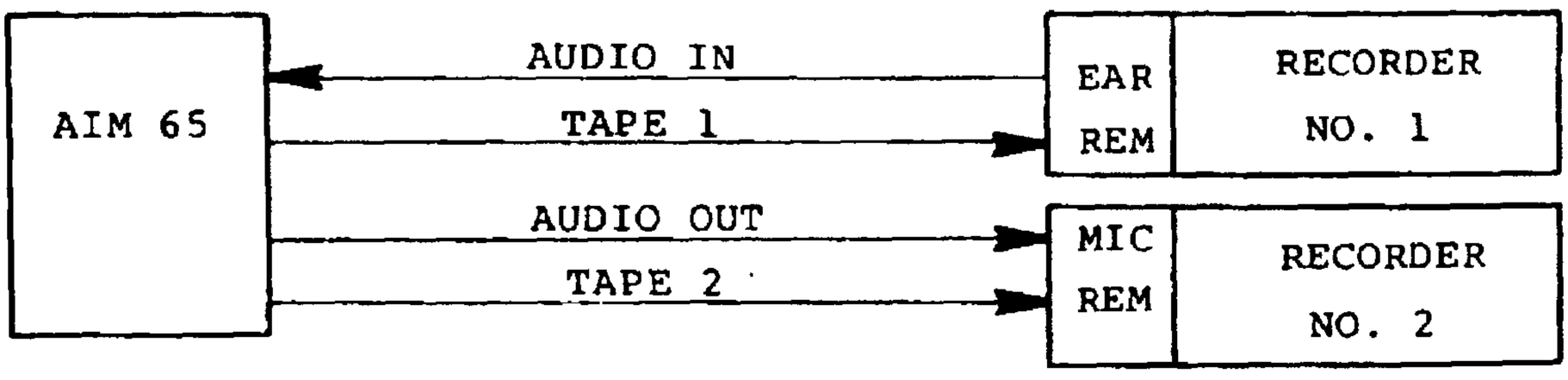
1. ONE RECORDER WITH NO REMOTE CONTROL LINE



2. ONE RECORDER WITH ONE REMOTE CONTROL LINE



3. TWO RECORDERS WITH NO REMOTE CONTROL LINES



4. TWO RECORDERS WITH TWO REMOTE CONTROL LINES

Figure 9-1. Typical Audio Cassette Recorder Hookups

INSTALLING RECORDERS WITHOUT REMOTE CONTROL

Recorders lacking remote control are connected to AIM 65 with just two lines, AUDIO IN and AUDIO OUT. If only one recorder is being used, AUDIO IN will plug into the recorder's EAR jack and AUDIO OUT will plug into the recorder's MIC jack. If two recorders are being used, AUDIO IN will plug into the EAR jack of the input recorder and AUDIO OUT will plug into the MIC jack of the output recorder.

Figure 9-2 shows how these lines should be wired to AIM 65 Application Connector J1. In making the hookup, the leads should be kept as short as possible, and should be positioned far away from such sources of electrical interference as AC line cords and transformers. Note also that the AIM 65 ground connection, J1-1, should be used rather than an external ground.

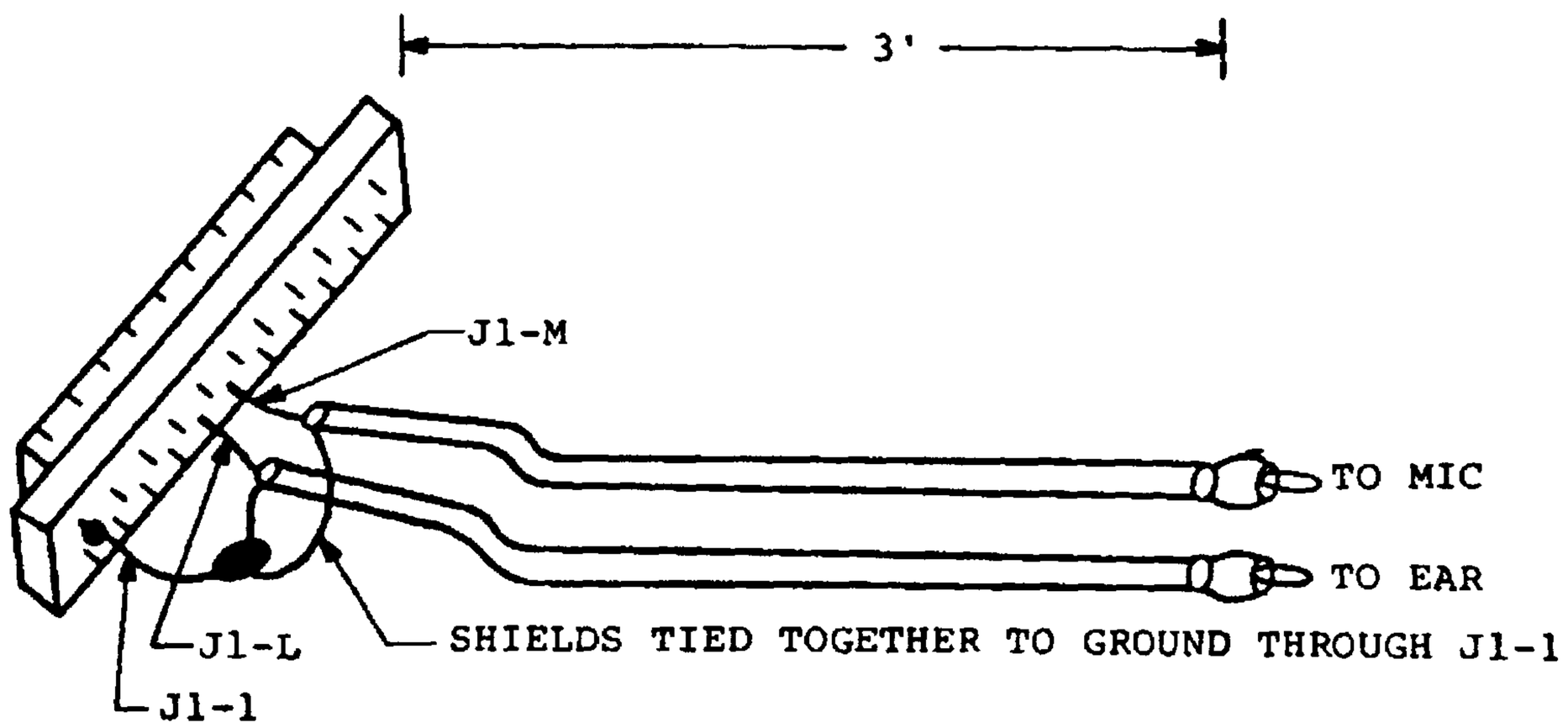
INSTALLING RECORDERS WITH REMOTE CONTROL

For remote control applications, you must install the AUDIO IN and AUDIO OUT lines using the procedure just described, then hook up the remote line(s).

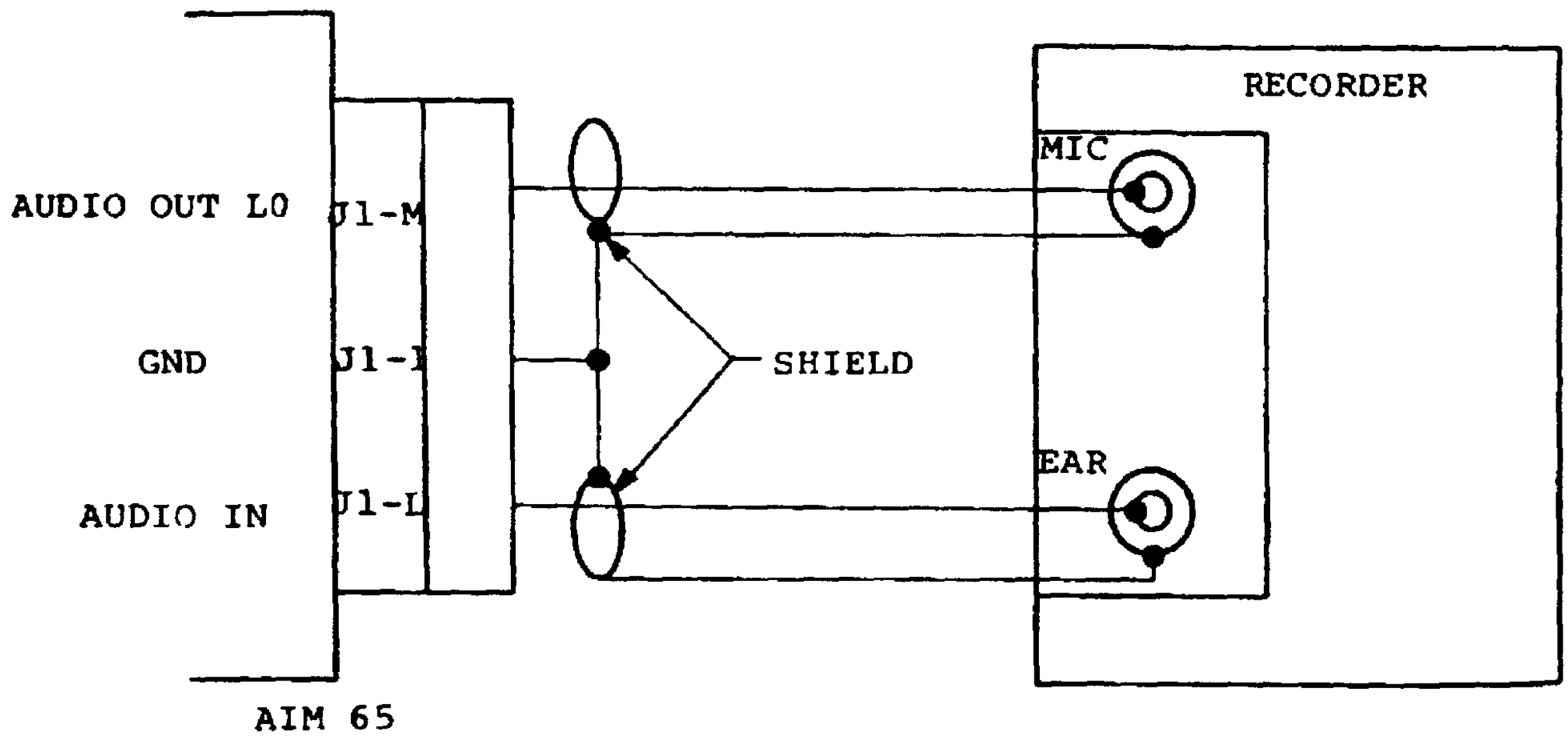
AIM 65's remote control circuitry supports four different types of recorder remote control circuits:

- Type I -- Positive Voltage, Motor Return to Phono Jack Shield Connection (PRS)

- Type II -- Positive Voltage, Motor Return to Phono Jack Center Connection (PRC)



PICTORIAL OF SUGGESTED INTERFACE WIRING

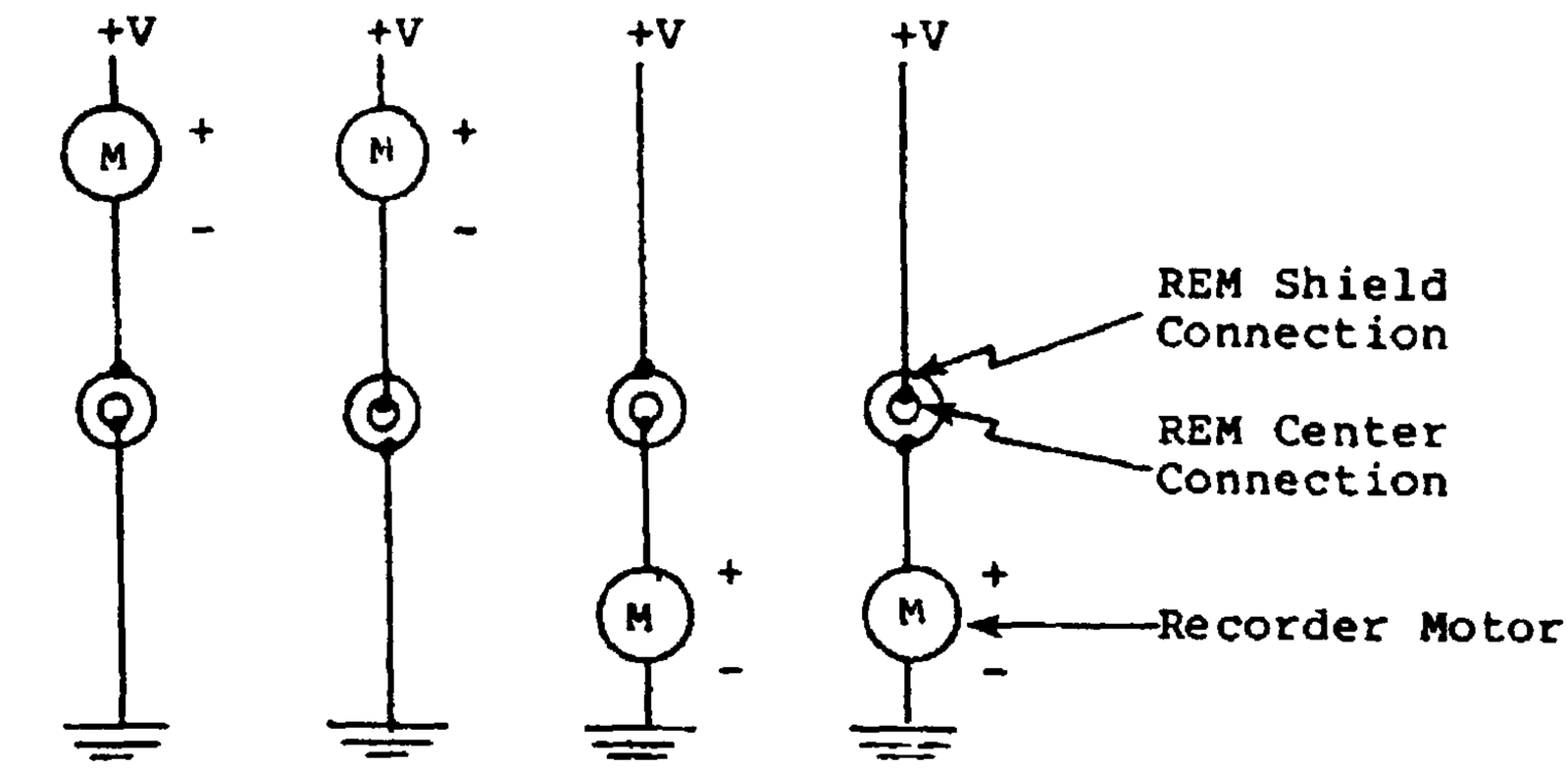


SCHEMATIC

Figure 9-2. Cassette Recorder Audio Line Connection

Type III -- Positive Voltage, Motor Voltage to Phono Jack Shield Connection (PVS)

Type IV -- Positive Voltage, Motor Voltage to Phono Jack Center Connection (PVC)



Type I (PRS) Type II (PRC) Type III (PVS) Type IV (PVC)

If you are not sure which type of recorder you have, you will find out in the course of this installation procedure:

1. Construct a recorder-to-AIM 65 remote control cable, by attaching a recorder-compatible phono plug to a single conductor shielded cable. To do this, connect the shield

cable inner conductor to the phono plug center connection, and the cable shield to the phono plug shield connection. Leave the AIM 65 end of the cable disconnected and not shorted together.

2. With the remote control cable disconnected from the recorder, turn the recorder on in the Play mode. The recorder motor should run.

3. Plug the remote control cable into the recorder's REM phono jack. The motor should stop. If the motor continues to run, a short circuit exists between the REM jack's center and shield connections. If this problem occurs, it must be corrected before proceeding; as an initial step, check your cable conductor connections to the phono plug.

4. Connect the remote control cable conductors together at the AIM 65 end. The recorder motor should run. If it does not, an open circuit exists between the phono jack center and shield connections; this problem must be corrected before proceeding.

5. Connect the phono plug to the recorder's REM phono jack. With the recorder motor running, use a voltmeter to make the following measurement: Touch the voltmeter's "+" lead to the remote control cable center conductor (at the AIM 65 end) and the "-" lead to a ground point, such as the recorder chassis, or the shield connection on either the MIC or ear jack. Record the measured voltage here: ____ Vdc.

   a. If the voltage measures 0 Vdc, you have a Type I or Type II recorder. Proceed to Step 6.

b. If the voltage measures +6 to +8 Vdc, you have a Type III or Type IV recorder. Proceed to Step 6.

c. If the voltage measures -6 to -8 Vdc, the remote control circuit in your recorder is not compatible with AIM 65, and you are restricted to using it with only the audio lines connected, unless an interface adapter or relays are employed.

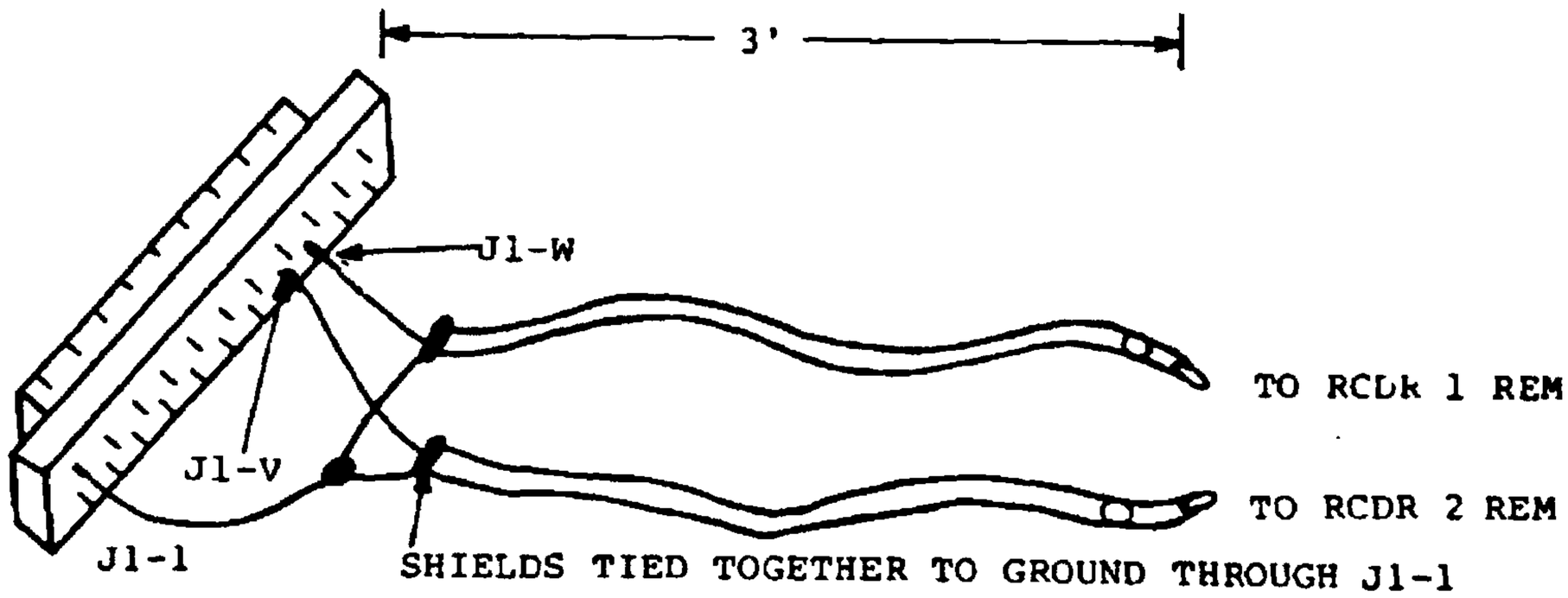
6. Disconnect the remote control cable center conductor from the shield conductor at the AIM 65 end. The recorder motor will stop. With the recorder in the Play mode, measure the voltage on the remote control cable center conductor. Record the measured voltage here: ____ Vdc. The recorder remote control circuit type can be determined by referring to this table:

Step 5 (Vdc)	0		+6 to +8	
Step 6 (Vdc)	0	+6 to +8	0	+6 to +8
Type	I	II	III	IV

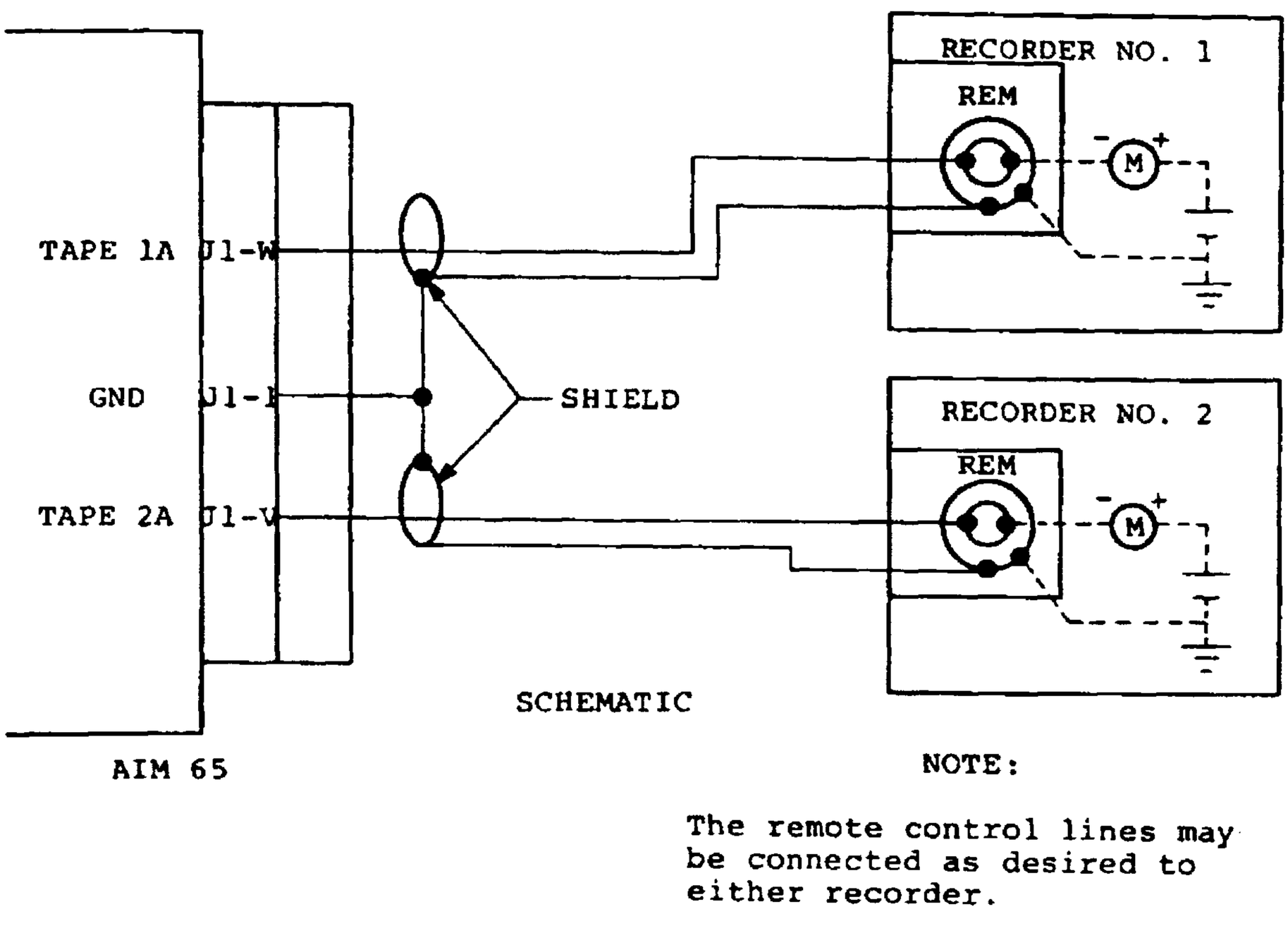
7. Connect the remote control cable to AIM 65 Application Connector J1 per Table 9-1. Figures 9-3 and 9-4 show hookups to recorders with Type II (PRC) and Type IV (PVC) remote control circuitry, respectively.

Table 9-1 Recorder Remote Control Connection

RECORDER TYPE	PHONO JACK CONNECTION TO AIM 65			
	RECORDER 1		RECORDER 2	
	SHIELD	CENTER	SHIELD	CENTER
Type I (PRS)	J1-W	J1-1	J1-V	J1-1
Type II (PRC)	J1-1	J1-W	J1-1	J1-V
Type III (PVS)	J1-F	J1-E	J1-J	J1-H
Type IV (PVC)	J1-E	J1-F	J1-H	J1-J



SUGGESTED INTERFACE WIRING

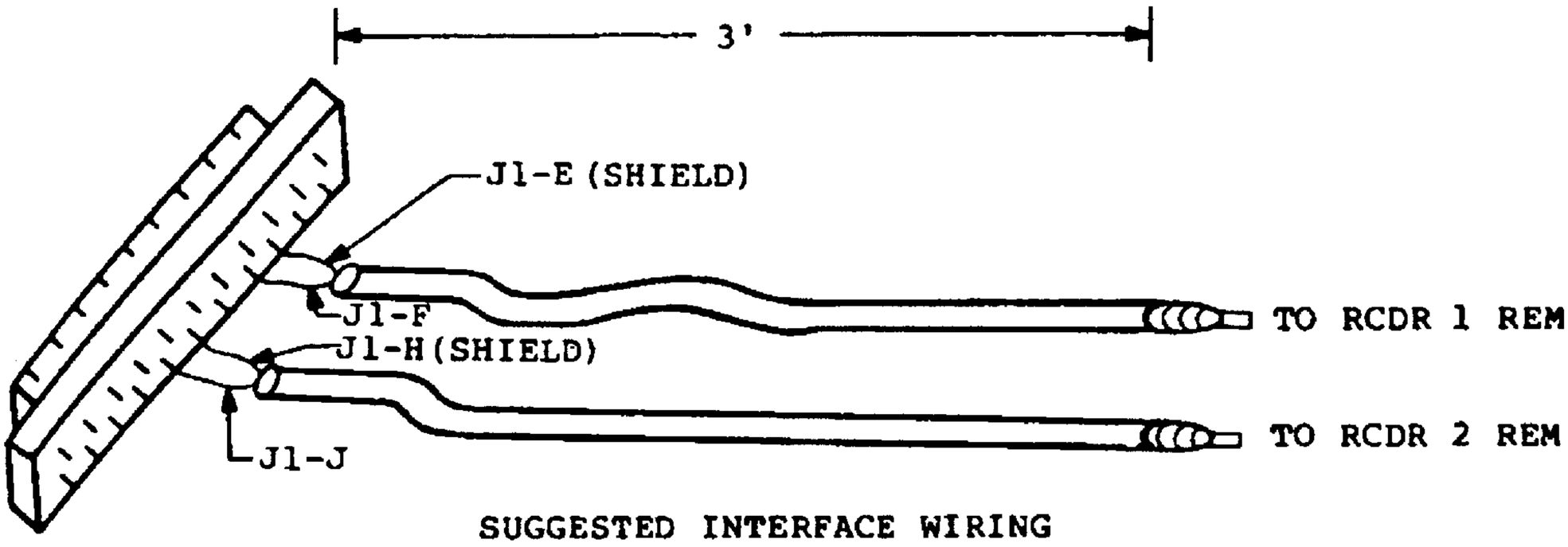


SCHEMATIC

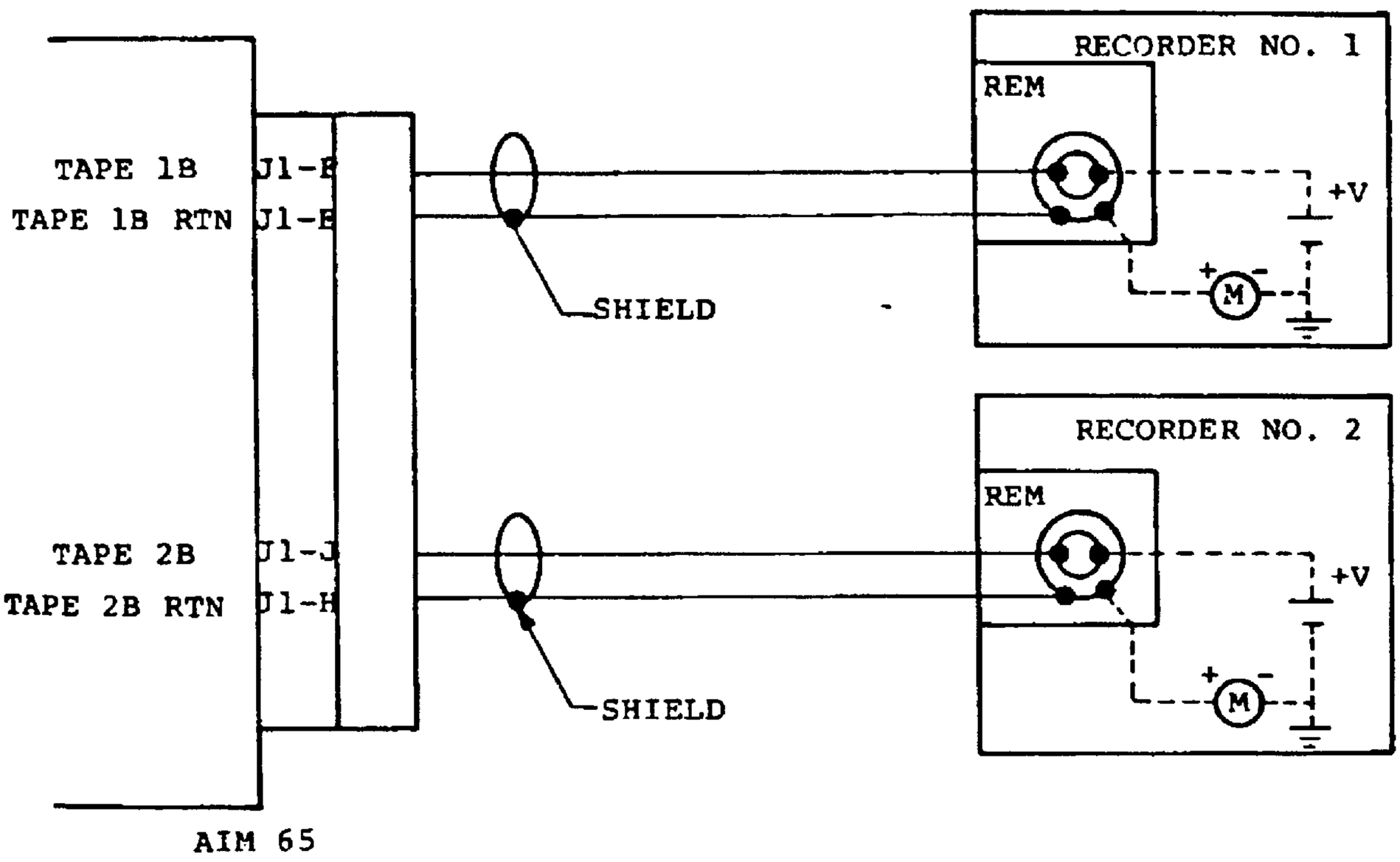
NOTE:

The remote control lines may be connected as desired to either recorder.

Figure 9-3. Audio Cassette Recorder Remote Control Connections - Type II (PRC)



SUGGESTED INTERFACE WIRING



NOTE:

The remote control lines may be connected as desired to either recorder.

SCHEMATIC

Figure 9-4. Audio Cassette Recorder Remote Control Connections - Type IV (PVC)

9.1.3 Cassette Recorder Operation

To save time and effort when using audio cassette recorders:

- When installing a cassette into a recorder, always rewind the cassette until it stops, then reset the recorder counter.

- When recording the first file on the tape, BE SURE THE CASSETTE HAS ADVANCED BEYOND ANY NON-RECORDABLE LEADER ON THE TAPE. If you cannot see the physical start of the magnetic portion of the tape through a transparent cassette housing, allow at least five counts on the recorder counter, or about 15 seconds.

- A large gap (10 counts, or three inches of tape) should be allowed between recorded files. This permits the tape to be manually positioned between files before initiating a read operation.

9.1.4 AIM 65 Operation Verification Procedure

After installing the audio cassette recorders, run a SYN pattern record and read test to verify correct recorder interface connection and operation. The short program described below will run such a test.

This test writes a continuous stream of SYN (ASCII 16) characters onto a cassette tape, then has AIM 65 attempt to read the tape. If the SYN characters are read by AIM 65, a Y is displayed, otherwise an N is displayed.

SYN TEST PATTERN PROGRAM

Enter the following program into AIM 65 RAM using the instructions mnemonic entry command (I).

SYN Write Program

```
0000     *=0300
0300 20 JSR FD1D
0303 20 JSR FD4A
0306 4C JMP 0301
```

SYN Read Program

```
0309     *=0310
0310 A2 LDX #00
0312 A9 LDA #0E
0314 20 JSR EF7E
0317 20 JSR EDBA
031A A2 LDX #00
031C A9 LDA #16
031E 20 JSR EF7B
0321 20 JSR EE19
0324 C9 CMP #16
0326 F0 EB  0311
0328 D0 BNE 0310
```

Write SYN Characters on Tape

1. Connect data lines per Figure 9-1

2. Install a blank, or scratch, tape into the recorder. Advance the tape past the leader.

3. Ensure the TSPEED value in $A408 equals $C7 for AIM 65 format, or equals $5A or $5B for KIM-1 format.

4. Execute the program to record the SYN characters on the tape as follows:

```
<*>=0300
<G>/
```

5. Start the tape recorder in the Record Mode.

6. Wait for a few minutes while a long string of continuous SYN characters are recorded.

7. Return control to Monitor by pressing the AIM 65 RESET button.

8. Turn off the recorder.

Read Recorder SYN Characters

1. Rewind tape to the start of the SYN file.

2. Execute the SYN read program.

```
<*>=0310
<G>/
```

3. Increase the recorder volume control to the highest level.

4. Increase the tone control (if available) to the highest treble level.

5. Start the tape recorder in the Play Mode.

6. The display will show N until the SYN characters are read, at which time it should show a steady Y. Decrease the volume control until the Y and N alternate on the display.

If a steady Y display cannot be obtained at or near maximum volume, check for one of the following:

- poor audio line connection
- recorder batteries are low
- recorder is malfunctioning
- AIM 65 VR1 is out of adjustment (see Circuit Adjustment)

7. Stop the recorder. Set the volume control to the maximum loudness level. A steady Y or N will remain on the display.

8. Return control to the Monitor by pressing the RESET button.

9. Turn off the tape recorder.

CIRCUIT ADJUSTMENT FOR CASSETTE INTERFACE

VR1 on the audio cassette interface (see Figure 7-12) has been factory-adjusted for proper operation. If the setting is accidentally altered, it can be re-adjusted by

the following two procedures. If you have a voltmeter or oscilloscope, perform both the coarse adjustment and the fine adjustment. In the absence of this equipment, perform only the fine adjustment.

The coarse adjustment procedure is:

1. Turn on AIM 65 power.

2. Connect the + lead of the voltmeter or oscilloscope to Z8-3 and the - lead to Z8-1.

3. Adjust VR1 for 2.5 $\pm$ 0.1 VDC.

The fine adjustment procedure is:

1. Record a SYN test pattern tape per Section 9.1.4.

2. Read the test pattern tape per Section 9.1.4.

3. After the volume control is adjusted until Y and N are alternating, adjust VR1 until a steady Y is displayed.

4. Decrease the volume control until Y and N again alternate on the display.

5. Adjust VR1 until a steady Y is again displayed.

6. Repeat Steps 4 and 5 until no further adjustment is required to VR1.

9.1.5 Recording On A Cassette

Text or object data may be recorded on audio cassette in AIM 65 format for any AIM 65 command allowing output device code =T. The recording format is shown in Appendix F. These commands and the type of recorded data are:

COMMAND	DATA TYPE	DATA FORM
Monitor Dump D	Object	Hexadecimal
Editor List L	Text	ASCII
Assembler Listing Output	Assembly Listing	ASCII
Assembler Object Code Output	Object	Hexadecimal

Object code may be recorded in KIM-1 format for the Monitor Dump command by specifying output device code =K. Before dumping in KIM-1 format, change user-alterable parameter TSPEED at location $A408 to $5A (for one times KIM-1 speed) or to $5B (for three times KIM-1 speed) before recording. TSPEED is defined in Section 7.6; KIM-1 format is described in Appendix G.

The record procedure is:

1. Install the cassette and manually position the tape to where the recording is to start. Be sure to initialize the counter to the start of the tape.

Allow about 10 counts (or three inches of tape) between the last recorded file and the new file. Enter the tape count on a tape dictionary for future reference. Figure 9-5 shows a typical form, with an example.

NOTE

If a remote control line is installed, the recorder will not manually operate unless the control line is ON.

Tape ID T001

File Name	SRC/ OBJ	Tape Count		Address		Program Name	Notes
		High	Low	Low	High		
TMR1S	S	010	032	---	---	Timer 1	
TMR1L	L	040	047	0200	03D0	Timer 1	
TST3S	S	060	071	---	---	Test 3	
TST3L	L	080	084	0200	02F2	Test 3	

Figure 9-5. AIM 65 Audio Cassette Dictionary Form

CAUTION

In installations using remote control, some recorders may require a larger audio tape GAP (location $A409) value than is provided by the default value, $08. If you encounter errors in reading from your recorder, alter $A409 to $40.

2. Set up the desired AIM 65 command. AIM 65 will prompt for the output device code with:

```
OUT=
```

Type K if the output from the Monitor Dump Command (D) is to be recorded in KIM-1 format; otherwise type T.

If T was entered, AIM 65 will display T and ask for the file name (F=). Enter the file name, up to five alphanumeric or special characters. If the file name is less than five digits, end the input with RETURN or SPACE. If five digits are entered, the file name entry will automatically end. AIM 65 will display the file name and ask for the recorder number. If NAME1 has been entered, AIM 65 will respond with:

```
OUT=T  F=NAME1  T=
```

If K was entered, type a two-digit hexadecimal file number in the range 01 to FE in response to the F= prompt.

3. Type the recorder number 1 or 2. RETURN will default to 1. If the entered number is incorrect, type ESC to escape back to the Monitor and reinitialize the command.

4. Put the recorder into the Record Mode. If the remote control is not being used, or if it is being used and is in the ON state, the tape will start recording. In this case, go directly to Step 5. If the remote control is hooked up and is in the OFF state, the recording will not start until the next step is performed.

5. Type RETURN or SPACE to initiate record command execution. If the remote control was hooked up in the OFF state, the ON state will automatically be commanded. The block count will be displayed as the recording progresses (unless the data is being dumped in KIM-1 format). For example:

```
OUT=T  F=NAME1  T=1  03
```

indicates block 03 is being recorded in the file NAME1 on recorder 1.

Completion of the dump is signaled by display of the Monitor prompt.

NOTE

The output process may be terminated at any time by holding down ESC until the Monitor prompt is displayed. This type of termination will cause the last end of file record to be omitted from the tape. Subsequent attempts to read the partially recorded file will read the data properly up to the termination point but will not properly close out the read process.

6. The output will end in accordance with the specific command termination. If the termination is normal, the control will be returned to command Monitor function, i.e., Monitor, Editor or Assembler. The tape must be stopped manually or by typing 1 or 2 to toggle the proper remote control command to OFF.

If a Monitor Dump command is being performed, the Monitor will automatically stop the tape when MORE? prompt is displayed if the remote control is connected. The tape will likewise be started when the response answers to this prompt sequence are complete. If a remote control line is not connected, the tape will continue to record during the prompt display and response entry. In this case the tape can be manually stopped, then restarted, prior to answering the MORE? prompt, or it can be allowed to run. If allowed to run, a punctual response to the MORE? prompt will minimize the delay during subsequent reading.

NOTE

Data is recorded by filling a tape buffer in memory with 80 bytes of data. When the buffer is full, the data is output to the recorder. When the MORE? prompt is displayed, up to 79 bytes of unrecorded object data may reside in the tape buffer. If the dump command is terminated with an ESC at this point rather than N, any unrecorded data in the tape buffer will remain unrecorded. Not only will unrecorded data be lost but the last record will not be recorded which will cause subsequent improper read termination.

Also, a short file -- of less than 80 bytes including the last record (see Appendix F) -- will not be recorded if the dump is terminated with an ESC or RESET instead of typing N to the MORE? prompt.

7. Upon completion of recording, switch the recorder out of the record mode, note the tape counter value on the tape directory, and advance the tape about 5 counts on the recorder counter for subsequent recording.

9.1.6 Reading From a Cassette

Text or object data may be read from audio cassette tape recorded in the AIM 65 format using any AIM 65 command allowing input device =T. These commands are:

COMMAND	DATA TYPE	DATA FORM
Monitor Load Command L	Object	Hexadecimal
Editor Read Command R	Text	ASCII
Assembler Source Code Input	Text	ASCII
Monitor Verify Tape Command 3	Text Object	ASCII Hexadecimal

Object code recorded in KIM-1 format may be read using the Monitor Load command by specifying input device code=K. However, before reading KIM-1 formatted object code, change user-alterable parameter TSPEED at location $A408 to either $5A (for one times KIM-1 speed) or $5B (for three times KIM-1 speed).

The read procedure is:

1. Install the cassette and manually position the tape to about five counts or a couple of inches of tape before the start of the desired file. Remember to initialize the tape counter to the start of the cassette tape if not done previously.

2. Set up the desired AIM 65 command. AIM 65 will prompt for the input device code with:

```
IN=
```

Type K if the input is from a cassette recorded in KIM-1 format; otherwise, type T.

If T was entered, AIM 65 will display T and ask for the file name (F=). Enter the file name under which the file was recorded, up to five alphanumeric or special characters. An input error before entry of the fifth digit may be corrected by typing DEL and retyping the correct character. If the file name is less than five digits, end the input with RETURN or SPACE. If five digits are entered, the file name entry will automatically end.

AIM 65 will display the file name and ask for the recorder number. If NAME1 has been entered, AIM 65 will respond with:

```
IN=T  F=NAME1  T=
```

If K was entered, type the two-digit hexadecimal file number in response to F=.

3. Type the recorder number, 1 or 2. RETURN will default to 1. If the entered number is incorrect, type ESC to escape back to the Monitor and re-initialize the command.

4. If remote control is not used, go to Step 5.

   If remote control is installed, set the proper recorder control line to OFF using the Monitor 1 or 2 command. Switch the recorder into the Read Mode. The tape should not move; if it does, the remote control line is probably hooked up incorrectly or it is in the ON state. If this occurs, stop the tape. If necessary, rewind the tape to position the start of the file before the read head. Either check the remote control and repeat this step or continue on under manual control.

5. Type RETURN or SPACE to initiate read command execution. If 1 is entered, AIM will display:

```
IN=T  F=NAME1  T=1
```

6. If manual control is used, place the recorder in the Read mode. The tape will then start reading. If remote control is used, the Monitor will turn the remote control line ON to enable tape reading.

AIM 65 will search for the entered file name. Upon locating the first readable tape file, the file name on tape will be compared to the entered file name. If the file names are not identical, AIM 65 will display the search message, the file name read from tape, and (unless KIM-1 formatted data is being read) the recorded block count (see Appendix F.1) as the file passes. If file name PROG1 was read, AIM 65 will display the search message.

```
SRCH   F=PROG1   BLK=XX
                 Where XX=the block count
```

If the displayed block count is the last block count on the file, the displayed block count will not change until a new file is located.

NOTE:

If the tape is started within a file, the block count will be displayed as part of the command message until the first file is read:

```
IN=T   F=NAME1   T=1XX
```

Upon reading the entered file name from the tape, AIM 65 will display the load message and block count (except for KIM-1 format) as the data is read. For example, when file NAME1 is located, AIM 65 will display:

```
LOAD F=NAME1   BLK=XX
```

Completion of the read is indicated by return to the command that existed before the tape read, e.g., Monitor or Editor Command states.

ERROR MESSAGES

If any error is detected during the reading of a file, either during search or load, an error message will be generated, the reading terminated, and the Monitor reentered.

9.2 INTERFACING WITH TELETYPE

AIM 65 provides an interface to a teletype (TTY). A TTY usually has a paper tape punch and reader. Using the punched paper, both source and object programs may be stored on and retrieved from paper tape. The Monitor provides the capability to dump and load object code while the Editor allows source code or any other text in ASCII format to be listed and read.

A TTY also provides an alternative keyboard and hard copy facility. Since AIM 65 includes a full-size keyboard, the operating procedure with the TTY keyboard is almost identical to AIM 65. This section describes the differences between AIM 65 keyboard and TTY keyboard operation.

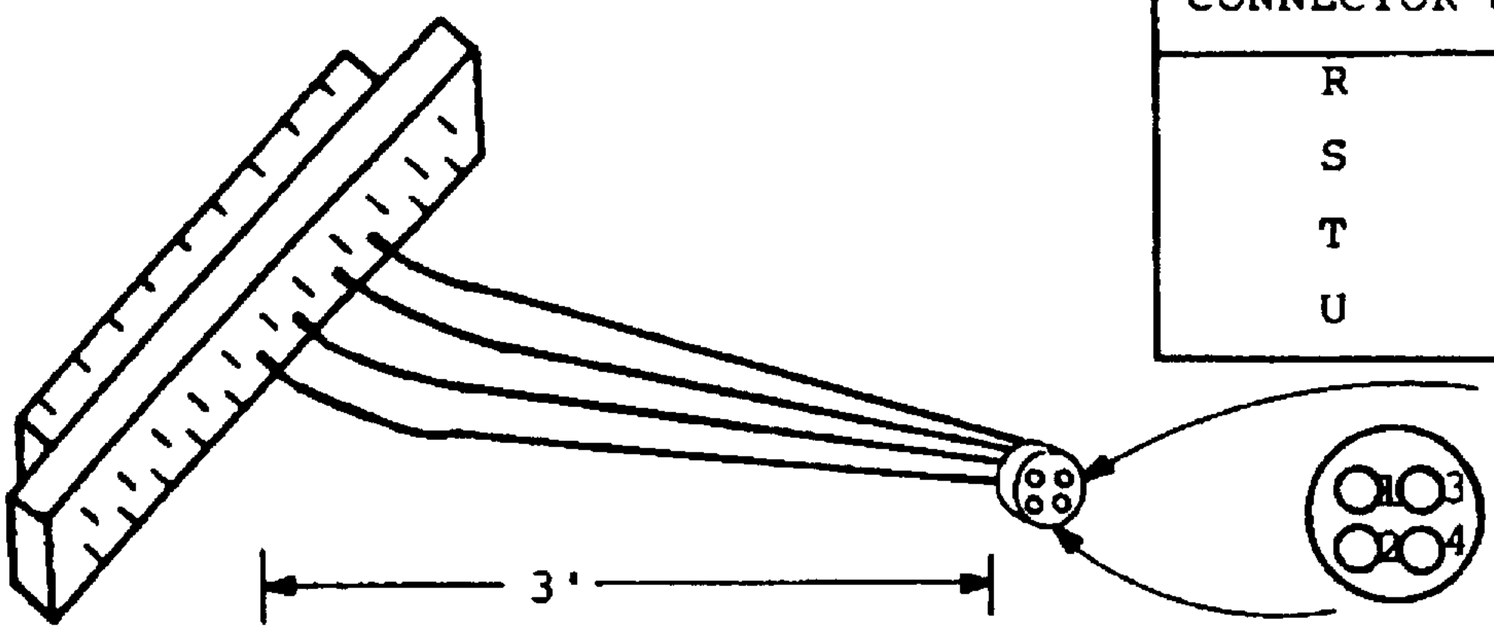
Since the TTY printer is wider than the AIM 65 printer, the printer output may be formatted differently.

9.2.1 Interface Considerations

AIM 65 provides a 4-wire, 20 mA current loop interface to the TTY. The TTY must be configured to operate with this interface, and to operate in full-duplex mode. Figure 9-6 shows a typical connection to a TTY ASR 33.

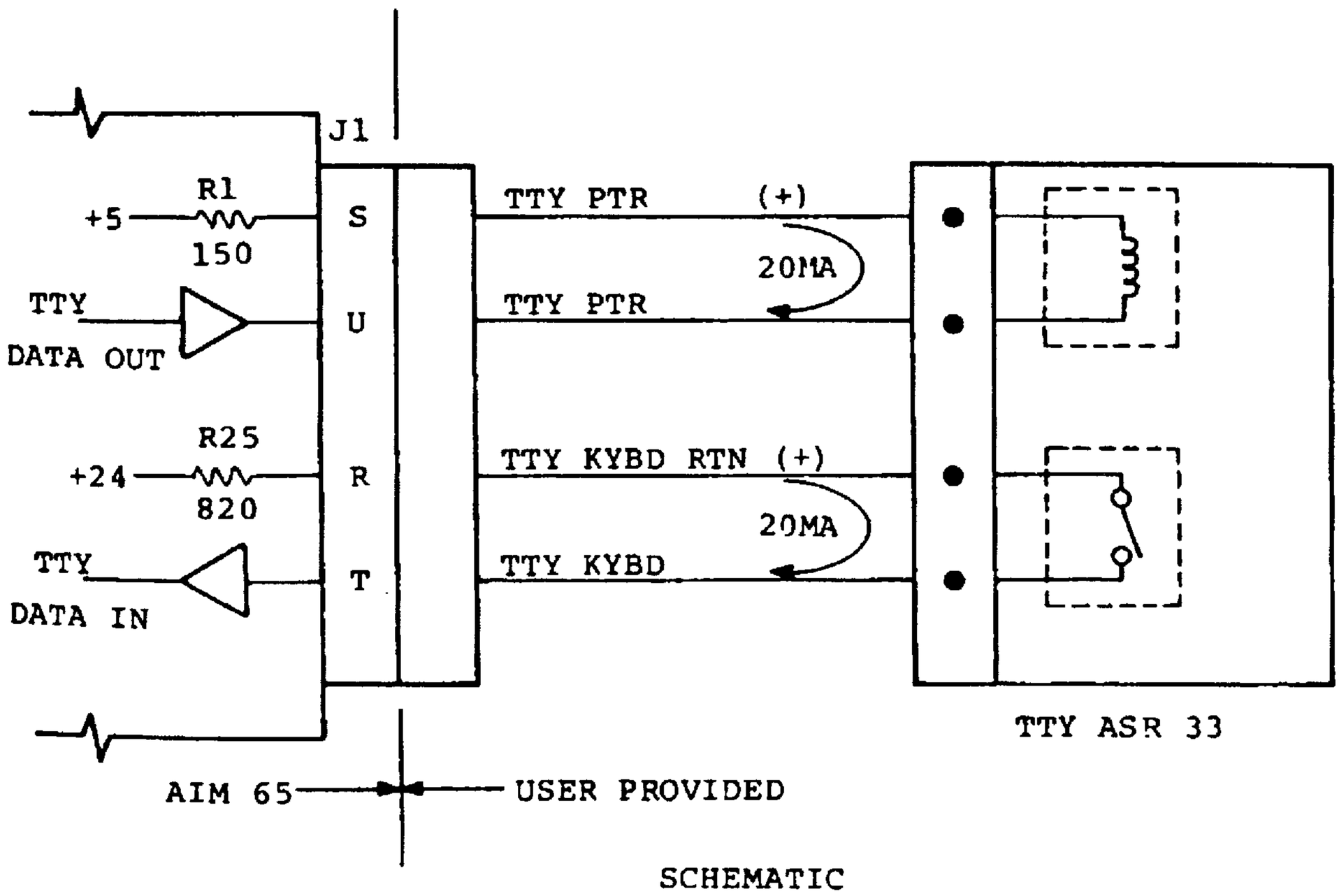
AIM 65 adjusts automatically to the TTY data transmission rate so no special adjustments are required.

AIM 65 CONNECTOR J1	TTY 4 PIN RECEPTACLE
R	2
S	4
T	3
U	1



FRONT VIEW

PICTORIAL OF SUGGESTED INTERFACE WIRING



SCHEMATIC

Figure 9-6. AIM 65 to TTY Connection

9.2.2 TTY Installation and Turn-On Procedure

1. Disconnect power from the TTY.

2. Configure the TTY for 4-wire, 20 mA current loop operation in full duplex mode per the TTY manufacturer's instructions.

3. Ensure AIM 65 power is turned off.

4. Connect the four TTY interface lines to the AIM 65 application connector (J1) per Figure 9-6.

5. Position the TTY control switch to OFF.

6. Apply TTY power.

7. Position the AIM 65 KB/TTY switch to KB.

8. Apply AIM 65 power. AIM 65 will display/print:

```
ROCKWELL AIM 65
<
```

9. Position the AIM 65 KB/TTY switch to TTY.

10. Position the TTY control switch to LINE.

11. Press AIM 65 RESET button.

12. Type RUBOUT on the TTY. AIM 65 will print:

```
ROCKWELL AIM 65
<
```

NOTE

This step is important since AIM 65 adjusts automatically to the TTY data transmission rate in response to the typing of RUBOUT. Once the data rate is computed, keyboard control can be subsequently transferred between AIM 65 and TTY without use of the RESET and RUBOUT entries.

All AIM 65 Monitor and Editor features described in Sections 3, 4, and 5 are available. Consult the following sections for any differences in operation for specific TTY operations.

If the ROCKWELL AIM 65 message and prompt do not appear, the hookup is probably incorrect. Repeat Steps 1 through 12. If the problem persists, refer to the TTY troubleshooting procedure in Section 11.

9.2.3 AIM 65 to TTY Keyboard Transfer

If the AIM 65 Keyboard is active and it is desired to switch to TTY keyboard operation, one of two procedures may be followed:

1. To use a send-and-receive TTY for the first time after AIM 65 power is turned on:

A. Position the KB/TTY switch to TTY.

B. Depress the AIM 65 RESET button.

C. Position the TTY control switch to LINE.

D. Type RUBOUT on the TTY.

AIM 65 will respond by entering the Monitor and printing:

```
ROCKWELL AIM 65
<
```

The next keyboard entry should be made from the TTY keyboard.

2. If the TTY keyboard has previously been active since AIM 65 power turn on and the computed and stored TTY data transmission rate has not been altered:

   A. Position the KB/TTY switch to TTY.

   B. Type SPACE on the keyboard. Control will switch to the TTY keyboard. If control does not transfer to the TTY keyboard, press AIM 65 RESET and type RUBOUT on the TTY to enter and initialize the Monitor.

3. AIM 65 can be used to communicate with a terminal over the 20 ma. current loop or SERIAL IN lines at rates up to 9600 baud. If the terminal cannot transmit

the RUBOUT character, the baud rate must be manually entered into memory before setting the KB/TTY switch to TTY. The baud rate can be specified as follows:

BAUD	CNTH30 ($A417)	CNTL30 ($A418)	MAXIMUM CHARACTERS/SECOND
110	23	3F	10
150	19	B7	15
300	0C	C2	30
600	06	3F	60
1200	02	FD	120
2400	01	5D	240
4800	00	8D	240
9600	00	25	240

The 240 character/second rate limitation above is due to the following:

- When the KB/TTY switch is in the TTY position, up to 4 milliseconds are required to display the incoming character.

- When the KB/TTY switch is in the KB position, the rate is limited due to the time required to process the TTY keyboard or SERIAL IN characters, even though they are not displayed.

9.2.4 TTY to AIM 65 Keyboard Transfer

To switch from the TTY keyboard to the AIM 65 keyboard:

1. Position the KB/TTY switch to KB.

2. Type the SPACE key on the TTY. Control will transfer to the AIM 65 keyboard. If control does not transfer to the AIM 65 keyboard, press AIM 65 RESET to enter and initialize the Monitor.

9.2.5 TTY Keyboard Operation Differences

Since the AIM 65 and the TTY keyboards are nearly identical, the command and data entry procedures are essentially the same. There are minor differences, however, that must be considered to ensure proper operation.

Use of RUBOUT Key

The TTY RUBOUT or DEL key is used the same as the AIM 65 DEL key: to delete input characters. When the AIM 65 DEL is typed, the deleted character is erased from the display and the character input cursor backspaced one position. When used on a TTY, a slash character is printed to indicate character deletion.

Character Input Cursor

No character input cursor (^) is printed on the TTY; the TTY print head shows character input position.

Line Input Cursor

In the Editor Read R and Insert I commands, a line input cursor (*) is printed to indicate the start of an input line.

Break/Escape

In some Monitor and Editor commands (see Table 2-5), the keyboard is sampled at the end of an output line to determine if ESC is typed for return to the Monitor. In these modes, the TTY BREAK key must be held down during the output processing until the output has ended. After releasing the BREAK key, ESC can then be typed to cause Monitor re-entry.

CAUTION

If the TTY hangs up in a local mode, TTY control can be returned to the AIM 65 Monitor by depressing AIM 65 RESET then TTY RUBOUT.

User Function Keys

To enter the user functions from a TTY,type the following keys:

	AIM 65 KEYBOARD	TTY KEYBOARD
User Function 1	F1	[
User Function 2	F2	]
User Function 3	F3	^

Print

The PRINT and CTRL PRINT functions are not available on the TTY keyboard.

9.2.6 Punching Paper Tape

Paper Tape may be punched by any AIM 65 command allowing output device code =L. These commmands are:

COMMAND	DATA TYPE	DATA FORM
Monitor Dump Command D	Object	Hexadecimal
Editor List Command L	Text	ASCII
Assembler Object Code Output	Object	Hexadecimal

The following procedure can be used to punch paper tape using either the AIM 65 or the TTY keyboard :

1. Position the TTY control switch to LOCAL.

2. Depress the TTY tape punch ON button.

3. Type the TTY HERE IS key a few times to obtain about 8 inches of clean paper tape leader.

4. Press the TTY tape punch OFF button.

5. Position the TTY control switch to LINE.

6. Set up the AIM 65 command using either the AIM 65 or TTY keyboard until AIM 65 asks for the output device code:

   OUT=

7. Press the TTY tape punch ON button.

8. If the AIM 65 keyboard is active, position the KB/TTY switch to TTY, then type L on the AIM 65 keyboard.

   If the TTY keyboard is active, type L on the TTY keyboard.

   The output will be punched on the TTY paper tape.

   NOTE

   If object code is being dumped, AIM 65 will ask MORE? at the end of the dump. Continue the dump by typing Y and responding to prompts; end the dump by typing N. Leave the punch on until the Monitor prompt < is displayed. The messages and prompts will be punched on the tape but will be ignored during subsequent load of object code.

9. Press the TTY tape punch OFF button.

10. Position the TTY control switch to LOCAL.

11. Press the TTY tape punch ON button.

12. Type HERE IS a few times to punch clear trailer tape containing only sprocket holes.

13. Press the TTY tape punch OFF button.

9.2.7 Reading Paper Tape

Punched paper tape may be read by any AIM 65 command allowing input device code =L. These commands are:

COMMAND	DATA TYPE	DATA FORM
Monitor Load Command L	Object	Hexadecimal
Editor Read Command R	Text	ASCII
Assembler Source Code Input	Text	ASCII

The following procedure can be used to read punched paper tape using either the AIM 65 or the TTY keyboard:

1. Set up the AIM 65 command using either the AIM 65 or the TTY keyboard until AIM 65 asks for the input device code:

   IN=

2. Position the paper tape in the TTY tape reader with the clear leader over the read head.

3. If the AIM 65 keyboard is active, position the AIM 65 KB/TTY switch to TTY, then type L on the AIM 65 key board.

   If the TTY keyboard is active, type L on the TTY keyboard.

4. Start the TTY tape reader.

SECTION 10
EXPANDING AIM 65

AIM 65 provides for on-board and off-board expansion. On-board RAM, ROM, or PROM can be added by simply plugging the device into an appropriate vacant socket on the Master Module. Off-board expansion is also readily accomplished since all address, data, and control bus lines are brought out to the J3 Expansion Connector.

Commercially available motherboards allow the AIM 65 to be interfaced to a variety of standard busses, including the Rockwell R6500 bus, the Motorola M6800 bus, and the S-100 bus.

10.1 ON-BOARD RAM EXPANSION

On-board RAM may be expanded up to 4K (4096_{10} or $0FFF_{16}$) by plugging in R2114 static RAM devices or equivalent. The RAM socket pin assignments are shown in Figure 10-1. These devices must be added in pairs since each socket provides only four data bits, i.e., one half of the required 8-bit data width of a byte. The optional RAM pairs are:

ADDRESS	SOCKETS
0400-07FF	Z6, Z7
0800-0BFF	Z11, Z12
0C00-0FFF	Z17, Z18

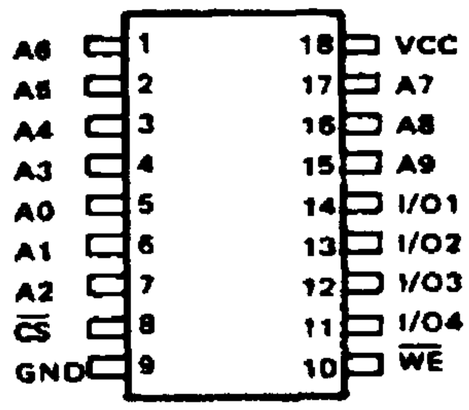


Figure 10-1. RAM Socket Pin Assignments

Install the RAM as follows:

1. Turn off AIM 65 power.

2. Align pin 1 of the RAM device with pin 1 of the socket.

3. Carefully insert the RAM device into the socket while applying pressure under the board to prevent board flexing until all pins are firmly seated.

CAUTION

Improper pin alignment may cause the pins to bend back during insertion and not make proper contact. If this happens, remove the device from the socket, very carefully straighten any bent pins, and reinsert the device back into the socket.

4. Turn on AIM 65 power.

5. Verify proper RAM installation and operation by entering the AIM 65 Text Editor and establishing the Text Buffer limits over the range of the added RAM. The Text Editor checks one address on each page for write and read operation. Improper RAM installation or operation will result in a <MEM ERROR.

10.2 ON-BOARD PROM/ROM EXPANSION

On-board PROM/ROM may be added up to 20K ($B000_{16}$ - $FFFF_{16}$) by plugging in AIM 65 optional R2332 ROM, user provided R2332 ROM, or user provided compatible PROM devices. The PROM/ROM socket pin assignments are shown in Figure 10-2.

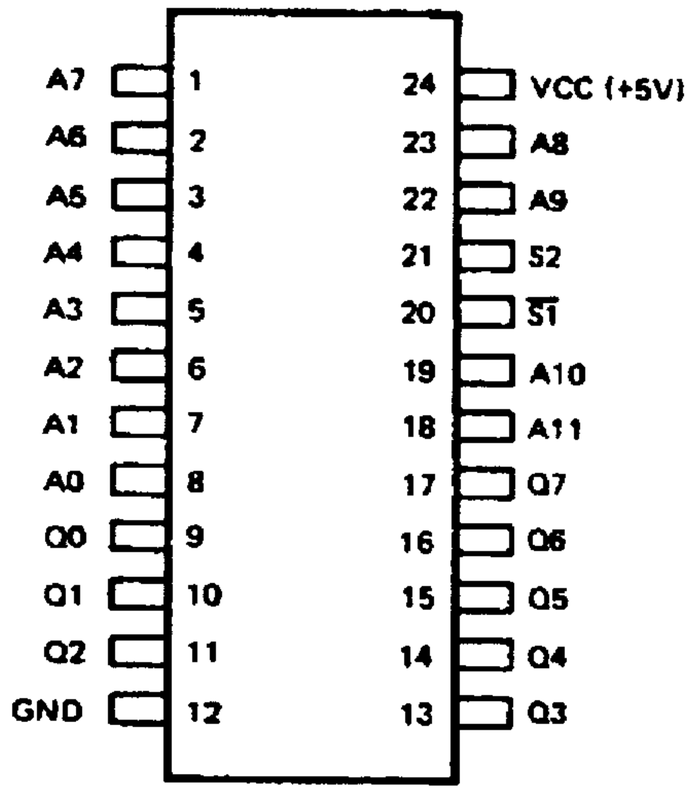


Figure 10-2. PROM/ROM Socket Pin Assignments

Install the PROM/ROM devices as follows:

1. Turn off AIM 65 power.

2. Align pin 1 of the PROM/ROM device with pin 1 of the socket.

3. Carefully insert the PROM/ROM device into the socket while applying pressure under the socket to prevent board flexing until all pins are firmly seated.

CAUTION

Improper pin alignment may cause the pins to bend back during insertion and not make proper contact. If this happens, remove the device from the socket, very carefully straighten any bent pins, and reinsert the device back into the socket.

4. Turn on AIM 65 power.

5. The program in the PROM/ROM devices can be tested by setting the program counter to the program starting address (using the * command) and executing the program using the G command, or by using a function key (F1, F2, or F3) after proper JMP instructions have been loaded ($010C-$010E, $010F-$0111, or $0112-$0114 respectively), or by using the AIM 65 option key linkage. The addresses in the spare PROM/ROM sockets called by the Monitor are:

KEY	SOCKET	ADDRESS
N	Z24	D000
5	Z26	B000
6	Z26	B003

10.3 OFF-BOARD EXPANSION

AIM 65 may be expanded off-board up to the 65K address limit of the R6502. The address, data, and control bus lines on the J3 Expansion Connector provide the interface signals. Expansion module or motherboards may be purchased that plug directly into J3. An expansion module may have components mounted directly on the board that perform desired functions or interface with other circuits. An expansion motherboard will have one or more card slots attached that allow standard interface expansion modules to be installed.

Alternatively, a custom circuit may be designed to interface with J3 through an interface cable and connector.

Whatever the expansion method, the loading limitations of the R6502 address, data, and control lines should be followed. Refer to the R6500 Hardware Manual for the detailed R6502 electrical characteristics.

SECTION 11
TROUBLESHOOTING, WARRANTY, AND SERVICE

Your AIM 65 has been functionally tested at the factory before packing for shipment. Should your AIM 65 not operate correctly, consult the troubleshooting procedure in Table 11-1. If the problem cannot be corrected, return the AIM 65 to Rockwell with a description of the failure in accordance with the following service policies.

11.1 LIMITED 90-DAY WARRANTY

Rockwell International Corporation warrants the R6500 Advanced Interactive Microcomputer (AIM 65), and any approved accessories provided by Rockwell, against defects in materials and workmanship for a period of ninety (90) days from the date of delivery. In case of a defect, Rockwell International will, at its option, repair or replace the AIM 65 without charge.

For service under this warranty:

1. Pack the AIM 65 carefully in its original container, to avoid breakage in transit. The unit must be returned complete, with all components intact.

2. Mail it prepaid and insured to:

   AIM 65 Customer Service Center
   Rockwell International
   6001 Threadgill Avenue
   El Paso, Texas 79924
   Phone 800/351-6018

3. With the AIM 65, enclose a copy of your invoice, sales slip, or other dated proof of purchase showing serial number of unit being returned.

EXCLUSIONS AND LIMITATIONS

This warranty does not extend to any damage or malfunction resulting from misuse, neglect, accident, or repairs or modifications made by other than authorized personnel at the above captioned service facility. This warranty (except as to title) is in lieu of all other warranties, expressed or implied, including merchantability or fitness for any particular purpose, arising by law, custom or conduct. The rights and remedies provided herein are exclusive and in lieu of any other rights or remedies. In no event shall Rockwell International be liable for consequential damages.

11.2 OUT-OF-WARRANTY SERVICE

If the warranty period has expired or the AIM 65 is returned without proof of purchase date or serial number documented, we will repair and return the AIM 65 to you C.O.D., at our prevailing service rates. To save C.O.D. handling and shipping charges, call the Customer Service Center in advance, toll-free 800/351-6018, for the amount of out-of-warranty repair charges, then foward your check payable to Rockwell International. Customers with established Rockwell accounts may enclose a company purchase order.

11.3 PRINTER PAPER

The AIM 65 Printer uses thermal/heat-sensitive paper. This paper is available from our Service Center, as Part Number TT 270, at a cost of 3 rolls for $3.50, plus shipping costs. You may also use Olivetti Type No. 295933R35 or Sears Type 3974. DO NOT USE Texas Instruments Type TP-27225 paper!

11.4 PRINTER ADJUSTMENT

The printer has been adjusted at the factory, and no further adjustment should be required during normal operation. There are four adjustments on the printer that may be required, however, after extended printer operation.

11.4.1 Release Level Print Adjustment

With the head release lever in the PRINT position, wing "A" of the level should not touch the Thermal Head group. There must be visible clearance at "B" so that the Thermal Head group may rest on the platen (see Figure 11-1a).

11.4.2 Release Level Release Adjustment

When the head release lever is in the RELEASE position, the Thermal Head group must be held away from the platen. Minimum clearance is 0.8mm, as shown. To obtain both these conditions, form wings "A" as necessary (see Figure 11-1b).

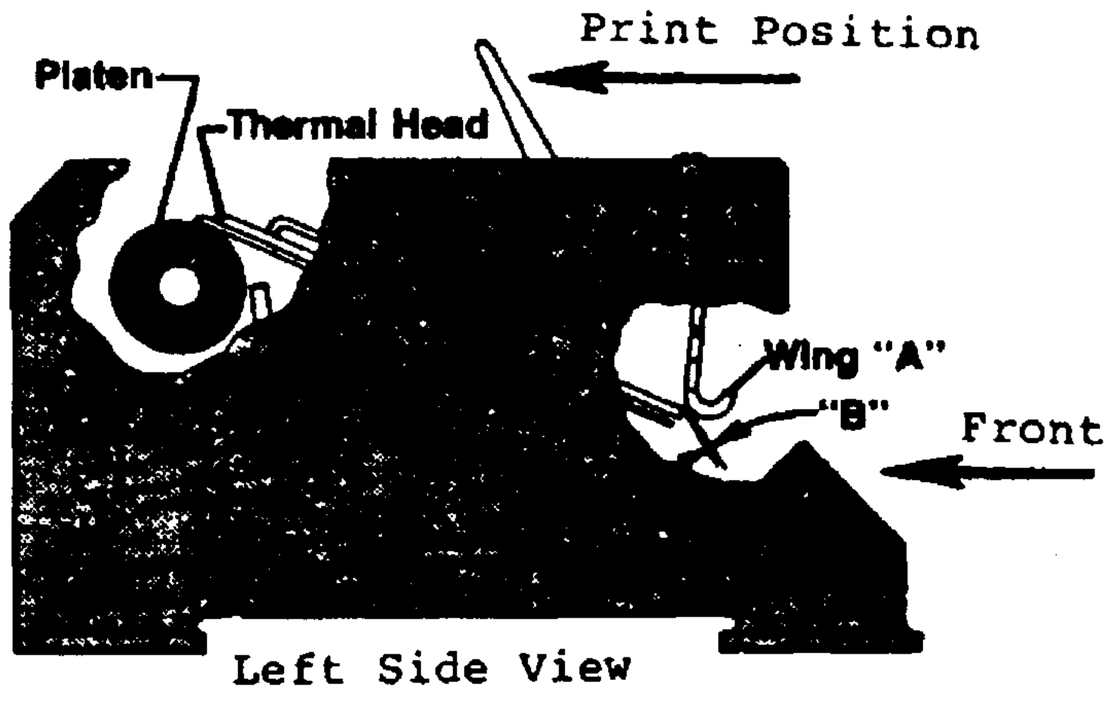
11.4.3 Motor Gear Mesh Adjustment

Motor gear mesh is adjusted by loosening the top and bottom motor mounting screws, and repositioning the motor as necessary. Mesh between the motor and the large transmission gear must be as deep as possible without binding. When this condition is obtained, tighten the motor mounting screws (see Figure 11-1c).

11.4.4 Vertical Dot Alignment Adjustment

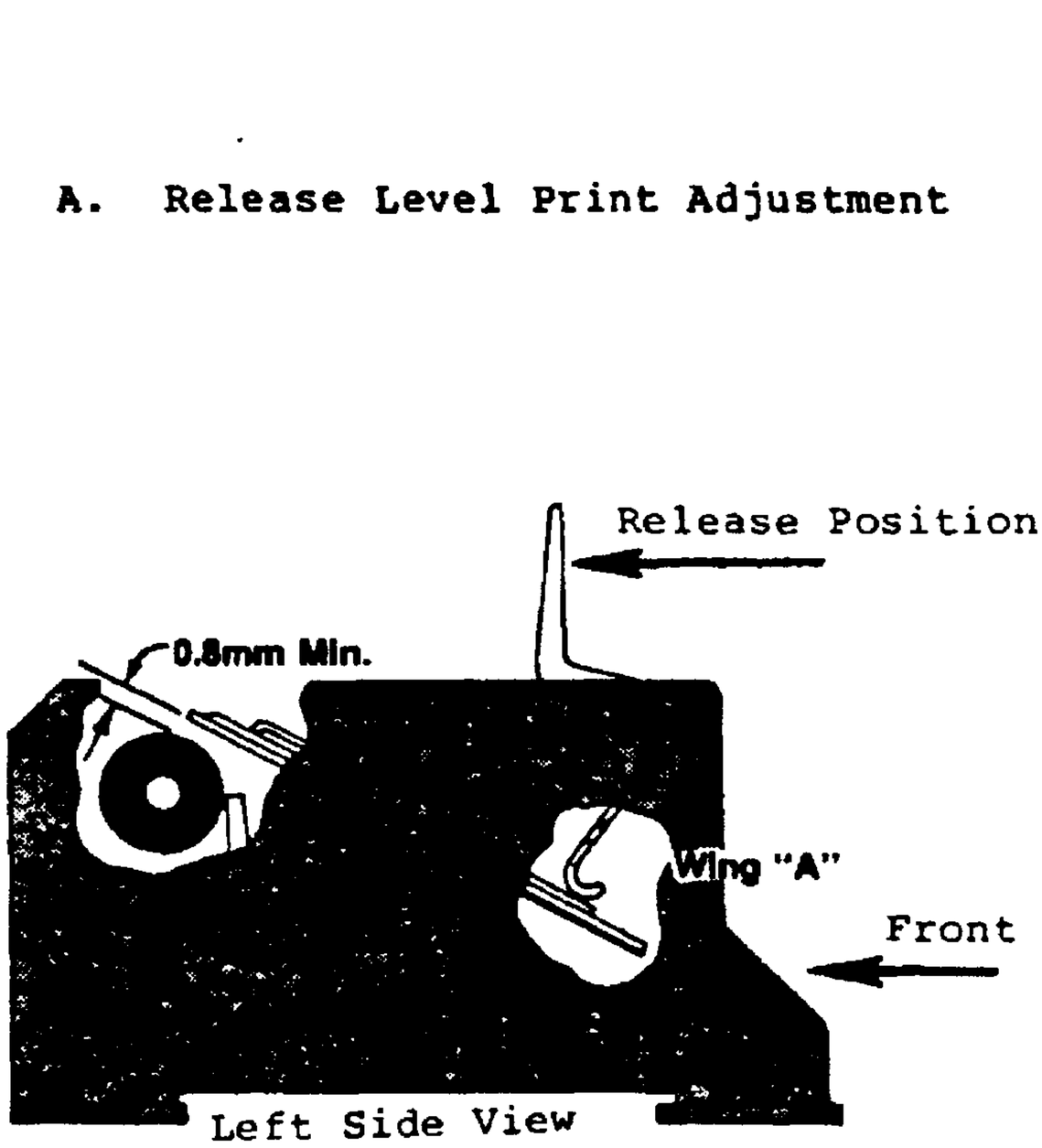
To adjust vertical dot alignments, print a series of eights and ones: 81818181...

Loosen the strobe cap mounting nut slightly. Rotate the strobe cap until all vertical dots are in line. Tighten the strobe cap mounting nut (see Figure 11-1d).



Left Side View

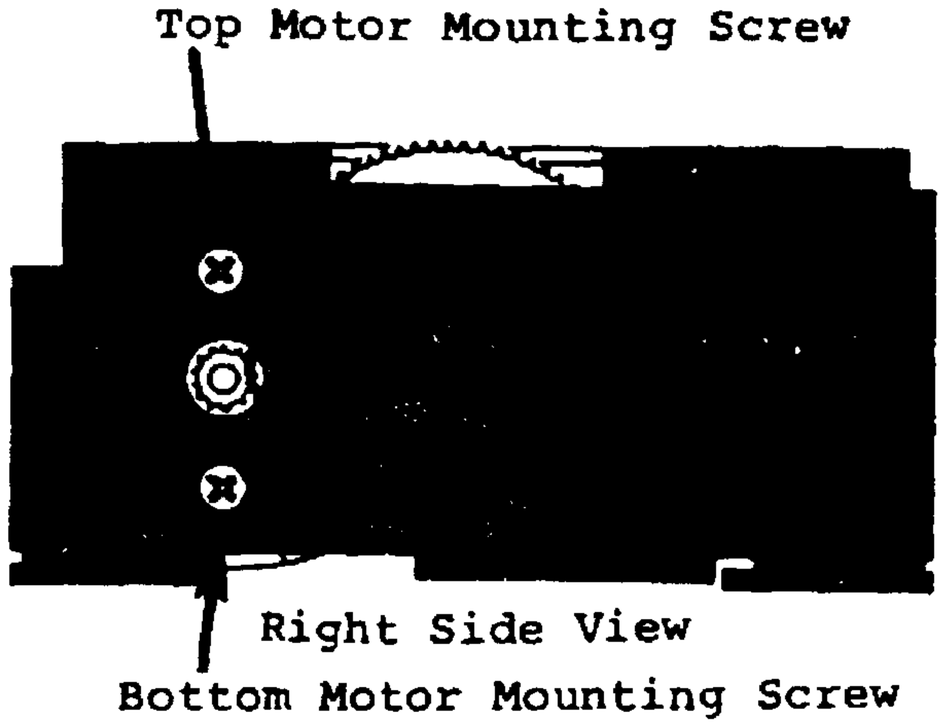
A. Release Level Print Adjustment



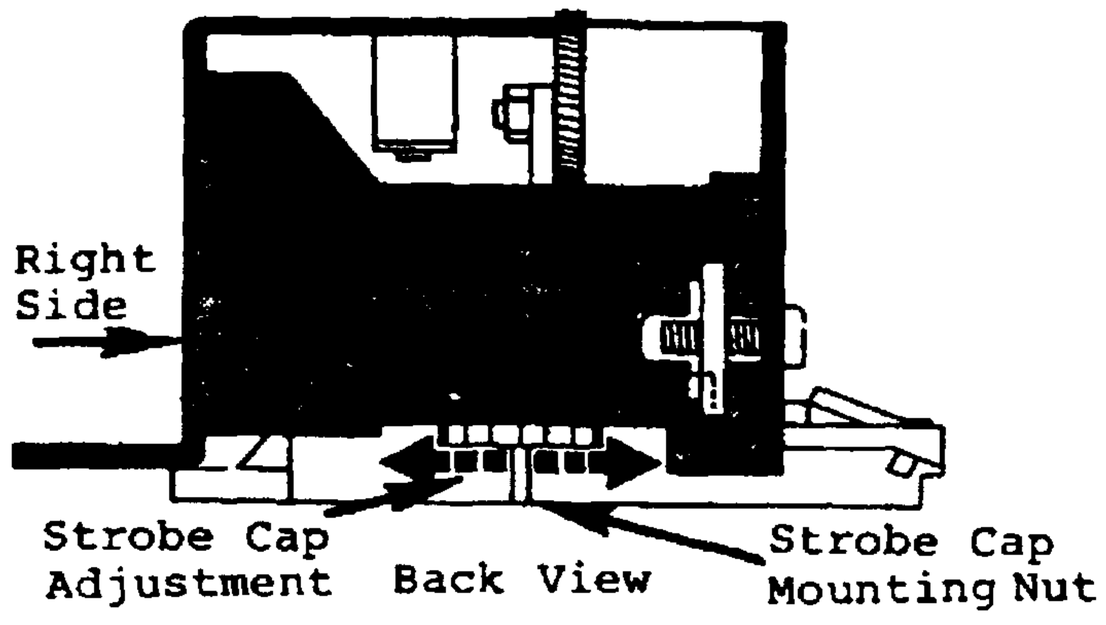
Left Side View

B. Release Level Release Adjustment

Figure 11-1. AIM 65 Printer Adjustments



C. Motor Gear Adjustment



D. Vertical Dot Alignment

Figure 11-1. AIM 65 Printer Adjustments (Cont.)

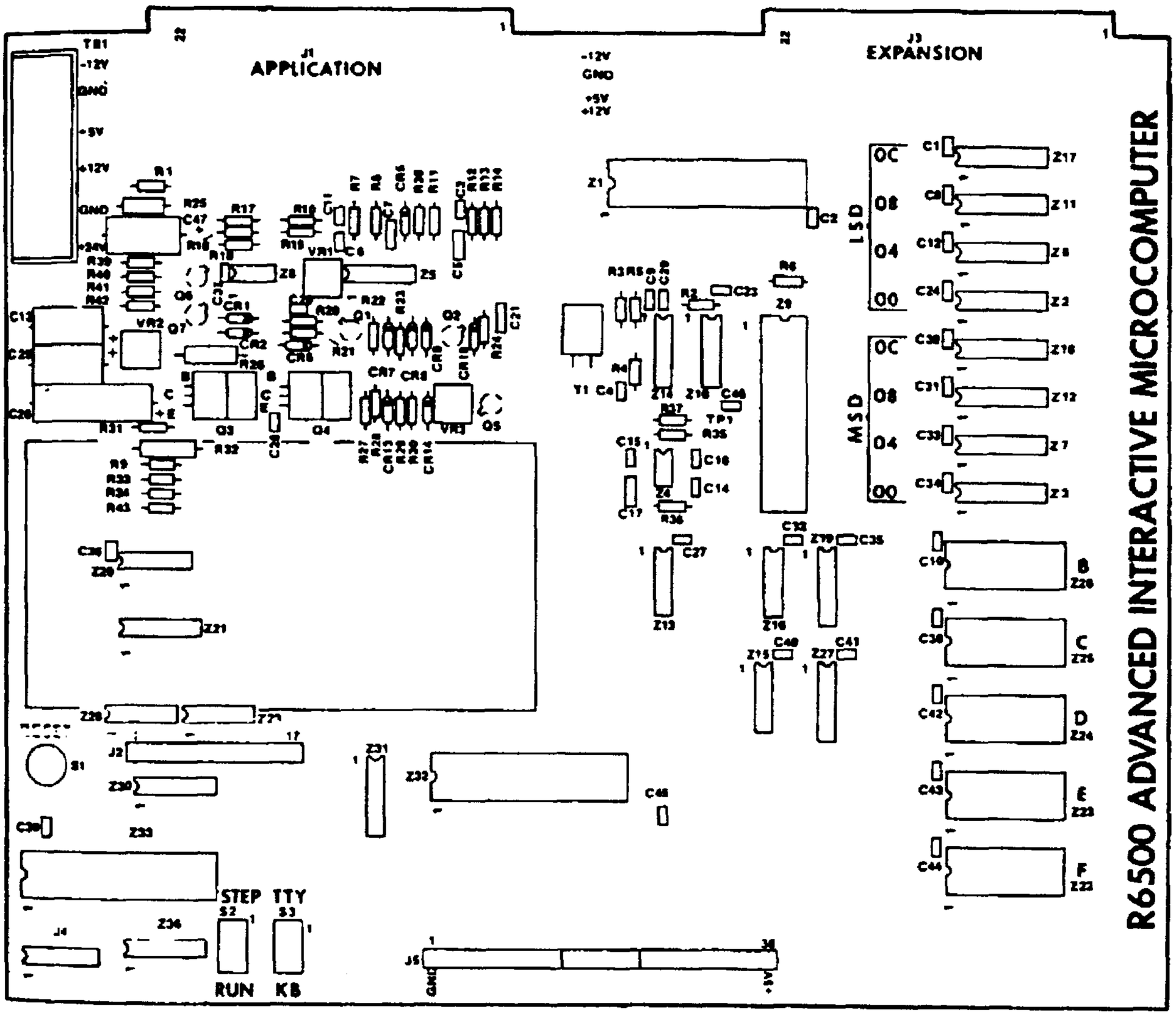


Figure 11-2. AIM 65 Master Module Layout

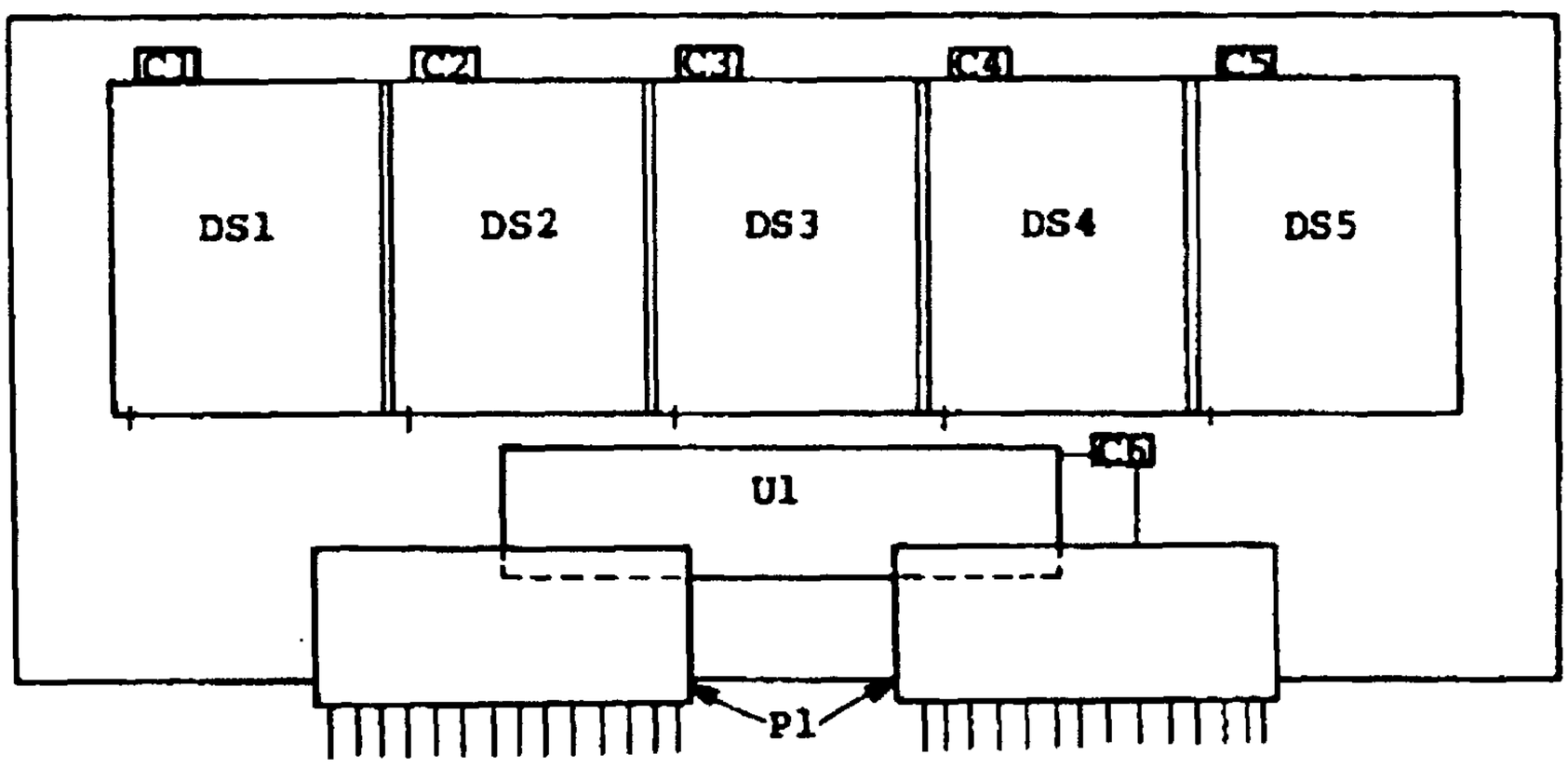


Figure 11-3. AIM 65 Display Module Layout

Table 11-1. Troubleshooting Procedure

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
1. No display	1a. +5V absent or low	1a. Ensure +5±0.5V on TB1-3
	1b. Monitor or user program hung up or lost	1b. Press RESET
	1c Display module pin(s) disconnected from J5	1c. Ensure all display module pins are securely inserted into J5
	1d. R6502 CPU incorrectly installed	1d. Ensure R6502 correctly installed in Z9
	1e. R3222 and R3223 ROMs incorrectly installed	1e. Ensure ROMs are correctly installed in Z22 and Z23, respectively.
	1f. RAM Page 0 and 1 not installed	1f. Ensure R2114 RAMs are installed incorrectly in Z2 and Z3

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
2. No response to keyboard entry	2a. KB/TTY switch not in proper position for selected AIM 65 or TTY keyboard.	2a. Select KB or TTY on KB/TTY switch to match desired keyboard. Press RESET.
	2b. Monitor or User program hung-up or lost.	2b. Press RESET.
	2c. AIM 65 Keyboard Module disconnected from Master Module.	2c. Ensure Keyboard Module to Master Module interconnect cable is securely connected to both modules.
	2d. Stuck key on keyboard.	2d. Release stuck key(s).
3. Printer not printing.	3a. Printer control turned off.	3a. Type CTRL & PRINT after "<" prompt displayed simultaneously until <ON is displayed.

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
	3b. Printer Release lever is in Release position.	3b. Move Printer Release Lever to Print position.
	3c. +24 absent or low.	3c. Ensure +24 ± 1.5V on TB1-6.
	3d. Printer cable . loose.	3d. Ensure printer cable contacts are securely inserted into J2
	3e. Printer cable pins misaligned in J2.	3e. Ensure printer cable contacts are properly aligned in J2
	3f. Z32 R6522 failed.	3f. Replace Z32 with Z1 R6522 to isolate failed R6522
4. Printer not printing one or more columns	4a. See 3d.	4a. See 3d.
	4b. See 3e.	4b. See 3e.
	4c. See 3f.	4c. See 3f.

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
5. Printer printing too light or too dark	5a. Potentiometer VR2 out of adjustment.	5a. Adjust VR2 counter-clockwise to darken printout, or clock-wise to lighten printout
6. Printer printing too fast or too slow.	6a. Potentiometer VR3 out of adjustment.	6a. Adjust VR3 clock-wise for slower operation, or counterclockwise for faster
7. Printer Vertical dots are misaligned	7a. Printer speed is too fast.	7a. Adjust VR3 clockwise for slower operation
	7b. Print vertical dots are out of adjustment.	7b. See Printer Vertical Dot adjustment (Sec-tion 11.4.4).
8. Printer is not printing evenly or consistently.	8a. Loose +24V power or GND connection.	8a. Ensure proper con-nections on power supply and TB1.
	8b. Foreign material between printer elements and paper.	8b. Release Printer Paper Release bar and en-sure nothing is between the print element and the paper.

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
	8c. Printer Thermal Head is not resting on the platen when the Printer Release lever is in the Print position.	8c. See Printer Release level adjustment. (Section 11.4.2)
9. Printer motor runs slow or is stopped when energized.	9a. Motor gear mesh is too tight.	9a. See Printer Gear Mesh adjustment. (Section 11.4.3)
10. Printer motor runs but Thermal Head does not move.	10a. Motor gear mesh is too loose.	10a. See Printer Gear Mesh adjustment. (Section 11.4.3)
	10b. Printer Release lever is in Release position.	10b. Move lever to Print position.
11. Incorrect Assembler operation.	11a. Incorrect R3224 ROM installation.	11a. Ensure R3224 ROM is correctly installed in Z24.

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
12. Incorrect BASIC operation.	12a. Incorrect R3225 and R3226 ROM installation.	12a. Ensure R3225 and R3226 ROMs are correctly installed in Z25 and Z26, respectively.
13. Audio Tape Recorder Motor does not operate.	13a. Inoperative Recorder.	13a. Disconnect all AIM 65 lines from recorder and verify proper recorder operation.
	13b. Incorrect recorder control line installation.	13b. Verify recorder line installation per Section 9.1
	13c. Incomplete recorder control line connection.	13c. Remove control line from recorder. Put recorder in Play Mode and verify tape movement. With at least one audio line (IN or OUT) attached and proper tape control line ON, connect tape control line to recorder and verify continued

Table 11-1. Troubleshooting Procedure (Cont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
		recorder motor operation. Wiggle tape control line plug in recorder REM jack to ensure proper plug connection.
	13d. Wrong tape control line pair used.	13d. Try the other tape control line pair.
14. Audio Tape does not read properly.	14a. Inoperative Recorder	14a. See 13a.
	14b. Incorrect recorder installation.	14b. Verify recorder interface connection and checkout per Section 9.1
	14c. Low recorder volume adjustment on play.	14c. Adjust recorder volume to maximum.
	14d. Incompatible tape format parameter values.	14d. Ensure input device (T or K) equals the recorded output device (T or K).

Table 11-1. Troubleshooting ProcedureCont.)

SYMPTOM	POSSIBLE CAUSE	CORRECTIVE ACTION
	14e. Incompatible tape speed.	14e. Ensure TSPEED ($A408) value equals recorded tape speed, i.e., $C7 (default), $5B or $5A.
	14f. Gap size is too short for Editor update or Assembler input.	14f. Ensure GAP ($A409) equals $80 on recorded tape. NOTE Default value is $08.

APPENDIX A
AIM 65 COMMAND DEFINITIONS

[ADDRESS] Hexadecimal address, one to four characters

[BYTE] Two-digit hexadecimal value from 00 to FF.

[DECIMAL NUMBER] A two-digit decimal number in the range 00 to 99.

[FILE NAME] A string of 1 to 5 characters.

[HEX OPERAND] The instruction operand.

Addressing Mode	Operand Format
Immediate	#HH
Zero Page	HH
Zero Page, X	HH, X or HHX
Zero Page, Y	HH, Y or HHY
Absolute	HHHH
Absolute, X	HHHH, X or HHHHX
Absolute, Y	HHHH, Y or HHHHY
Relative	HH or HHHH
(Indirect, X)	(HH,X) or (HHX) or (HH,X or (HHX
(Indirect), Y	(HH,Y or (HH,Y)
(Absolute Indirect)	(HHHH)

[INPUT DEVICE] RETURN or SPACE — AIM 65 Keyboard (S2 = KB) or TTY Keyboard (S2 = TTY)

- M — Memory
- T — Audio Tape, AIM 65 format
- K — Audio Tape, KIM-1 format
- L — TTY Paper Tape Reader
- U — User-defined input device

[MNEMONIC OPCODE] A three-letter mnemonic abbreviation.

[OUTPUT DEVICE] RETURN or SPACE — AIM 65 Display/Printer (S2 = KB) or TTY Printer (S2 = TTY)

- P — AIM 65 Printer
- X — Dummy
- T — Audio Tape, AIM 65 format
- K — Audio Tape, KIM-1 format
- L — TTY Paper Tape Punch
- U — User-defined output device

APPENDIX B
AIM 65 MONITOR COMMAND SUMMARY

MAJOR FUNCTION ENTRY COMMANDS

[RESET] — Enter and Initialize Monitor
ROCKWELL AIM 65

E — Enter and Initialize Editor
<E>

T — Re-enter Text Editor at Top of Text
<T>
TOP LINE OF TEXT

N — Enter Assembler
<N>

5 — Enter and Initialize BASIC Interpreter
<5>

6 — Re-enter BASIC Interpreter
<6>

INSTRUCTION ENTRY AND DISASSEMBLY COMMANDS

I — Enter Mnemonic Instruction Entry Mode
<I>
AAAA[*] = [ADDRESS]
AAAA XX [OPCODE][HEX OPERAND]
AAAA XX XX XX

K — Disassemble Memory
<K> * = [ADDRESS]
/[DECIMAL NUMBER]
AAAA XX OPCODE HEX OPERAND

DISPLAY/ALTER REGISTER COMMANDS

* — Alter Program Counter
<*> = [ADDRESS]

A — Alter Accumulator
<A> = [BYTE]

X — Alter X Register
<X> = [BYTE]

Y — Alter Y Register
<Y> = [BYTE]

P — Alter Processor Status
<P> = [BYTE]

S — Alter Stack Pointer
<S> = [BYTE]

R — Display Register Values
<R>

****	PS	AA	XX	YY	SS
0200	00	00	01	02	FF

DISPLAY/ALTER MEMORY CONTENTS

M — Display Specified Memory Locations
<M> = [ADDRESS] XX XX XX XX

SPACE — Display Next 4 Memory Locations
< > AAAA XX XX XX XX

/ — Alter Current Memory Locations
</> AAAA XX XX XX XX

LOAD/DUMP MEMORY COMMANDS

L — Load Object Code into Memory
<L> IN = [INPUT DEVICE]

D — Dump Memory
<D>
FROM = [ADDRESS] TO = [ADDRESS]
OUT = [OUTPUT DEVICE]
MORE? [Y, N]

BREAKPOINT MANIPULATION COMMANDS

— Clear All Breakpoints
<#> OFF

4 — Toggle Breakpoint Enable
<4> OFF/ON

B — Set/Clear Breakpoint Address
<B> BRK/[0, 1, 2, 3] = [ADDRESS]

? — Display Breakpoint Addresses
<?>
AAAA AAAA AAAA AAAA

EXECUTION/TRACE CONTROL COMMANDS

G — Start Execution of User's Program
<G>/[DECIMAL NUMBER]

Z — Toggle Instruction Trace Mode
<Z> ON/OFF

V — Toggle Register Trace Mode
<V> ON/OFF

H — Trace Program Counter History
<H>
AAAA
⋮
AAAA

CONTROL PERIPHERAL DEVICES

CTRL PRINT — Toggle Printer On/Off
<CTRL><PRINT>

PRINT — Print Display Contents
<PRINT>

LF — Advance Printer Paper
<LF>

1 — Toggle Tape 1 Control On/Off
<1>

2 — Toggle Tape 2 Control On/Off
<2>

3 — Tape Verify Block Checksum
<3> IN = [T] F = [FILE NAME] T = [1, 2]

USER FUNCTION COMMANDS

F1 — Call User Function 1
<F1>

F2 — Call User Function 2
<F2>

F3 — Call User Function 3
<F3>

ENTER AND EXIT EDITOR COMMANDS

E — Enter and Initialize Editor
<E>
EDITOR
FROM = [ADDRESS] TO = [ADDRESS]
IN = [INPUT DEVICE]
Note: Defaults are TO = $0200,
FROM = Last contiguous RAM, IN = Keyboard

Q — Exit the Text Editor and Return to Monitor
= <Q>

LINE ORIENTED COMMANDS

R — Read Lines into Text Buffer from Input Device
= <R>
IN = [INPUT DEVICE]

I — Insert One Line of Text Ahead of Active Line
= <I>
INSERTED TEXT LINE
ACTIVE LINE OF TEXT

K — Delete Current Line of Text
= <K>
DELETED LINE OF TEXT
ACTIVE LINE OF TEXT

U — Move the Text Pointer Up One Line
= <U>
PRIOR LINE OF TEXT

D — Move the Text Pointer Down One Line
= <D>
NEXT LINE OF TEXT

T — Move the Text Pointer to the Top of the Text
= <T>
TOP LINE OF TEXT

B — Move the Text Pointer to the Bottom of the Text
= <B>
BOTTOM LINE OF TEXT

L — List Lines of Text to Output Device
= <L>
/[DECIMAL NUMBER]

SPACE — Display the Active Line
= < >
ACTIVE LINE OF TEXT

STRING ORIENTED COMMANDS

F — Find a Character String
= <F>
[CHARACTER STRING]
LINE CONTAINING CHARACTER STRING

C — Change a Character String
= <C>
[OLD STRING]
LINE CONTAINING OLD STRING
TO = [NEW STRING]
SAME LINE, WITH NEW STRING

APPENDIX D
AIM 65 ASSEMBLER COMMAND SUMMARY

D.1 ASSEMBLER COMMAND SEQUENCE

```
<N>
ASSEMBLER
FROM = [ADDRESS] TO = [ADDRESS]
IN = [INPUT DEVICE]
LIST? [Y, N]
LIST-OUT = [OUTPUT DEVICE]
OBJ? [Y, N]  Note: N = Object code to Memory
OBJ-OUT = [OUTPUT DEVICE]  Note: Prompts only on Y response to OBJ?
PASS 1
 SYM TBL OVERFLOW
 ASSEMBLER                  } Displayed only if Symbol Table overflows
PASS 2
= = AAAA LABEL
OBJECT CODE   MNEMONIC OPCODE   } Displayed only if
SYMBOLIC OPERAND   ;COMMENT     } LIST?Y, or LIST?N
                                  and error detected
* * ERROR NN  Note: Error code displayed only on error
ERRORS =  MMMM  Decimal count of errors detected
```

D.2 ASSEMBLER EXPRESSIONS

ELEMENTS

Numeric constants — may be written in one of four bases.

Prefix Character	Base
(none)	10 (Decimal)
$	16 (Hexadecimal)
@	8 (Octal)
%	2 (Binary)

OPERATORS

Type	Operator	Operation
Arithmetic	+	Addition
Arithmetic	–	Subtraction
Special	>	High-Byte Selection
Special	<	Low-Byte Selection

Operators < and > truncate a two-byte value to its high or low byte, respectively.

D.3 ASSEMBLER DIRECTIVES

= — Assigns the value of an operand containing no forward references to either a symbol or the location counter.

$\left\{\begin{matrix}\text{SYMBOL}\\ *\end{matrix}\right\}$ = Operand

.BYTE — Assigns multiple ASCII strings or expressions to consecutive single byte memory locations in high-byte, low-byte order.

.BYT Expression, Expression, . . . Expression

.WORD — Assigns multiple expression operands to consecutive memory locations in low-byte, high-byte order.

.WOR Expression, Expression, . . . Expression

.DBYTE — Assigns multiple expression operands to consecutive double byte (16 bits) memory locations.

.DBY Expression, Expression, . . . Expression

.PAGE — Generates a title under a dashed line.

.PAG $\left\{\begin{matrix}\text{'NEW TITLE'}\\ \text{BLANK}\\ \text{' '}\end{matrix}\right\}$ (New Title)
(No Change of Title)
(Blanks Title)

.SKIP — Generates one blank line.

.SKI

.OPT — Controls assembly listings. All are optional and can be specified in any order or in separate statements.

.OPT $\left\{\begin{matrix}\text{LIS}\\ \text{NOL}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{GEN}\\ \text{NOG}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{ERR}\\ \text{NOE}\end{matrix}\right\}$

.FILE — Last record in a multiple file source program (except the last file) which points to the continuation file.

.FIL File Name

.END — Last record in a single or multiple source file.

.END

D.4 ASSEMBLER ERROR CODES

01	Undefined Symbol
02	Label Previously Defined or Forward Reference to Page 0 Symbol
03	Illegal or Missing Opcode
04	Address Not Valid
05	Accumulator Mode Not Allowed
06	Forward Reference to Page Zero
07	Ran off End of Line
08	Label Does Not Begin with Alphabetic Character
09	Label Greater Than Six Characters
10	Label or Opcode Contains Non-Alphanumeric
11	Forward Reference in Equate
12	Invalid Index — Must Be X or Y
13	Invalid Expression
14	Undefined Assembler Directive
17	Relative Branch Out of Range
18	Illegal Operand Type for This Instruction
19	Out of Bounds on Indirect Addressing
20	A, X, Y, S and P are Reserved Labels
21	Program Counter Negative — Reset to 0

APPENDIX E
ASCII CHARACTER SET

ASCII CHARACTER SET (7-BIT CODE)

LSD \ MSD		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	•	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

NOTES

1. [labeled F1 on AIM 65 keyboard.
   ] labeled F2 on AIM 65 keyboard.
   ↑ labeled F3 on AIM 65 keyboard.

2. ↑ in ASCII appears a ^ on AIM 65 display and printer.

NUL — Null
SOH — Start of Heading
STX — Start of Text
ETX — End of Text
EOT — End of Transmission
ENQ — Enquiry
ACK — Acknowledge
BEL — Bell
BS — Backspace
HT — Horizontal Tabulation
LF — Line Feed
VT — Vertical Tabulation
FF — Form Feed
CR — Carriage Return
SO — Shift Out
SI — Shift In

DLE — Data Link Escape
DC — Device Control
NAK — Negative Acknowledge
SYN — Synchronous Idle
ETB — End of Transmission Block
CAN — Cancel
EM — End of Medium
SUB — Substitute
ESC — Escape
FS — File Separator
GS — Group Separator
RS — Record Separator
US — Unit Separator
SP — Space (Blank)
DEL — Delete

APPENDIX F
AIM 65 AUDIO TAPE FORMAT

The AIM 65 audio cassette tape format is designed to provide fast, reliable recording and reading of both object code and source code. Object code is recorded in binary form, exactly as it appears in memory. Recording object code in binary, is twice as fast as recording it in ASCII, since one byte contains two hexadecimal numbers in binary format, whereas one byte contains only one hexadecimal number in ASCII format.

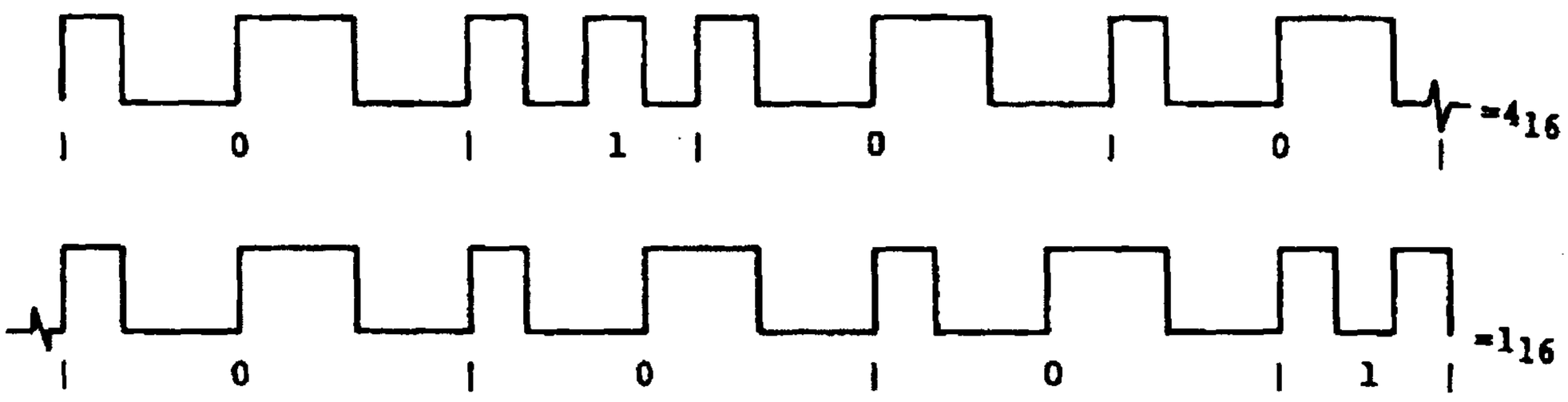
Source code is recorded in ASCII format, the form in which it appears in the Editor Text Buffer.

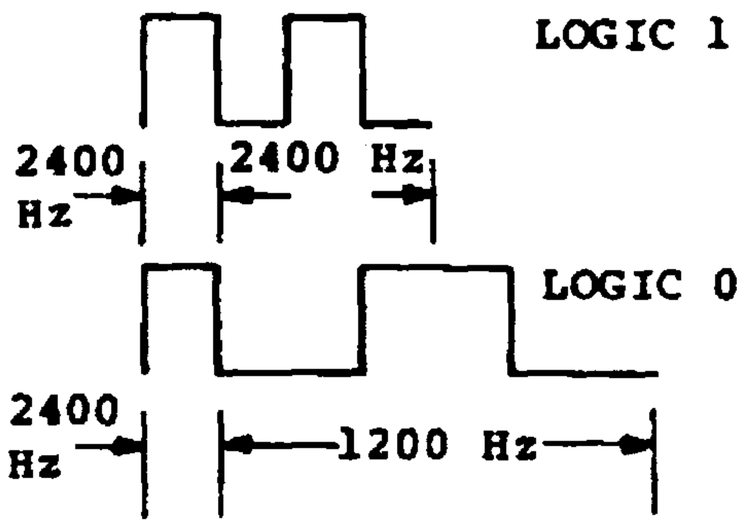
BIT LOGIC STATE DEFINITION

Each transmitted bit begins with a positive one-half cycle of 2400 Hz. tone. The following three half-cycles determine the logic state of the bit. Three half-cycles of 2400 Hz. equal a logic "1". Three half-cycles of 1200 Hz. tone equals a logic "0".

The following example shows eight bits of data. If this data represents one byte, the hexadecimal value of 41 equals two hexadecimal numbers, 4 and 1. If the data is in ASCII format, the character A is represented.

EXAMPLE:





F.1 BLOCKED FORMAT

The data is recorded in blocked format. One block contains 80 bytes of data and 36 bytes of synchronization, control, and block checksum information. The block format is:

SUBFIELD	SYN	#	BLK	OBJECT OR TEXT DATA	BLK CHKSUM
Value	1616...1616	23	HH	XXX...XXXX	HHHH
No. of bytes	(32)	(1)	(1)	(79)	(2)

SYN

The block begins with 32 bytes of Synchronous Idle (SYN) characters (ASCII 16). During read operations, the SYN pattern allows AIM 65 to sense the start of the block and synchronize to the incoming serial data stream.

SYN characters are also used to provide an interblock gap. The number of SYN characters is determined by the contents of address A409 (GAP). The default value of 08, established by a "cold" RESET, generates 32 SYN characters. This value provides a minimum gap for loading object code or source code into an empty Editor text buffer.

For assembling from tape or reading data into a partially-filled Editor text buffer,a larger gap size is required. This allows AIM 65 to stop the tape after a block has been read in order to process the data before reading another block. To lengthen the gap size additional SYN characters are required. The gap value in address A409 should be changed from 08 (32 SYN characters) to 80 (512 SYN characters). This number should be adequate for all audio cassette recorders, but a lower number may be suitable

for your recorder. That number can be determined by experimentation.

#

The character # (ASCII 23) denotes that the data on the tape is recorded in the AIM 65 format, as opposed to the KIM-1 format described in Appendix G.

BLK

The block count (BLK) defines the block number. This number starts with 00 on the first block and increments by one for each block recorded, in hexadecimal, to FF. If more than FF blocks are recorded, the number restarts at 00.

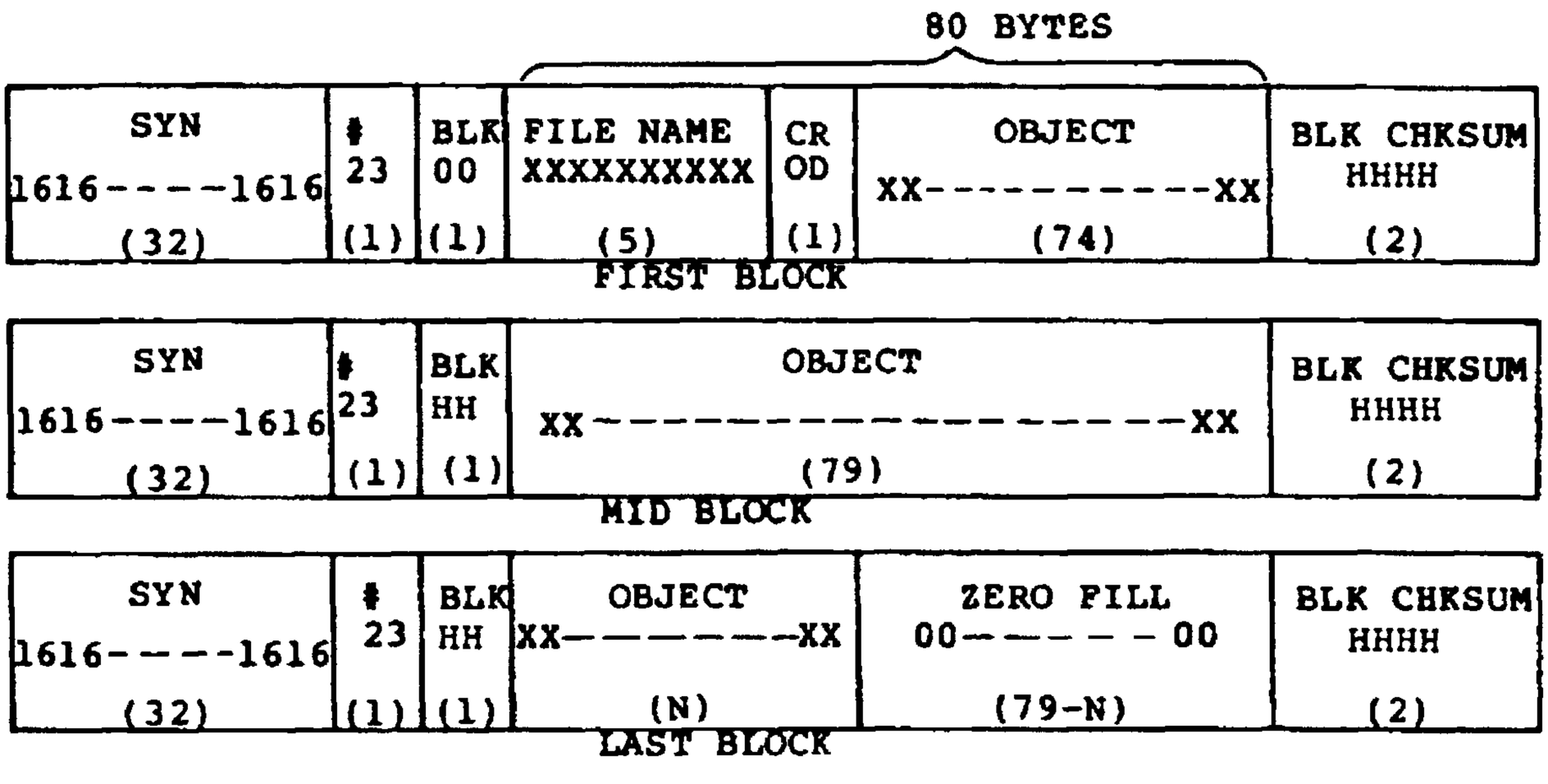
Data

The actual recorded data represents either source or object code. Within each data type are three unique data block types: First block, mid-blocks, and last block. See the object and text data record formats for detailed definition of the data format.

BLK CHKSUM

The block checksum (BLK CHKSUM) is the hexadecimal sum of the 80 data characters, truncated to four hexadecimal digits, (i.e., carry is ignored).

F.2 OBJECT DATA FILE FORMAT



FILE NAME

The file name (FILE NAME) consists of one to five ASCII characters that uniquely identify the file.

CR

The CR character (ASCII 0D) after the FILE NAME indicates that the data format is object rather than text.

Object Data Format

The object data consists of multiple object data records, each containing a starting address and up to 24 bytes of information. The object data is recorded in hexadecimal form. The object data includes both object instructions and data.

The object data record format is:

$$\text{Data Record: } ;N_1N_0A_3A_2A_1A_0\overbrace{D_1D_0}^{1}\overbrace{D_1D_0}^{2}\ldots\overbrace{D_1D_0}^{n}X_3X_2X_1X_0CR$$

$$\text{Last Record: } ;00C_3C_2C_1C_0X_3X_2X_1X_0CR$$

Where:

;	= Start of the record (ASCII 3B)
N_1N_0	= Number of data bytes in the record, in hexadecimal. The maximum number of bytes in one record is 18_{16} (24_{10}).
	= 00 for the last record.
$A_3A_2A_1A_0$	= Address of the first data byte in the record, in hexadecimal.
D_1D_0	= One 8-bit data byte = Two hexadecimal numbers.

$C_3C_2C_1C_0$ = Number of records in hexadecimal, including the data records and the last record.

$X_3X_2X_1X_0$ = Record checksum, in hexadecimal. This is the sum of all the characters in the object code record except the ; character and the record checksum. The checksum is truncated to four hexadecimal digits, i.e., carry is ignored.

CR = Carriage Return (ASCII 0D) which indicates end of record.

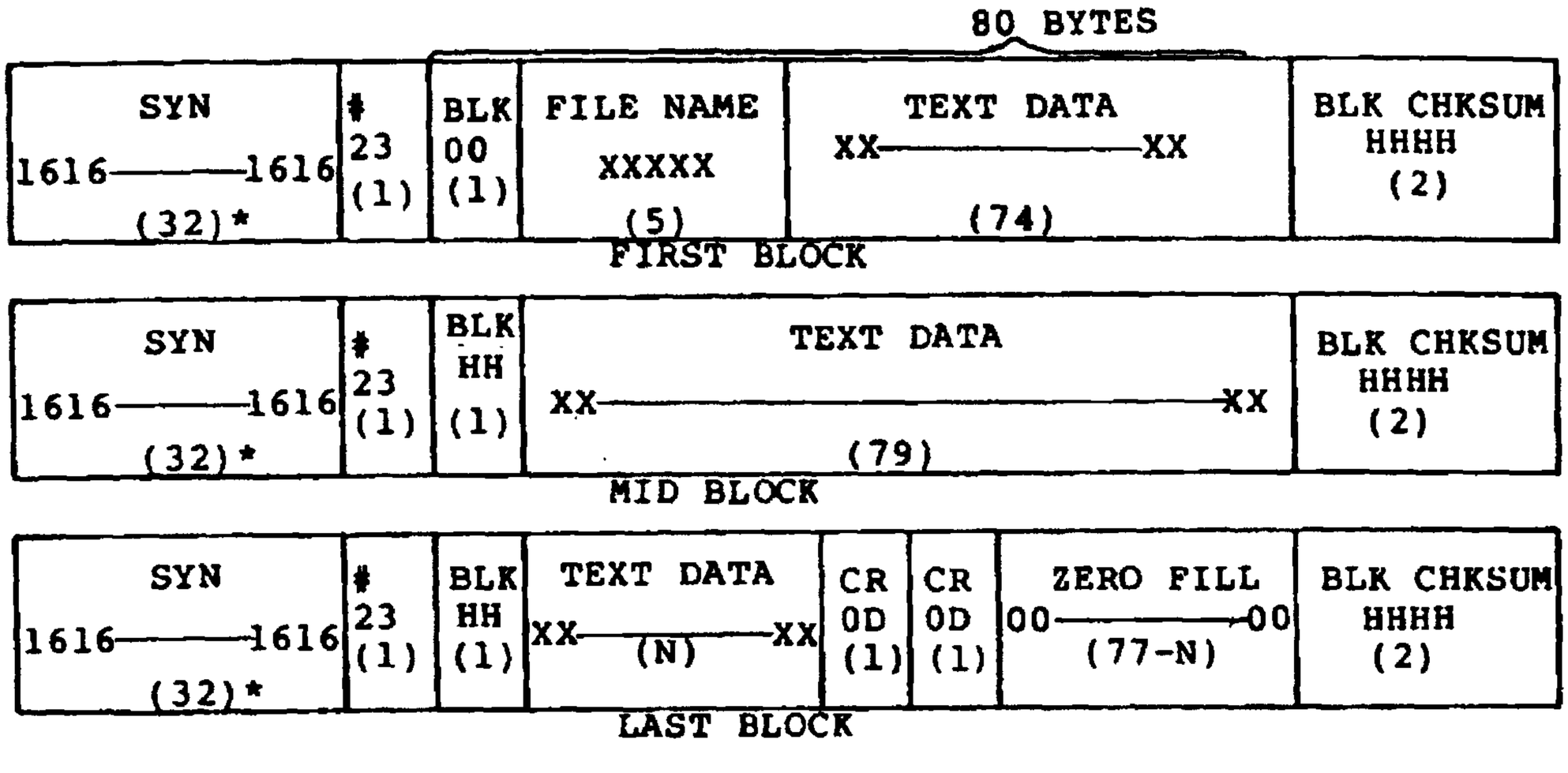
All files contain at least two object code records: the first record and the last record. The last record uniquely identifies the end of the file data.

Since each object code record contains a starting address, various portions of memory can be recorded in one file. Programs or program segments residing in different parts of memory may therefore be recorded on the same file. This simplifies subsequent memory loading procedures, as well as saving load setup time.

Zero Fill

After the last data record is recorded, the remaining data bytes are filled with hexadecimal zeros.

F.3 TEXT DATA FILE FORMAT



*The value shown corresponds to a gap size in $A409 (GAP) of $08.

FILE NAME

The file name (FILE NAME) consists of one to five ASCII characters that uniquely identify the file.

TEXT DATA

The text data consists of characters recorded from the Editor Text Buffer. The data is recorded in ASCII format as it exists in memory. The text data may be the source program for input into the assembler or any text information.

CR

The CR character (ASCII 0D) indicates end of a text record in the text buffer. CR will appear throughout the text buffer separated by no more than 60 characters. Two CR's in succession indicate end of the text file.

ZERO FILL

After the end-of-file indication, the remainder of the block is filled with hexadecimal zeros.

APPENDIX G

KIM-1 AUDIO TAPE FORMAT

Data is transmitted to the tape recorder in the form of serial ASCII encoded characters (seven data bits plus Parity bit). Object data retrieved from the memory is converted into this form by separating each byte into two half-bytes. The half bytes are then converted into their ASCII equivalents.

Each record transmitted begins with a leader of 100 "SYN" characters (ASCII 16) followed by a * character (ASCII 2A). During playback, this pattern allows AIM 65 to detect the start of a valid data record and synchronize to the serial data stream. Following the *, the record identification number (ID), and starting address low (SAL) and the starting address high (SAH) are transmitted. The data specified by the starting (SAL, SAH) and ending limits (EAL, EAH) is transmitted next followed by a "/" character (ASCII 2F) to indicate the end of the data portion of the record. Following the "/" two "CHECK-SUM" bytes are transmitted for comparison with a calculated checksum number during playback to further insure that a proper data retrieval has taken place. Two "EOT" characters (ASCII 04) mark the end of record transmission.

Each transmitted bit begins with a 3700 hertz tone and ends with a 2400 hertz tone. "Ones" have the high-to-low frequency transition at one-third of the bit period. "Zeros" have the transition at two-thirds of the period. During playback the phase locked loop function locks to, and tracks these two frequencies producing a logic "1" pulse of one-third

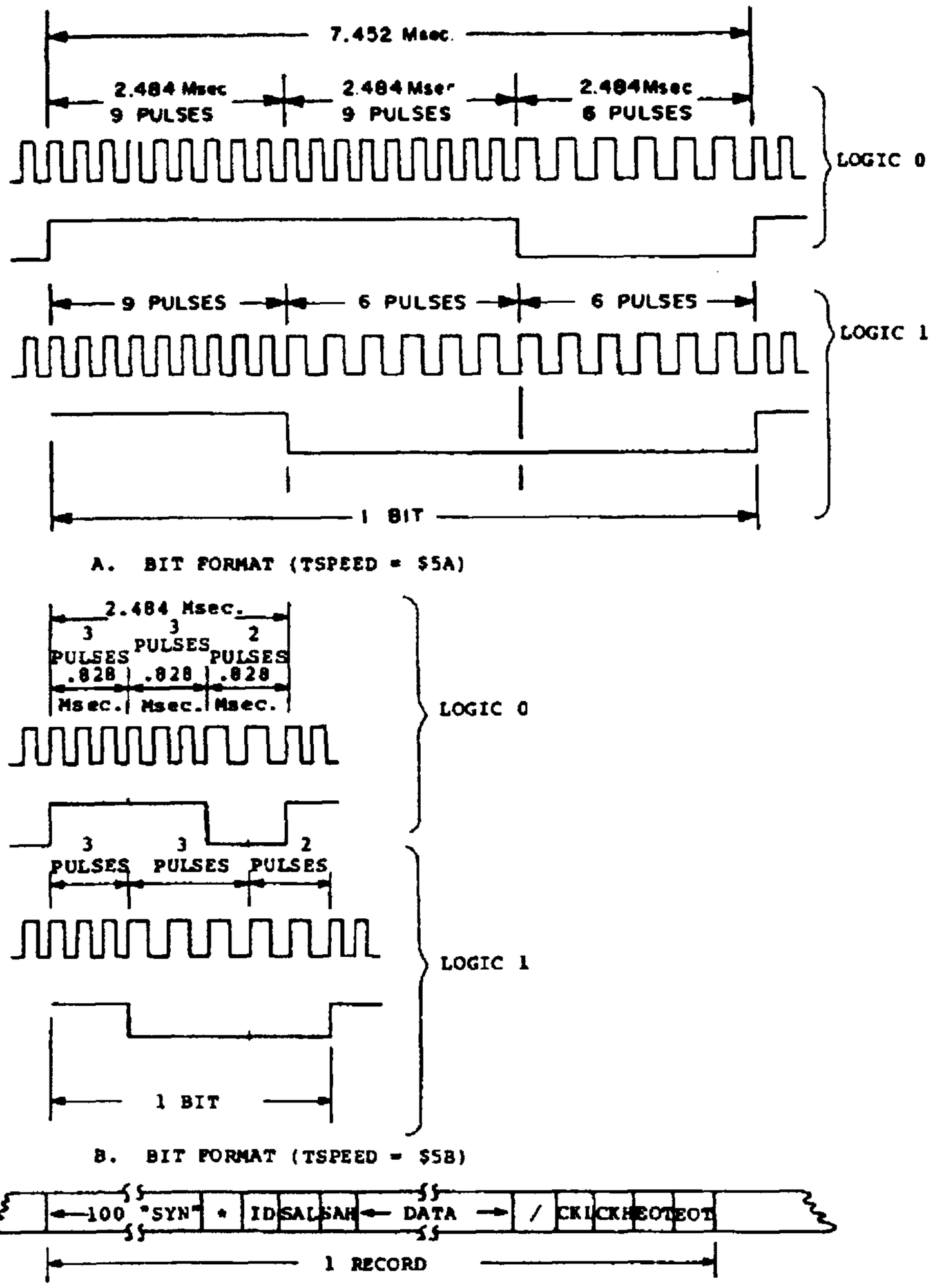


Figure G-1. KIM-1 Audio Tape Format

the bit period for a "One". A pulse two thirds the bit period is likewise produced for a "Zero". AIM 65 Monitor software converts two ASCII characters read from the tape into hexadecimal numbers and packs two numbers into one eight bit byte in memory.

APPENDIX H
PAPER TAPE FORMAT

The AIM 65 Load and Dump commands (Section 3.8) store and retrieve data in a format designed to ensure error free recovery. Each byte of data to be stored is converted to two half-bytes. The half-bytes (possible values are 0 to F) are translated into their ASCII equivalents and written out onto paper tape in this form.

Each output record begins with a ";" character (ASCII 3B) to mark the start of a valid record. The next byte transmitted (18_{16}) or (24_{10}) is the number of data bytes contained in the record. The record's starting address High (1 byte, 2 characters), starting address Low (1 byte, 2 characters), and data (24 bytes, 48 characters) follow. Each record is terminated by the record's checksum (2 bytes, 4 characters), a carriage return (ASCII 0D), line feed (ASCII 0A), and a "DEL" characters (ASCII FF).

The last record transmitted has zero data bytes (indicated by ;00). The starting address field is replaced by a four digit hexadecimal number representing the total number of data records contained in the transmission, followed by the records usual checksum digits. A "XOFF" character ends the transmission.

;180000FFEEDDCCBBAA00998877665544332211223344556677889 90AFC
;0000010001

APPENDIX I

R6500 INSTRUCTION SET

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM			IMPLIED			(IND, X)			(IND), Y			Z PAGE, X			ABS, X			ABS, Y			RELATIVE			INDIRECT			Z PAGE, Y			PROCESSOR STATUS CODES		
																																								7 6 5 4 3 2 1 0		
MNEMONIC	OPERATION	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	OP	n	a	N V - B D I Z C		MNEMONIC
ADC	A + M + C → A (1)(2)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										N V - - - - Z C		ADC
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										N - - - - - Z -		AND
ASL	C ← [0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													N - - - - - Z C		ASL
BCC	BRANCH ON C = 0 (2)																															90	2	2							- - - - - - - -		BCC
BCS	BRANCH ON C = 1 (2)																															B0	2	2							- - - - - - - -		BCS
BEQ	BRANCH ON Z = 1 (2)																															F0	2	2							- - + - - - - -		BEQ
BIT	A∧M				2C	4	3	24	3	2																														M7 M6 - - - Z -		BIT
BMI	BRANCH ON N = 1 (2)																															30	2	2							- - - - - - - -		BMI
BNE	BRANCH ON Z = 0 (2)																															D0	2	2							- - - - - - - -		BNE
BPL	BRANCH ON N = 0 (2)																															10	2	2							- - - - - - - -		BPL
BRK	BREAK													00	7	1																								- - - - 1 - 1 - -		BRK
BVC	BRANCH ON V = 0 (2)																															50	2	2							- - - - - - - -		BVC
BVS	BRANCH ON V = 1 (2)																															70	2	2							- - - - - - - -		BVS
CLC	0 → C													18	2	1																								- - - - - - - 0		CLC
CLD	0 → D													D8	2	1																								- - - - 0 - - -		CLD
CLI	0 → I													58	2	1																								- - - - - 0 - -		CLI
CLV	0 → V													B8	2	1																								- 0 - - - - - -		CLV
CMP	A - M	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										N - - - - - Z C		CMP
CPX	X - M	E0	2	2	EC	4	3	E4	3	2																														N - - - - - Z C		CPX
CPY	Y - M	C0	2	2	CC	4	3	C4	3	2																														N - - - - - Z C		CPY
DEC	M - 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													N - - - - - Z -		DEC
DEX	X - 1 → X													CA	2	1																								N - - - - - Z -		DEX
DEY	Y - 1 → Y													88	2	1																								N - - - - - Z -		DEY
EOR	A⊻M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										N - - - - - Z -		EOR
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													N - - - - - Z -		INC
INX	X + 1 → X													E8	2	1																								N - - - - - Z -		INX
INY	Y + 1 → Y													C8	2	1																								N - - - - - Z -		INY
JMP	JUMP TO NEW LOC				4C	3	3																												6C	5	3							- - - - - - - -		JMP
JSR	JUMP SUB				20	6	3																																		- - - - - - - -		JSR
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										N - - - - - Z -		LDA
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	N - - - - - Z -		LDX
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													N - - - - - Z -		LDY
LSR	0 → [0] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0 - - - - - Z C		LSR
NOP	NO OPERATION													EA	2	1																								- - - - - - - -		NOP
ORA	A∨M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										N - - - - - Z -		ORA
PHA	A → Ms S - 1 → S													48	3	1																								- - - - - - - -		PHA
PHP	P → Ms S - 1 → S													08	3	1																								- - - - - - - -		PHP
PLA	S + 1 → S Ms → A													68	4	1																								N - - - - - Z -		PLA
PLP	S + 1 → S Ms → P													28	4	1																								(RESTORED)		PLP
ROL	[C] ← [0]				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													N - - - - - Z C		ROL
ROR	[C] → [0]				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													N - - - - - Z C		ROR
RTI	RTRN INT													40	6	1																								(RESTORED)		RTI
RTS	RTRN SUB													60	6	1																								- - - - - - - -		RTS
SBC	A - M - C̄ → A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										N V - - - - Z C (3)		SBC
SEC	1 → C													38	2	1																								- - - - - - - 1		SEC
SED	1 → D													F8	2	1																								- - - - 1 - - -		SED
SEI	1 → I													78	2	1																								- - - - - 1 - -		SEI
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										- - - - - - - -		STA
STX	X → M				8E	4	3	86	3	2																												96	4	2	- - - - - - - -		STX
STY	Y → M				8C	4	3	84	3	2													94	4	2																- - - - - - - -		STY
TAX	A → X													AA	2	1																								N - - - - - Z -		TAX
TAY	A → Y													A8	2	1																								N - - - - - Z -		TAY
TSX	S → X													BA	2	1																								N - - - - - Z -		TSX
TXA	X → A													8A	2	1																								N - - - - - Z -		TXA
TXS	X → S													9A	2	1																								- - - - - - - -		TXS
TYA	Y → A													98	2	1																								N - - - - - Z -		TYA

(1) ADD 1 to N IF PAGE BOUNDARY IS CROSSED
(2) ADD 1 TO N IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO N IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT ∴ BORROW
(4) IF IN DECIMAL MODE Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
Ms MEMORY PER STACK POINTER
+ ADD
- SUBTRACT
∧ AND
∨ OR
⊻ EXCLUSIVE OR
M7 MEMORY BIT 7
M6 MEMORY BIT 6
n NO CYCLES
NO BYTES

APPENDIX J
AIM 65 CONNECTOR SIGNALS

TERMINAL	SIGNAL	
1	-12V	
2	GND	*
3	+5	
4	+12V	
5	GND	*
6	+24V	

*Connected together on Master Module.

Figure J-1. Terminal Board TB1 (Power) Signals

PIN	SIGNAL	PIN	SIGNAL
22		Z	
21	CA2	Y	SERIAL INPUT
20	CA1	X	
19	CB2	W	TAPE 1A
18	CB1	V	TAPE 2A
17	PB6	U	TTY PTR
16	PB5	T	TTY KYBD
15	PB7	S	TTY PTR RTN (+)
14	PA0	R	TTY KYBD RTN (+)
13	PB4	P	AUDIO OUT HI
12	PB3	N	+12V
11	PB2	M	AUDIO OUT LO
10	PB1	L	AUDIO IN
9	PB0	K	
8	PA7	J	TAPE 2B
7	PA6	H	TAPE 2B RTN
6	PA5	F	TAPE 1B
5	PA4	E	TAPE 1B RTN
4	PA1	D	R/$\overline{W}$
3	PA2	C	ø2
2	PA3	B	
1	GND	A	+5V

Connector J1 Pin Assignments (Back View)

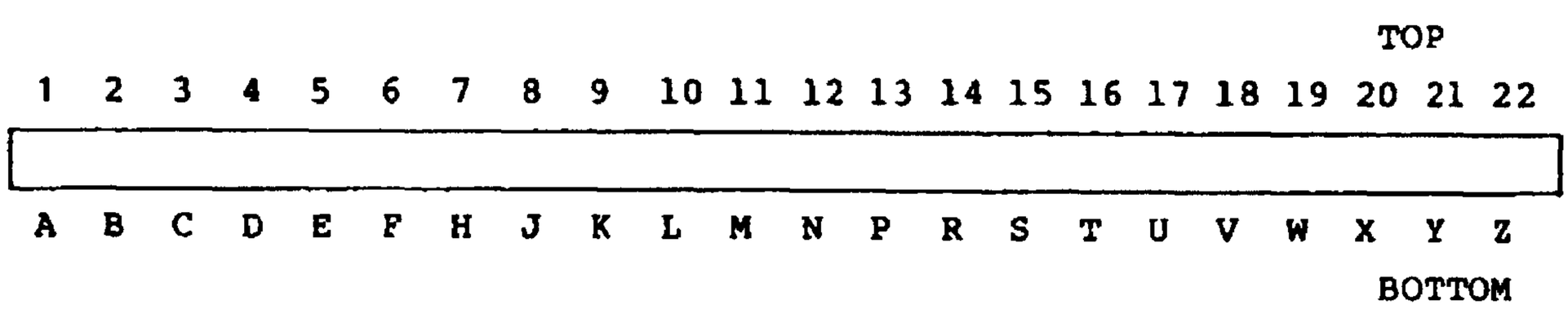


Figure J-2. Connector J1 (Application) Signals

PIN	SIGNAL
1	TE10
2	TE9
3	TE8
4	TE7
5	TE6
6	VTH
7	TE5
8	TE4
9	TE3
10	TE2
11	TE1
12	P2
13	P1
14	START
15	COMMON
16	GND
17	M+

Connector J2 Pin Assignments (Top View)

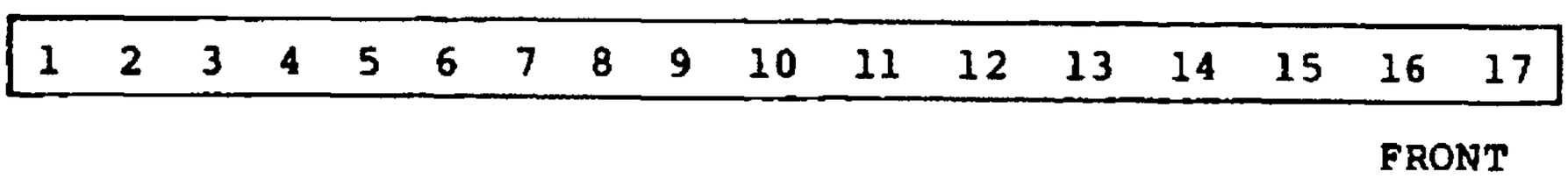


Figure J-3 Connector J3 (Printer) Signals

PIN	SIGNAL	PIN	SIGNAL
22	GND	Z	RAM R/$\overline{W}$
21	+5V	Y	$\overline{\phi 2}$
20	$\overline{CSA}$	X	TEST
19	$\overline{CS9}$	W	R/$\overline{W}$
18	$\overline{CS8}$	V	SYS R/$\overline{W}$
17	+12V	U	SYS ϕ2
16	-12V	T	A15
15	D0	S	A14
14	D1	R	A13
13	D2	P	A12
12	D3	N	A11
11	D4	M	A10
10	D5	L	A9
9	D6	K	A8
8	D7	J	A7
7	$\overline{RES}$	H	A6
6	$\overline{NMI}$	F	A5
5	S.O.	E	A4
4	$\overline{IRQ}$	D	A3
3	ϕ1	C	A2
2	RDY	B	A1
1	SYNC	A	A0

Connector J3 Pin Assignments (Back View)

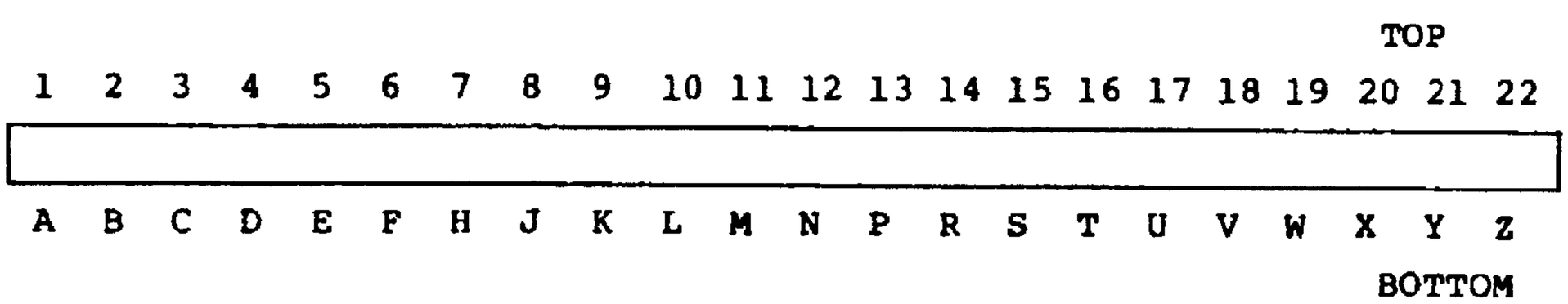
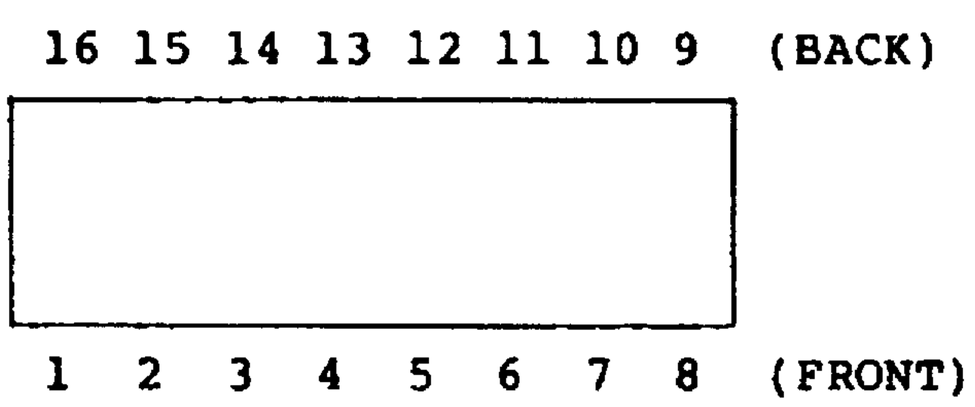


Figure J-4. Connector J3 (Expansion) Interface Signals

PIN	SIGNAL
1	KO1
2	KI1
3	KI8
4	KO7
5	KI7
6	KI2
7	KI4
8	KI3
9	KO8
10	KO2
11	KI5
12	KI6
13	KO6
14	KO5
15	KO4
16	KO3

Master Module J4 Pin Assignments (Top View):



Keyboard Module J1 Pin Assignments (Top View):

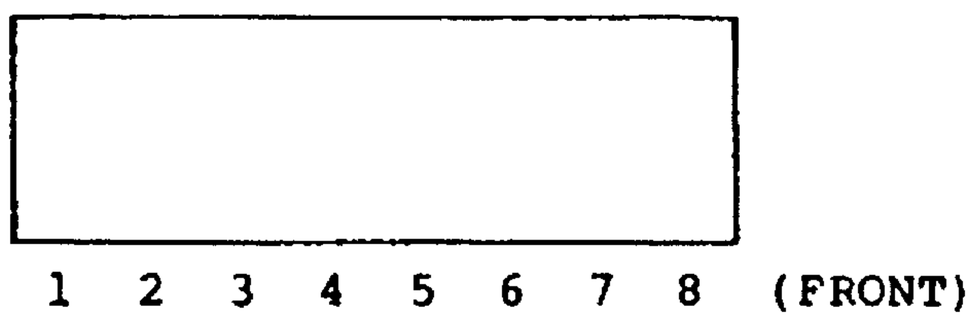


Figure J-5. Connector J4 (Keyboard) Signals

PIN	SIGNAL	PIN	SIGNAL
1	GND	20	A7
2	SYNC	21	A6
3	RDY	22	A5
4	ø1	23	A4
5	$\overline{\text{IRQ}}$	24	A3
6	S.O.	25	A2
7	$\overline{\text{NMI}}$	26	A1
8	$\overline{\text{RES}}$	27	A0
9	$\overline{\text{CSAC}}$	28	D0
10	SYS R/$\overline{\text{W}}$	29	D1
11	SYS ø2	30	D2
12	A15	31	D3
13	A14	32	D4
14	A13	33	D5
15	A12	34	D6
16	A11	35	D7
17	A10		
18	A9		
19	A8		

Connector J5 Pin Assignments (Top View):

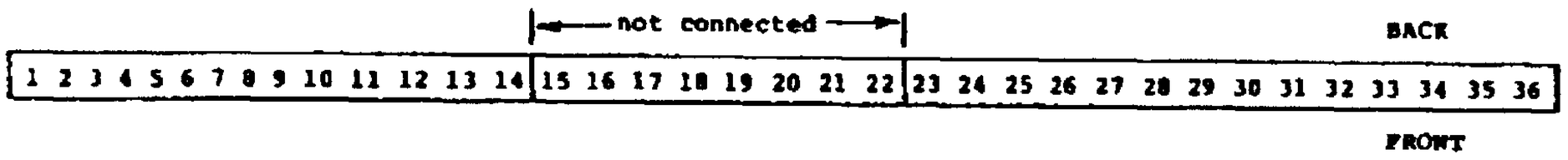


Figure J-6. Connector J5 (Display) Signals

APPENDIX K

AIM 65 USER SUB-MONITOR WITH 24-HOUR CLOCK

This appendix describes a typical user program - an AIM 65 User Sub-Monitor with 24-hour clock. This program features a Sub-Monitor, entered from the AIM 65 Monitor, which allows user-defined application subroutines to be called by typing certain user-defined keys.

Also included with the Sub-Monitor is a 24-hour clock that may be displayed continuously or displayed and printed upon command, along with a 20 character message.

An included I/O Monitor displays and prints the state of the 16 User R6522 Port A and Port B lines and the current time whenever a change is detected on any of the I/O lines. This program can easily be changed to display/print a message corresponding to the changed lines, e.g., 'PUMP MOTOR OFF' and the time of turn off.

The Sub-Monitor is listed in disassembly, source assembly and dump forms in Table K-1 to K-4. Although this program is shown as an example, it may be used as a model, or starter, for a similar user application. This program was initially prepared using the mnemonic entry capability (I command) so it reflects some operands in absolute and relative hexadecimal form rather than symbolic form. If the source code is loaded for input into the Assembler, all linkage should be symbolic to eliminate possible branching

errors. The program, as shown, is fairly long and should be loaded after you are familiar with AIM 65 operations.

The Sub-Monitor described will execute in the 1K RAM and requires addresses $0200-$03C9. In addition to the main program located in this area of RAM, a JMP instruction to address $0200 must be loaded in address $010C and the IRQ after Monitor Vector address $0368 must be loaded in address $A400.

K.1 SUB-MONITOR FUNCTIONS

The five functions programmed into the listed Sub-Monitor are:

KEY T: Allows the initial time to be entered in 24-hour format, HH:MM:SS (e.g., 15:01:15 = 1 minute 15 seconds after 3 p.m.). Enter all six digits and two colons. Automatic return to the Sub-Monitor occurs after entry of the last digit.

KEY M: Allows a 12-character message to be entered that will be displayed/printed with the time. If exactly 12 characters are entered (including spaces) the message will be displayed on the same line as the time. If less than 12 characters are entered (follow with a RETURN), the message will be displayed on a separate line, preceding the time.

KEY C: Causes continuous display of message and time. The time is updated at one second intervals. Return to the Sub-Monitor by pressing ESC.

KEY D: Display and print the message and time one time. Repeat as often as desired.

KEY I: Monitor the 16 User R6522 Port A and Port B I/O lines. Whenever any line changes state, the current state of all the lines is displayed and printed along with the time. Return to the Sub-Monitor by pressing ESC.

The Sub-Monitor is entered from the Monitor by typing F1. The Sub-Monitor prompt "%" is displayed upon entry from the Monitor and upon return from a commanded Sub-Monitor function.

When % is displayed, any of the programmed Sub-Monitor functions may be executed by typing the appropriate function key (T, M, D, C or I). To return to the AIM 65 Monitor, type ESC whenever % is displayed.

CAUTION

The Sub-Monitor 24-hour clock uses the User R6522 Timer 1 and the IRQ interrupt. Timer 1 and the IRQ interrupt processing routine will continue to run even after return to the AIM 65 monitor. This allows the

CAUTION (Cont.)

Sub-Monitor to be reentered and the current time to be displayed/printed at any time.

If you alter the memory locations used by the IRQ interrupt processing routine ($0368-$03C9) without first disabling the R6522 Timer 1 time-out generated IRQ interrupt, an error may occur. The easiest way to avoid this situation is to press RESET when you desire to run the Sub-Monitor 24-hour clock function.

Example:

```
<E>
%T
ENTER TIME: 12:30:00
%M
MESSAGE:AIM 65 TIME
%D
AIM 65 TIME 12:30:09
%D
AIM 65 TIME 12:30:14
%D
AIM 65 TIME 12:30:19
%M
MESSAGE:USER TIME
%D
USER TIME   12:30:25
%D
```

Example (Cont.)

```
USER TIME    12:30:39
12:30:41 A= 11111111
         B= 00001010
12:30:42 A= 11111111
         B= 11111111
12:30:55 A= 11111101
         B= 11111111
12:30:59 A= 11111111
         B= 11111111
12:31:01 A= 10111111
         B= 11111111
12:31:04 A= 11111111
         B= 11111111

%D
USER TIME    12:31:11
%
```

K.2 MNEMONIC ENTRY OF THE SUB-MONITOR

1. Enter the program as shown in Table K-1 using the I command.

2. Verify by running a disassembly listing using the K command.

3. Enter the Sub-Monitor by typing F1 and enter any function by typing T, M, D, C or I after % is displayed.

   If the Sub-Monitor or any of the commands do not operate correctly, compare the disassembly listing with Table K-1 and correct any detected errors.

4. Save the object code on audio cassette tape using the D command as shown in Table K-4.

Dump $A400 - $A401, $010C - $010E, and $0200 - $03C9 (unless changes have been made).

K.3 ASSEMBLY OF THE SUB-MONITOR

Assembly of the Sub-Monitor in single source file form requires the AIM 65 Assembler ROM, 4K RAM and one audio cassette recorder. With 4K RAM the source can be read into $03D0 - $0E6F then assembled for errors only and object code output to dummy device with the symbol table at $0E70 - $1000. Any detected errors during assembly can easily and quickly be corrected since the source code remains in RAM.

The assembly can also be performed in 1K RAM, by segmenting the source code into six files and assembling from tape. Any errors detected during assembly will require loading the source file into RAM, updating the source file and listing back to tape for a subsequent assembly.

The following procedure assumes 4K RAM is available:

1. Enter and initialize the Text Editor from $03D0 - $0E6F.

2. Type the source code from Table K-1 into the Text Buffer. List the source code to the printer to verify it with Table K-1. List the source also at selected checkpoints to audio cassette tape before assembly to save the source in case of accidental assembly symbol table overwrite or AIM 65 power turn-off.

3. Enter the Assembler and set the symbol table limits to $0E70 to $1000:

```
(M)
ASSEMBLER
FROM=0E70 TO=1000

IN=M
LIST?N
LIST-OUT=

OBJ?Y
OBJ-OUT=X

PASS 1

PASS 2

 ERRORS= 0000
```

If any errors are detected by the Assembler, re-enter the Text Editor using the T command and correct such errors. Then re-run the Assembler until no errors are detected as shown above.

5. After an error-free Assembler run is completed, list the source to audio cassette for intermediate saving.

6. Run the Assembler again but this time direct the source code to the display/printer and the object code output to memory:

```
(M)
ASSEMBLER
FROM=0E70 TO=1000
IN=M
LIST?Y
LIST-OUT=

OBJ?N
PASS 1

PASS 2
```

7. Execute the program by typing the commands as defined in Section K.1.

8. If the Sub-Monitor or any of the Sub-Monitor commands do not operate correctly, compare the source and/or the assembly listing with Tables K-1 and/or K-2 to find any errors. Correct any errors using the Text Editor, re-assemble, and checkout.

9. After the Sub-Monitor is correctly operating, permanently save the source and object programs on audio cassette tape. Also run a final assembly listing for future reference.

Table K-1. Sub-Monitor Source Listing

```
=<L>
/
OUT=
;*USER SUB-MONITOR*
;*CALLED BY F1 KEY*
;
;T=ENTER TIME
;   (IE: 14:34:51 -
;   24 HR. CLOCK)
;M=ENTER MESSAGE
;   (12 CHARS. TOTAL)
;D=DISPLAY TIME ONCE
;C=CONTINUOUS TIME
;I=I/O PORT MONITOR
;
;PRESS <ESC> KEY
;TO EXIT SUB-MONITOR
;
CLR =$EB44
CRLF =$E9F0
GETK2 =$EC82
OUTPUT =$E97A
RCHEK =$E907
RDRUB =$E95F
RDONEK =$ECEF
*=$92
SECSTO *=*+1
CNT *=*+1
STORA *=*+1
STORB *=*+1
*=$A000
ORB *=*+1
*=$A005
T1CH *=*+1
T1LL *=*+1
*=$A00B
ACR *=*+1
*=$A00E
IER *=*+1
ORA1 *=*+1
*=$A411
PRIFLG *=*+1
*=$010C
JMP UMON
*=$A400
 WOR INT
*=$0200
UMON JSR CRLF
LDA #$25 ;'%'
JSR OUTPUT ;PROMPT
JSR RCHEK ;<ESC>?
JSR RDRUB ;INPUT
CMP #$54 ;'T'
BNE *+5
JSR ENTIME
CMP #$4D ;'M'
BNE *+5
JSR ENTMES
CMP #$44 ;'D'
BNE *+5
JSR DISP
CMP #$43 ;'C'
BNE *+5
JSR CONT
CMP #$49 ;'I'
BNE *+5
JSR IOMON
JMP UMON
ENTIME JSR CRLF
LDX #00
MSGLP LDA TIMSG,X
JSR OUTPUT
INX
CPX #12
BNE MSGLP
LDX #00
TINLP JSR RDRUB
STA $A5,X
INX
CPX #08
BMI TINLP
LDA #$30
STA $98
LDA #$C0
STA IER
LDA #$40
STA ACR
LDA #$00, CALIBRATE
STA T1LL ; FOR
LDA #$F4 ; CORRECT
```

Table K-1. Sub-Monitor Source Listing (Cont)

```
STA TICH ; TIME
RTS
TIMSG .BYT 'ENTER
.BYT 'TIME: '
ENTMES JSR CRLF
LDX #00
MESLP LDA MESG,X
JSR OUTPUT
INX
CPX #08
BNE MESLP
LDX #00
MINLP JSR RDRUB
STA $99,X
INX
CPX #12
BMI MINLP
RTS
MESG .BYT 'MESSAGE:'
DISP JSR CRLF
LDX #00
DISPLA LDA $99,X
JSR OUTPUT
INX
CPX #20
BMI DISPLA
RTS
CONT LDA PRIFLG
STA $97
LDA #00 ;DISABLE
STA PRIFLG ;PRINTER
CONTLP JSR CLR
LDX #00
JSR DISPLA
LDA $AC
STA SECSTO
DISLP LDA $AC
CMP SECSTO
BEQ DISLP
JSR ESCHEK
BNE CONTLP
LDA $97
STA PRIFLG
RTS
ESCHEK JSR RDONEK
DEY
BMI NOESC
LDX #00
JSR GETK2
CMP #$1B ;<ESC>
BNE NOESC
LDA #00
RTS
NOESC LDA #$FF
RTS
IOMON JSR ESCHEK
BNE *+5
JMP UMON
TESTA LDA ORA1
CMP STORA
BEQ TESTB
STA STORA
JMP CHANGE
TESTB LDA ORB
CMP STORB
BEQ RETURN
STA STORB
CHANGE JSR CLR
LDX #00
STX CNT
CHLOOP LDA $A5,X
; TIME STORAGE LOC.
JSR OUTPUT
INX
CPX #03
BNE CHLOOP
LDA #$20 ;SPACE
JSR OUTPUT
LDA #$41 ; 'A'
JSR OUTPUT
LDA #$3D ; '='
JSR OUTPUT
LDA #$20 ; SPACE
JSR OUTPUT
LDX STORA
JSR ORLOOP
LDA #$20 ; SPACE
LDX #00
STX CNT
SPLOOP JSR OUTPUT
INX
CPX #03
```

Table K-1. Sub-Monitor Source Listing (Cont)

```
BNE SPLOOP ;MORE SP
LDA #$42 ; 'B'
JSR OUTPUT
LDA #$3D ; '='
JSR OUTPUT
LDA #$20 ; SPACE
JSR OUTPUT
LDX STORB
JSR ORLOOP
RETURN JMP IOMON
ORLOOP TXA
ASL A
TAX
LDA #$30
ADC #00
JSR OUTPUT
INC CNT
LDA CNT
CMP #08
BNE ORLOOP
JSR CRLF
RTS
INT PHA
NOP
TXA
PHA
INC $98
LDX #$40
CPX $98
BNE CONTIN
LDA #$30
STA $98
INC $AC
LDX #$3A
CPX $AC
BNE TIMOK
STA $AC
INC $AB
LDX #$36
CPX $AB
BNE TIMOK
STA $AB
INC $A9
LDX #$3A
CPX $A9
BNE TIMOK
STA $A9
INC $A8
LDX #$36
CPX $A8
BNE TIMOK
STA $A8
INC $A6
LDX #$3A
CPX $A6
BNE NUDAY
STA $A6
INC $A5
JMP TIMOK
NUDAY LDX #$34
CPX $A6
BNE TIMOK
LDX #$32
CPX $A5
BNE TIMOK
STA $A5
STA $A6
TIMOK NOP
NOP
NOP
CONTIN PLA
TAX
NOP
LDA $A004
PLA
RTI
.END
```

Table K-2. Sub-Monitor Assembly Listing

```
CND
ASSEMBLER
FROM=0E70 TO=1000
IN=M
LIST?Y
LIST-OUT=

OBJ?Y
OBJ-OUT=X

PASS 1

PASS 2

==0000
;*USER SUB-MONITOR*
;*CALLED BY F1 KEY*
;
;T=ENTER TIME
;   (IE: 14:34:51
;   24 HR. CLOCK)
;M=ENTER MESSAGE
;   (12 CHARS. TOTAL)
;D=DISPLAY TIME ONCE
;C=CONTINUOUS TIME
;I=I/O PORT MONITOR
;
;PRESS <ESC> KEY
;TO EXIT SUB-MONITOR
;
==0000 CLR
       =$E844
==0000 CRLF
       =$E3F0
==0000 GETK2
       =$EC82
==0000 OUTPUT
       =$E97A
==0000 RCHEK
       =$E907
==0000 RDRUB
       =$E95F
==0000 RDONEK
       =$ECEF
==0000
       *=$92
==0093 SECSTO
       *=*+1
==0094 CNT
       *=*+1
==0095 STORA
       *=*+1
==0096 STORB
       *=*+1
==0097
       *=$A000
==A000 ORB
       *=*+1
==A001
       *=$A005
==A005 T1CH
       *=*+1
==A006 T1LL
       *=*+1
==A007
       *=$A00B
==A00B ACR
       *=*+1
==A00C
       *=$A00E
==A00E IER
       *=*+1
==A00F ORA1
       *=*+1
==A010
       *=$A411
==A411 PRIFLG
       *=*+1
==A412
       *=$010C
4C0002 JMP UMON
==010F
       *=$A400
==A400
5803     WOR INT
==A402
       *=$0200
==0200 UMON
20F0E3 JSR CRLF
A925   LDA #$25
;'%'
207AE3 JSR OUTPUT
;PROMPT
2007E3 JSR RCHEK
;<ESC>?
205FE3 JSR RDRUB
;INPUT
```

Table K-2. Sub-Monitor Assembly Listing (Cont)

```
C954    CMP #$54
; 'T'
==0210
D003    BNE *+5
203402  JSR ENTIME
C94D    CMP #$4D
; 'M'
D002    BNE *+5
207502  JSR ENTMES
C944    CMP #$44
; 'D'
D003    BNE *+5
==0229
209A02  JSR DISP
C943    CMP #$43
; 'C'
D003    BNE *+5
20AA02  JSR CONT
C949    CMP #$49
; 'I'
D003    BNE *+5
20E602  JSR IOMON
==0231
4C0002  JMP UMON
==0234  ENTIME
20F0E9  JSR CRLF
A200    LDX #00
==0239  MSGLP
BD6902  LDA TIMSG,X
207AE9  JSR OUTPUT
E8      INX
E00C    CPX #12
D0F5    BNE MSGLP
A200    LDX #00
==0246  TINLP
205FE9  JSR RDRUB
95A5    STA $A5,X
E8      INX
E008    CPX #08
30F6    BMI TINLP
A930    LDA #$30
8598    STA $98
A9C0    LDA #$C0
==0256
8D0EA0  STA IER
A940    LDA #$40
8D0BA0  STA ACR
A900    LDA #$00
; CALIBRATE
8D06A0  STA T1LL
; FOR
A9F4    LDA #$F4
; CORRECT
8D05A0  STA T1CH
; TIME
==0268
60      RTS
==0269  TIMSG
454E    .BYT 'ENTER '
5449    .BYT 'TIME: '
==0275  ENTMES
20F0E9  JSR CRLF
A200    LDX #00
==027A  MESLP
BD9202  LDA MESG,X
207AE9  JSR OUTPUT
E8      INX
E008    CPX #08
D0F5    BNE MESLP
A200    LDX #00
==0287  MINLP
205FE9  JSR RDRUB
9599    STA $99,X
E8      INX
E00C    CPX #12
30F6    BMI MINLP
60      RTS
==0292  MESG
4D45    .BYT 'MESSAGE
'
==029A  DISP
20F0E9  JSR CRLF
A200    LDX #00
==029F  DISPLA
B599    LDA $99,X
207AE9  JSR OUTPUT
E8      INX
E014    CPX #20
30F6    BMI DISPLA
```

Table K-2. Sub-Monitor Assembly Listing (Cont)

```
60      RTS
==02AA CONT
AD11A4 LDA PRIFLG
8597    STA $97
A900    LDA #00
;DISABLE
8D11A4 STA PRIFLG
;PRINTER
==0284 CONTLP
2044EB JSR CLR
A200    LDX #00
209F02 JSR DISPLA
A5AC    LDA $AC
8593    STA SECSTO
==02C0 DISLP
A5AC    LDA $AC
C593    CMP SECSTO
F0FA    BEQ DISLP
20D102 JSR ESCHEK
D0E9    BNE CONTLP
A597    LDA $97
8D11A4 STA PRIFLG
==02D0
60      RTS
==02D1 ESCHEK
20EFEC JSR ROONEK
88      DEY
300C    BMI NOESC
A200    LDX #00
2082EC JSR GETK2
C91B    CMP #$1B
;<ESC>
D003    BNE NOESC
A900    LDA #00
==02E2
60      RTS
==02E3 NOESC
A9FF    LDA #$FF
60      RTS
==02E6 IOMON
20D102 JSR ESCHEK
D003    BNE *+5
4C0002 JMP UMON
==02EE TESTA
AD0FA0 LDA ORA1
C595    CMP STORA
F005    BEQ TESTB
8595    STA STORA
4C0303 JMP CHANGE
==02FA TESTB
AD00A0 LDA ORB
C596    CMP STORB
F04E    BEQ RETURN
8596    STA STORB
==0303 CHANGE
2044EB JSR CLR
A200    LDX #00
8694    STX CNT
==030A CHLOOP
B5A5    LDA $A5,X
; TIME STORAGE LOC.
207AE9 JSR OUTPUT
E8      INX
E003    CPX #03
D0F5    BNE CHLOOP
A920    LDA #$20
;SPACE
207AE9 JSR OUTPUT
A941    LDA #$41
; 'A'
==031B
207AE9 JSR OUTPUT
A93D    LDA #$3D
; '='
207AE9 JSR OUTPUT
A920    LDA #$20
; SPACE
207AE9 JSR OUTPUT
A695    LDX STORA
205203 JSR ORLOOP
==032D
A920    LDA #$20
; SPACE
A200    LDX #00
8694    STX CNT
==0333 SPLOOP
207AE9 JSR OUTPUT
E8      INX
E009    CPX #09
D0F8    BNE SPLOOP
;MORE SP
A942    LDA #$42
```

Table K-2. Sub-Monitor Assembly Listing (Cont)

```
; 'B'
207AE9 JSR OUTPUT
A930   LDA #$30
; '='
207AE9 JSR OUTPUT
==0345
A920   LDA #$20
; SPACE
207AE9 JSR OUTPUT
A696   LDX STORB
205203 JSR ORLOOP
==034F RETURN
4CE602 JMP IOMON
==0352 ORLOOP
8A     TXA
0A     ASL A
AA     TAX
A930   LDA #$30
6900   ADC #00
207AE9 JSR OUTPUT
E694   INC CNT
A594   LDA CNT
C903   CMP #03
==0362
D0EE   BNE ORLOOP
20F0E9 JSR CRLF
60     RTS
==0368 INT
48     PHA
EA     NOP
8A     TXA
48     PHA
E698   INC $98
A240   LDX #$40
E498   CPX $98
D04E   BNE CONTIN
A930   LDA #$30
8598   STA $98
==0378
E6AC   INC $AC
A23A   LDX #$3A
E4AC   CPX $AC
D03F   BNE TIMOK
85AC   STA $AC
E6AB   INC $AB
A236   LDX #$36
E4AB   CPX $AB
==0388
D025   BNE TIMOK
85AB   STA $AB
E6A9   INC $A9
A23A   LDX #$3A
E4A9   CPX $A9
D02B   BNE TIMOK
85A9   STA $A9
E6A8   INC $A8
==0398
A236   LDX #$36
E4A8   CPX $A8
D021   BNE TIMOK
85A8   STA $A8
E6A6   INC $A6
A23A   LDX #$3A
E4A6   CPX $A6
D007   BNE NUDAY
==03A8
85A6   STA $A6
E6A5   INC $A5
4C2F03 JMP TIMOK
==03AF NUDAY
A234   LDX #$34
E4A6   CPX $A6
D00A   BNE TIMOK
A232   LDX #$32
E4A5   CPX $A5
D004   BNE TIMOK
85A5   STA $A5
85A6   STA $A6
==03BF TIMOK
EA     NOP
EA     NOP
EA     NOP
==03C2 CONTIN
68     PLA
AA     TAX
EA     NOP
AD04A0 LDA $A004
68     PLA
40     RTI
       .END
 ERRORS= 0000
```

Table K-3. Sub-Monitor Disassembly Listing

```
<M>=A400 60 03 78 E0

<C>*=010C
/01
 010C 4C JMP 0200

<K>*=0200
/
 0200 20 JSR E9F0
 0203 A9 LDA #25
 0205 20 JSR E97A
 0208 20 JSR E907
 020B 20 JSR E95F
 020E C9 CMP #54
 0210 D0 BNE 0215
 0212 20 JSR 0234
 0215 C9 CMP #4D
 0217 D0 BNE 021C
 0219 20 JSR 0275
 021C C9 CMP #44
 021E D0 BNE 0223
 0220 20 JSR 029A
 0223 C9 CMP #43
 0225 D0 BNE 022A
 0227 20 JSR 02AA
 022A C9 CMP #49
 022C D0 BNE 0231
 022E 20 JSR 02E6
 0231 4C JMP 0200
 0234 20 JSR E9F0
 0237 A2 LDX #00
 0239 BD LDA 0263,X
 023C 20 JSR E97A
 023F E8 INX
 0240 E0 CPX #0C
 0242 D0 BNE 0239
 0244 A2 LDX #00
 0246 20 JSR E95F
 0249 95 STA A5,X
 024B E8 INX
 024C E0 CPX #08
 024E 30 BMI 0246
 0250 A9 LDA #30
 0252 85 STA 99
 0254 A9 LDA #C0
 0256 8D STA A00E
 0259 A9 LDA #40
 025B 8D STA A00B
 025E A9 LDA #00
 0260 8D STA A006
 0263 A9 LDA #F4
 0265 8D STA A005
 0268 60 RTS
 0269 45 EOR 4E
 026B 54 ???
 026C 45 EOR 52
 026E 20 JSR 4954
 0271 4D EOR 3A45
 0274 20 JSR F020
 0277 E9 SBC #A2
 0279 00 BRK
 027A BD LDA 0292,X
 027D 20 JSR E97A
 0280 E8 INX
 0281 E0 CPX #08
 0283 D0 BNE 027A
 0285 A2 LDX #00
 0287 20 JSR E95F
 028A 95 STA 99,X
 028C E8 INX
 028D E0 CPX #0C
 028F 30 BMI 0287
 0291 60 RTS
 0292 40 EOR 5345
 0295 53 ???
 0296 41 EOR (47,X)
 0298 45 EOR 3A
 029A 20 JSR E9F0
 029D A2 LDX #00
 029F B5 LDA 99,X
 02A1 20 JSR E97A
 02A4 E8 INX
 02A5 E0 CPX #14
 02A7 30 BMI 029F
 02A9 60 RTS
 02AA AD LDA A411
 02AD 85 STA 97
 02AF A9 LDA #00
 02B1 8D STA A411
 02B4 20 JSR E844
 02B7 A2 LDX #00
 02B9 20 JSR 029F
 02BC A5 LDA AC
```

Table K-3. Sub-Monitor Disassembly Listing (Cont)

```
02BE 85 STA 93
0200 A5 LDA AC
0202 C5 CMP 93
0204 F0 BEQ 0200
0206 20 JSR 02D1
0209 D0 BNE 0284
020B A5 LDA 97
020D 8D STA A411
0200 60 RTS
02D1 20 JSR ECEF
02D4 88 DEY
02D5 30 BMI 02E3
02D7 A2 LDX #00
02D9 20 JSR EC82
02DC C9 CMP #1B
02DE D0 BNE 02E3
02E0 A9 LDA #00
02E2 60 RTS
02E3 A9 LDA #FF
02E5 60 RTS
02E6 20 JSR 02D1
02E9 D0 BNE 02EE
02EB 4C JMP 0200
02EE AD LDA A00F
02F1 C5 CMP 95
02F3 F0 BEQ 02FA
02F5 85 STA 95
02F7 4C JMP 0301
02FA AD LDA A000
02FD C5 CMP 96
02FF F0 BEQ 034F
0301 85 STA 96
0303 20 JSR EB44
0306 A2 LDX #00
0308 86 STX 94
030A B5 LDA A5,X
030C 20 JSR E97A
030F E8 INX
0310 E0 CPX #08
0312 D0 BNE 030A
0314 A9 LDA #20
0316 20 JSR E97A
0319 A9 LDA #41
031B 20 JSR E97A
031E A9 LDA #1D
0320 20 JSR E97A
0323 A9 LDA #20
0325 20 JSR E97A
0328 A6 LDX 95
032A 20 JSR 0352
032D A9 LDA #20
032F A2 LDX #00
0331 86 STX 94
0333 20 JSR E97A
0336 E8 INX
0337 E0 CPX #09
0339 D0 BNE 0333
033B A9 LDA #42
033D 20 JSR E97A
0340 A9 LDA #1D
0342 20 JSR E97A
0345 A9 LDA #20
0347 20 JSR E97A
034A A6 LDX 96
034C 20 JSR 0352
034F 4C JMP 02E6
0352 8A TXA
0353 0A ASL  A
0354 AA TAX
0355 A9 LDA #30
0357 69 ADC #00
0359 20 JSR E97A
035C E6 INC 94
035E A5 LDA 94
0360 C9 CMP #08
0362 D0 BNE 0352
0364 20 JSR E9F0
0367 60 RTS
0368 48 PHA
0369 EA NOP
036A 8A TXA
036B 48 PHA
036C E6 INC 98
036E A2 LDX #40
0370 E4 CPX 98
0372 D0 BNE 03C2
0374 A9 LDA #30
0376 85 STA 98
0378 E6 INC AC
037A A2 LDX #3A
037C E4 CPX AC
037E D0 BNE 03BF
```

Table K-3. Sub-Monitor Disassembly Listing (Cont)

```
0380 85 STA AC
0382 E6 INC AB
0384 A2 LDX #36
0386 E4 CPX AB
0388 D0 BNE 03BF
038A 85 STA AB
038C E6 INC A9
038E A2 LDX #3A
0390 E4 CPX A9
0392 D0 BNE 03BF
0394 85 STA A9
0396 E6 INC A8
0398 A2 LDX #36
039A E4 CPX A8
039C D0 BNE 03BF
039E 85 STA A8
03A0 E6 INC A5
03A2 A2 LDX #3A
03A4 E4 CPX A5
03A6 D0 BNE 03AF
03A8 85 STA A5
03AA E6 INC A5
03AC 4C JMP 03BF
03AF A2 LDX #34
03B1 E4 CPX A6
03B3 D0 BNE 03BF
03B5 A2 LDX #32
03B7 E4 CPX A5
03B9 D0 BNE 03BF
03BB 85 STA A5
03BD 85 STA A6
03BF EA NOP
03C0 EA NOP
03C1 EA NOP
03C2 68 PLA
03C3 AA TAX
03C4 EA NOP
03C5 AD LDA A004
03C8 68 PLA
03C9 40 RTI
```

Table K-4. Sub-Monitor Dump Listing

```
<D>
FROM=A400 TO=A401
OUT=

;02A40068030111
MORE?Y
FROM=010C TO=010E

;03010C4C0002005E
MORE?Y
FROM=0200 TO=03C9

;18020020F0E9A925207
AE92007E9205FE9C954D
003203402C94DD00E08
;18021803207502C944D
003209A02C943D00320A
A02C949D00320E608FE
;180230024C000220F3E
9A2008D6902207AE9E3E
00CD0F5A200205F0A9A
;180248E995A5E8E0083
0F6A9308599A9C08D0EA
0A9408D0BA0A9000CDF
;1802608D06A0A9F48D0
5A060454E54455220544
94D453A2020F0E909FC
;180278A200ED9202207
AE9E8E008D0F5A200205
FE99599E8E00C300CD9
;180290F6604D4553534
147453A20F0E9A200B59
9207AE9E8E0143008B7
;1802A8F660AD11A4959
7A9008D11A42044EBA20
0209F02A5AC85930B9C
;1802C0A5ACC593F0FA2
0D102D0E9A5978D11A46
020EFEC88300CA20E58
;1802D8002082ECC918D
003A90060A9FF6020D10
2D0034C0002AD0F0A18
;1802F0A0C595F005859
54C0303AD00A0C596F04
E85962044EBA2000C57
;1803088694B5A5207AE
9E8E008D0F6A920207AE
9A941207AE9A93D0D4F
;180320207AE9A920207
AE9A695205203A920A20
08694207AE9E8E0088A
;18033809D0F8A942207
AE9A93D207AE9A920207
AE9A6962052034C0B44
;180350E6028A0AAAA93
05900207AE9E594A594C
908D0EE20F0E9600CF1
;18036848EA8A48E598A
240E498D04EA9308598E
6ACA23AE4ACD03F0E84
;18038085ACE6ABA236E
4ABD03585ABE6A9A23AE
4A9D02B85A9E5A80FA8
;180398A236E4A8D0218
5A8E6A6A23AE4A6D0078
5A6E6A54CBF03A20E64
;1803B034E4A6D00AA23
2E4A5D00485A585A6EAE
AEA68AAEAAD04A00EF4
;0203C868400175
MORE?N;0000170017
```

APPENDIX L
ERROR MESSAGES AND CODES

L.1 MONITOR/EDITOR ERROR MESSAGES

MEM FAIL HHHH - Memory failed to read stored data at address $HHHH.

ERROR BLK = HH - Block checksum failed during reading of audio tape at block number $HH.

ERROR HHHH - Record checksum failed during reading of audio tape at record beginning address $HHHH.

L.2 ASSEMBLER ERROR CODES

01 Undefined Symbol
02 Label Previously Defined or Forward Reference to Page 0 Symbol
03 Illegal or Missing Opcode
04 Address Not Valid
05 Accumulator Mode Not Allowed
06 Forward Reference to Page Zero
07 Ran off End of Line
08 Label Does Not Begin with Alphabetic Character
09 Label Greater Than Six Characters
10 Label or Opcode Contains Non-Alphanumeric
11 Forward Reference in Equate
12 Invalid Index — Must Be X or Y
13 Invalid Expression
14 Undefined Assembler Directive
15 Invalid Page 0 Operand
17 Relative Branch Out of Range
18 Illegal Operand Type for This Instruction
19 Out of Bounds on Indirect Addressing
20 A, X, Y, S and P are Reserved Labels
21 Program Counter Negative — Reset to 0