2 7 0 0 1 2

P R O G R A M M O V E R

Z-80 Slave Processor

Reference Manual

DECEMBER, 1982

TABLE OF CONTENTS

1. INTRODUCTION AND INSTALLATION - 1
   1.1 Introduction - 1
   1.2 Installation - 3
   1.3 Running the PROGRAMMOVER Test Programs - 5
   1.4 Jumper Options - 6

2. PROGRAMMING THE PROGRAMMOVER - 8
   2.1 PROGRAMMOVER Architecture - 8
   2.2 6502 Addressable Interface Registers - 12
   2.3 Z-80 Addressable Interface Register - 13
   2.4 Programming the Parallel Printer Port - 14
   2.5 Programming the Serial Port - 15
   2.6 Loading and Running a Z-80 Program - 18
   2.7 Using MACASM to Prepare Z-80 Programs - 19
   2.8 Using CP/M - 20

3. PRINCIPLES OF OPERATION - 21
   3.1 6502 Address Decoding - 21
   3.2 6502 Control Register - 22
   3.3 6502 Status Register - 22
   3.4 4MHz Clock Generator - 23
   3.5 Memory Timing - 24
   3.6 Memory Cycle Arbitration - 24
   3.7 Refresh Address Counter - 25
   3.8 Memory Address Multiplexor - 26
   3.9 Memory Array and Drivers - 26
   3.10 Z-80 Chip and Interface - 27
   3.11 Z-80 I/O Address Decoding - 27
   3.12 Z-80 Control Register - 27
   3.13 Z-80 Status Register - 27
   3.14 Parallel Printer Port - 27
   3.15 Serial Port - 28
   3.16 Interrupt to Z-80 - 29
   3.17 Interrupt to 6502 - 29
   3.18 Power Supply - 29

4. CONNECTOR PIN ASSIGNMENTS - 30

5. ADJUSTMENT AND TROUBLESHOOTING - 31

6. BLOCK DIAGRAM - 33

7. TIMING DIAGRAMS - 34

8. PARTS LIST - 35

9. PARTS LAYOUT - 37

10. SCHEMATIC DIAGRAMS - 38

1. INTRODUCTION AND INSTALLATION

1.1 INTRODUCTION

The PROGRAMMOVER is an advanced microprocessor/memory expansion for the MTU-130 desktop computer. In use it provides the MTU-130 user with 64K additional bytes of memory, all fully accessible by the MTU-130's 6502 processor. Also it provides a slave processor function using the Zilog Z-80 8 bit microprocessor which in turn allows the CP/M operating system to be run. This capability is of particular significance to business users because most of the business oriented microcomputer software available is written for the Z-80 and CP/M. The Z-80 can directly access the 64K of memory as well. In addition to the Z-80 and memory, a parallel printer port and a serial printer/terminal port is included on the board. These ports are directly addressed by the Z-80 making the PROGRAMMOVER useful as a hardware printer spooler. Together these DATAMOVER features allow the MTU-130 user to tap into the large and varied library of CP/M programs at low cost while enhancing the memory capacity and flexibility of the '130.

This manual fully describes the hardware of the PROGRAMMOVER board as well as programming techniques peculiar to the PROGRAMMOVER. Separate manuals describe the CP/M operating system and MTU supplied CP/M BIOS software.

1.1.1 The Slave Processor Concept

A slave processor is, literally, an additional microprocessor (computer chip) that is controlled by the master (main) system microprocessor. Slave processors may be added to a system to split up the total system workload, to execute a "foreign" operating system, or to upgrade easily to a more powerful processor. The major advantages of using a slave processor (as opposed to replacing the main processor which is not generally possible) are preservation of investment, low incremental cost, and often better performance. Of these, preservation of investment is perhaps the most important since current system hardware, software, and use experience may continue to be utilized.

A slave processor generally has direct access only to some quantity of its own memory and an interface to the main processor. The master processor has direct access to all system resources as usual. Thus the slave must access other system resources by communicating with the master. This two-step access to resources such as the disk and display can generally be made transparent to the system user, however.

1.1.2 Advantages of the Z-80 as a Slave Processor

The primary function of the PROGRAMMOVER board is to provide the MTU-130 user with the capability of running the CP/M operating system. Since most programs that run under CP/M and CP/M itself are written in assembly language, the slave processor chosen must be object code compatible with the Intel 8080. Of the available choices (8080, 8085, and Z-80), the Z-80 was chosen for its greatly enhanced instruction set, ready availability, and low cost. Other advantages include a single phase 5 volt clock, fairly simple bus interfacing (compared to the 8080 or 8085), and single 5 volt power requirements. Finally, next to the 6502, the Z-80 is the most popular processor currently used in personal computers.

1.1.3 Summary Specifications

Following is a summary of the PROGRAMMOVER specifications. Detailed descriptions of these and other points may be found elsewhere in this manual.

1. Memory Capacity - 64K bytes RAM on-board using type 4164 or equivalent RAM ICs operated with a 375NS cycle time.

2. Microprocessor - Zilog Z-80A or equivalent operated at 4MHz.

3. Architecture - Slave processor with dual-port memory plus bi-directional interrupts and 2 I/O ports. All memory is accessible from either port.

4. Z-80 Port - Organized as 64K 8 bit bytes, zero wait state access unless pre-empted by MTU-130 access or a "panic mode" refresh cycle. Guaranteed worst case access time is less than 2uS, guaranteed worst case cycle rate is .93MHz when the MTU-130 bus port is 100% utilized (1.0MHz cycle rate).

5. MTU-130 Bus Port - Organized as 64K 8 bit bytes normally appearing in bank 3 of the MTU-130 address space. Guaranteed zero wait state access. The bus port operates at 1MHz.

6. Refresh - Transparent refresh with a cycle taken after each Z-80 instruction fetch. If the Z-80 is not active or the Z-80 and 6502 memory accesses become synchronized, refresh is performed in "panic mode" with a cycle taken each 16uS.

7. Memory Contention Overhead - The Z-80 runs at 100% of theoretical 4MHz zero wait state speed when the MTU-130 bus port is not addressed, typically greater than 90% when accessed by the MTU-130 as a data bank, typically greater than 65% when accessed by the MTU-130 as a program bank, and guaranteed greater than 46% with 100% usage of the MTU-130 bus port.

8. Z-80 Control - The Z-80 may be halted and reset from the MTU-130 port under program control as well as by a general MTU-130 reset. The halt/run status of the Z-80 may be read from the MTU-130 port.

9. Interrupts to the Z-80 - The MTU-130 port may independently initiate maskable and non-maskable interrupts. The maskable interrupt uses Mode 1 (autovector to location $0038).

10. Interrupts to the 6502 - The Z-80 may set an interrupt on the MTU-130 bus and determine when it is cleared. The MTU-130 may enable or disable such interrupts from the Z-80.

11. I/O Ports - Parallel output port for a Centronics-like printer including 8 latched data bits, a Strobe, a Busy input, and a Fault input. One serial I/O port with programmable baud rate (50-19200) and full modem controls. Both ports are directly Z-80 addressable.

12. Special Features -Jumpers to select between Bank 2 and Bank 3, control-status register at $BFB6 or $BFB7, and enable/disable on reset to facilitate the use of two boards in one system or with DATAMOVER boards.

13. Power Consumption - +8 volts unregulated at .6 amp with the Z-80 running, ±12 volts at 20MA with the serial ports active.

14. Physical Size - The PROGRAMMOVER board measures 11" wide by 5" deep. Maximum component height is 0.5".

1.2 INSTALLATION

Installation of the PROGRAMMOVER involves removing the top cover of the MTU-130 keyboard unit, removing the RF interference shield, inserting the PROGRAMMOVER into an empty slot, and then reassembling the keyboard unit. After installation, a test program may be run which is included with the CP/M operating system if it was ordered. NOTE: If you already have a DATAMOVER board installed or you will be installing a DATAMOVER along with the PROGRAMMOVER, please read Section 1.4 before proceeding.

1.2.1 Disassembly Instructions

1. Unplug all cables from the MTU-130 keyboard unit and move it to an uncluttered, well lit work area.

2. Carefully stand the keyboard unit on its right end with the fan down.

3. Locate then remove two rear cover screws which are nearest the two back corners of the bottom plate and symmetrically placed. Also remove two front cover screws which are nearest the two front corners and go through long slotted holes in the bottom plate.

4. Lay the unit back down in its normal operating position but with the front overhanging the table by a couple of inches.

5. Remove the 4 screws along the front bottom surface that fasten the top cover to the bottom plate. Be sure to save the washers if present.

6. Slide the top cover forward about 1/2 inch then lift the front while keeping the back stationary so that it effectively "hinges" up about 40 degrees.

7. On the left side of the cover, locate the 2-wire cable going to the speaker and pull off the quick-disconnect lugs from the speaker terminals.

8. Lift the cover off completely and stand it upright at the right side of the base. The keyboard cable will be removed later. The fan wires may remain connected to the power supply board.

9. Remove the R-F shield which covers the front part of the boards. There are two screws on each edge of the shield.

10. Gently unplug the keyboard cable from its connector on the bottom PC board (use a small screwdriver to pry it up).

11. You should see the Monomeg CPU board (the large one in the bottom slot) and the Disk Controller board plugged into the card file. These need not be disturbed when installing the DATAMOVER.

1.2.2 Handling and Installing the PROGRAMMOVER

The PROGRAMMOVER board uses a number of MOS ICs in its construction. These are more sensitive to damage from static electricity than are TTL ICs which is why the PROGRAMMOVER is shipped in a black, conductive plastic bag. When handling the PROGRAMMOVER outside of this bag you should always hold it by the large black finned heatsink. If it must be set down temporarily, sit it on top of its bag flattened out. If it is being set on a metal table or other highly conductive surface, touch the surface first with your hand to equalize the charge between your body (and the board since you are holding it) and the surface.

The PROGRAMMOVER may be installed into any empty slot in the MTU-130's card file. Unless the boards have been moved, there will be an empty slot for large boards between the Monomeg CPU board and the Disk Controller board. There will also be two empty slots for smaller boards above the Disk Controller. Since the PROGRAMMOVER is a small (5" deep) board, it is desirable that it be installed in one of the top two slots. If this is the only expansion board in your MTU-130 (i.e., a total of 3 boards in the card file), it is desirable that it be installed in the top slot for maximum ventilation clearance.

To install the PROGRAMMOVER, remove it from its black bag and hold it by the heatsink. Before plugging it into a slot, touch one of the heatsinks on the MTU-130 power supply to discharge your body. Then slide the board into the card guides and firmly seat its edge fingers into the socket by applying pressure with the heels of both hands. There should be a definite final movement when the edge fingers seat into the socket. Make sure the board is straight in its socket and that each edge is in its white plastic card guide slot.

At the rear edge of the PROGRAMMOVER are two connectors for the I/O ports. The single-row connector is for the serial port and the double-row connector is for the parallel port. Cables to mate these connectors on one end and provide rear panel mounted standard parallel and serial connectors on the other end are available from MTU. Instructions for installing these cables are included with the cables. Alternatively you may make your own cables using the pin connection data in Section 4.

1.2.3 Reassembly Instructions

1. Find the top cover and re-attach the keyboard cable to the Monomeg CPU board.

2. Find the R-F shield and install it with its 4 screws. Insert each screw only part way until all 4 have been inserted. Then push the shield up and back as far as it will go and tighten the screws.

3. Position the top cover in the "hinged up" position it was in step 6 earlier and reattach the two speaker lugs.

4. Lower the cover and slide it back until the top rear edge is flush with the back panel. The short tabs under the rear of the top panel should hook under the lip of the back panel. It may be necessary to press the top panel flat while doing this.

5. While holding the two halves together with your hands, stand the unit on its right end as it was in step 2 of the disassembly instructions.

6. Carefully manipulate the cover as necessary to make the rear corner holes in the bottom plate match up with the internal cover mounting brackets and install the two rear corner screws (these were the ones that were removed in disassembly step 3). Also install the two front corner screws.

7. Set the unit back down with the front overhanging the tabletop edge and reinstall the 4 front cover screws that were removed in disassembly step 5.

8. Reconnect the keyboard unit to the disk drives, monitor, and power cord.

1.3 RUNNING THE PROGRAMMOVER TEST PROGRAMS

Before specifically testing the PROGRAMMOVER, obtain a known good working disk and start up the MTU-130 as usual. There should be no perceptible difference in its operation. If the system fails to function normally, turn the power off and refer to the Troubleshooting section of this manual for advice.

The PROGRAMMOVER diagnostic programs are written on the CODOS formatted diskette that is part of MTU's CP/M package. If you did not order CP/M, refer to section 1.3.1 below for some quick tests you can perform without the diagnostics. If you did order CP/M from MTU, unpack the CODOS formatted diskette at this time and follow the instructions in the MTU CP/M manual addendum for copying the diskette before proceeding.

The first diagnostic program performs a functional test of the PROGRAMMOVER's circuitry. It checks the memory access logic, run/halt/reset logic, the bidirectional interrupts, and verifies that the Z-80 can run a simple program. Assuming that a disk with the PROGRAMMOVER files is in drive 0, enter the following command to run this program:

```
PMFTEST
```

The program should print a message as each functional area of the board is tested indicating that it is OK. If a problem is found, refer to the Troubleshooting section for advice.

The second diagnostic program performs a comprehensive test of the PROGRAMMOVER's memory. Assuming that a disk with the PROGRAMMOVER files is in drive 0, enter the following command to run this test program:

```
PMMTEST
```

To make this test more meaningful, the MTU-130's 6502 processor will test half of the memory (the upper 32K) while the Z-80 simultaneously tests the other half. As each processor completes a pass, it will print a message in the upper portion of the screen indicating the pass number. If either processor detects an error, the error information will be printed in the lower portion of the screen. Both processors will stop when the lower part of the screen fills. The test algorithm consists of storing random numbers in the block to be tested and then reading them back. Every 16 passes there is a 15 second delay inserted between the store and readback phase to check the memory refresh logic. Note that the 6502 performs a test pass in about 90% of the time required by the Z-80 (the test algorithms are identical). The long term ratio however is closer to unity since a 15 second delay takes the same amount of time on either processor.

1.3.1 Quick Functional Test

First check the Status register by entering: DUMP BFB7. If the board has not been used since the last reset and the standard jumper settings have not been changed, this location should read 07 which indicates that the PROGRAMMOVER memory is enabled but the Z-80 is being held in the reset state. If 07 is not seen, try SETting BFB7 to 07 and then look again.

To briefly check the memory, enter: COPY 700 BDFF 0:3 and wait a few seconds. Now enter: COMPARE 700 BDFF 0:3 and wait a few seconds for the COMPARE command to print SAME which indicates that the PROGRAMMOVER's memory is working. Refer to the Troubleshooting section if either of these quick tests reveals a problem.

1.4 JUMPER OPTIONS

Provisions have been made for installing both a PROGRAMMOVER Z-80 board and a DATAMOVER 68000 board in the same MTU-130 system. You may in fact even have two PROGRAMMOVERS and a DATAMOVER or two DATAMOVERS and a PROGRAMMOVER in the same system for a total of four processors (including the MTU-130's 6502) all running simultaneously at essentially full speed. The 6502 can access all of the memory on each slave processor board as well as its own memory. Each slave processor however can only access the memory on its own board. Successful operation of multiple slave processors involves both hardware and software considerations.

The configuration flexibility required to accomplish the foregoing is provided by jumper pins and plugs. The pins are sets of 2 or 4 small square metal posts and the plugs can be slipped onto two adjacent posts to connect them together. For the usual case of only one slave processor, the board is shipped with the correct jumper configuration already installed and tested. When two or more slave processors are used, the jumpers on one or both will have to be changed. The recommended settings for the most common processor combinations are described in section 1.4.2 below.

1.4.1 Jumper Effects and Locations

The OFF-ON jumper is located just above U5 (see the parts layout diagram in section 6). It consists of two pairs of posts, the left pair marked OFF and the right pair marked ON. When a plug connects the ON posts together, the 64K of memory on-board the PROGRAMMOVER is enabled to be accessible by the 6502 after power-up or reset. When the OFF posts are connected, the memory is disabled after power-up or reset until the board-enable bit (bit 2) in the PROGRAMMOVER control register is set by a 6502 program. When multiple slave processors are used, only one of them should be enabled after reset to prevents conflicts on the bus. The PROGRAMMOVER is normally shipped with a plug over the ON posts.

The BANK 2/3 jumper is at the right edge of the board between U39 and U47. It consists of two pairs of posts, the left pair marked "3" and the right pair marked "2". When a plug connects the "3" pair together, the PROGRAMMOVER's memory resides in Bank 3 of the 6502's address space. When a plug connects the "2" pair together, PROGRAMMOVER memory is in Bank 2. The normal setting of this plug is "3" which matches the expectations of standard PROGRAMMOVER support software as well as other software written to utilize expanded memory such as BASIC 1.5 and MACASM. The "2" setting is provided to allow two PROGRAMMOVERS to be installed and enabled at once, the first in Bank 2 and the second in Bank 3.

The BD0/BD1 jumper is located just to the right of U2 close to the edge fingers. It consists of two pairs of posts, the top pair marked BD0 and the bottom pair marked BD1. When a plug connects the BD1 pair together, the Control and Status registers are addressed at $BFB7. When a plug connects the BD0 pair together, these registers are at $BFB6. The normal setting of this plug is BD1 which is expected by standard PROGRAMMOVER support software. The BD0 setting is provided to allow two PROGRAMMOVERS to be installed and individually controlled through the Control and Status registers. Note that with two PROGRAMMOVERS this jumper must be moved on one of them to avoid conflict among the registers; moving the BANK 2/3 jumper described above is optional.

The final jumper, which is just above U37, controls interrupts from the serial port on the PROGRAMMOVER. Since the serial port is somewhat susceptible to spurious interrupts from noise on the modem control signals, this jumper should be left off unless interrupts from the serial port will actually be used by your software. This jumper is normally absent.

1.4.2.1 One PROGRAMMOVER and One DATAMOVER

Probably the most common multiple processor system configuration will be one DATAMOVER and one PROGRAMMOVER board. Since the DATAMOVER has more memory (256K), and uses a more powerful processor (68000), it will be given "priority". Thus, the PROGRAMMOVER ON/OFF jumper should be set to the OFF position and the DATAMOVER jumpers can be left in their standard configuration. The other jumpers on the PROGRAMMOVER should be left in their standard positions. With this setup, the DATAMOVER is fully operational after power-up or Reset and is therefore immediately available to BASIC 1.5, MACASM, DMXASM, DMXMON, and other MTU DATAMOVER software. The MTU loader for CP/M is written to turn the DATAMOVER off (if present) and the PROGRAMMOVER on so there should be no problem with switching to CP/M even with a DATAMOVER present. However, if CP/M is exited with the INT key, the PROGRAMMOVER will remain enabled and the DATAMOVER disabled until either Reset is pressed or the respective control registers are written into.

1.4.2.2 Two PROGRAMMOVERs

Another potentially useful configuration is two PROGRAMMOVERs. Typically, one would be used for running CP/M and for memory expansion while the other would be used as an intelligent printer buffer and memory expansion. The first or primary PROGRAMMOVER would retain the standard jumper settings and be the one used to run CP/M. The secondary PROGRAMMOVER should have the jumpers changed so that its memory resides in Bank 2 and its control/status registers are accessed at $BFB6. The ON/OFF jumper can be left in the ON position since there will be no conflicts with both boards enabled when the memory is in different banks. Another possibility would be to leave both boards set for Bank 3 and the secondary board initially off. The printer buffer software could then be arranged so that the secondary board would be enabled only when being loaded with a print job and then disabled.

1.4.2.3 A DATAMOVER and Two PROGRAMMOVERs

This is handled similarly to 1.4.2.1 above. The DATAMOVER would be installed with the standard jumper settings and the primary PROGRAMMOVER would also use the standard settings except for the ON/OFF jumper which should be set OFF. The secondary PROGRAMMOVER would normally be set for BD0, Bank 2, and OFF.

1.4.2.4 Two DATAMOVERs and a PROGRAMMOVER

In this system, the primary DATAMOVER is installed with the standard jumper configuration. The secondary DATAMOVER is installed with the BD0/BD1 jumper moved to the BD0 position. The PROGRAMMOVER is installed with the standard settings except with ON/OFF jumper moved to OFF.

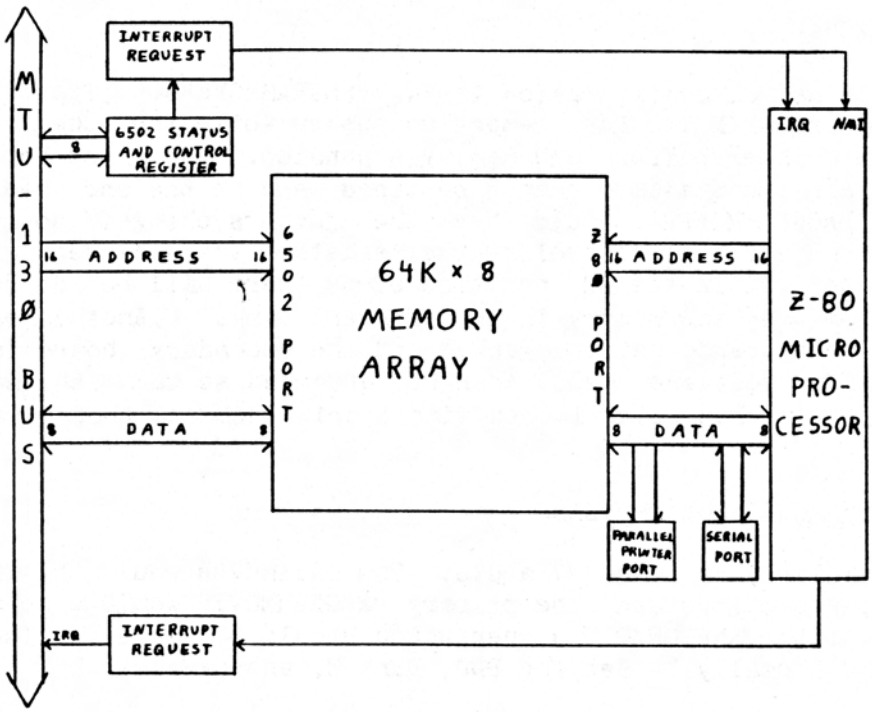
As noted in the DATAMOVER manual, any system with two DATAMOVERs needs to be treated with some care because there is no ON/OFF jumper on the DATAMOVER. This means that both DATAMOVERs will come up enabled so there should be a command in the STARTUP.J file to turn one of them off.

2. PROGRAMMING THE PROGRAMMOVER

This section deals with programming the interface between the MTU-130's 6502 microprocessor and the memory/I-O/Z-80 microprocessor on-board the PROGRAMMOVER. Programming of the Z-80 itself or use of the CP/M operating system will not be covered here. If you plan to use the PROGRAMMOVER strictly as a memory board then only the Control Register description in section 2.2.3 is applicable. If you only intend to use the PROGRAMMOVER through the CP/M operating system, then it is not necessary to be familiar with this section.

2.1 PROGRAMMOVER ARCHITECTURE

Below is an illustration of the fundamental architecture of the PROGRAMMOVER. Essentially it is a 2-port memory with a Z-80 connected to one port and the MTU-130 bus with its 6502 processor connected to the other port. The two processors may also communicate by interrupting each other. Finally, the 6502 exercises ultimate control by being able to halt and reset the Z-80.



2.1.1 Shared Memory

The majority of the board's logic implements a 2-port memory. Its size is fixed at 64k bytes. Unlike usual microprocessor memory arrays, the PROGRAMMOVER's memory has two functionally independent access ports. Each access port implements a complete random access read/write memory function and each can reach all parts of the memory array. Since the two ports refer to the same memory cells, it follows that what is written through one port can be read through the other and vice-versa. This of course is fundamental since it allows the 6502 and CODOS to load a Z-80 program and data into the memory and conversely allows the 6502 to read back the results. The relation between the ports is such that a given 6502 port byte address will refer to the same memory cells as the exact same Z-80 port byte address. Since the Z-80 uses separate I/O instructions to reach I/O ports, the on-board parallel and serial ports are accessible only by the Z-80 and do not overlap or preempt any RAM locations.

Although the two ports are functionally and logically independent they are not physically independent. What this means is that the memory cells cannot do two operations (i.e., respond to the two ports) simultaneously. Thus if one port is busy accessing the memory cells, the other will have to await its turn. In nearly all cases, the Z-80 will use its port much more than the 6502 will. In these cases the Z-80 will seldom have to wait for access and so its execution speed will closely approach the theoretical maximum. It is possible however to tie up the 6502 port and thus significantly slow down the Z-80. This would occur if a 6502 program were loaded into the PROGRMAMOVER memory and executed there (i.e., the program bank were set to 3). In such a case, the Z-80 could be slowed to as little as 1/2 of its normal speed although 2/3 is more typical. On the other hand, occasional (such as every 20uS) 6502 accesses for data (data bank set to 3) would slow the Z-80 by only a few percent.

The PROGRAMMOVER memory is dynamic and therefore must be refreshed to retain its data. Logic on-board does this automatically and need not concern the programmer or user. Refreshing can normally be done immediately after every instruction fetch while the Z-80 is decoding the instruction. If the 6502 program is running in Bank 3 (Program Bank=3) it is possible, although very unlikely, that Z-80 accesses and 6502 accesses to the PROGRAMMOVER memory may become synchronized and thus not allow normal refreshing. If this condition persists for more than 8uS, the refresh logic enters "panic mode" in which the Z-80 is forced to wait briefly for a refresh cycle. The timing design is such that the 6502 port never waits on refresh.

2.1.2 Reset and Halt of the Z-80

In addition to the MTU-130's access into PROGRAMMOVER memory, there are two addressable registers that are in the 130's I/O address space. These registers allow the 6502 to control the Z-80's "front panel". The most fundamental panel operations are Reset and Halt/run.

The Reset function does just what its name implies - it instantly halts the Z-80 program and forces the Z-80 and its interface circuitry into a known and idle state. This Reset signal is actually generated by an interface register bit. The 6502 can freely set this bit to a zero (forces and holds Z-80 reset) or set it to a one (allows the Z-80 to start and run a program). Pressing the Reset key on the MTU-130 console also forces the reset bit to a zero and thus resets the Z-80 as well as the 6502. When the Reset signal is released (6502 writes a one into the interface register bit), the Z-80 goes through a well-defined startup sequence and then starts executing its program. Thus to load and run a Z-80 program, you would write a zero into the Reset bit, load the program into PROGRAMMOVER memory, and then write a one into the Reset bit to start execution. This sequence is explained in more detail in section 2.4.

Although the Z-80 cannot be halted by the 6502 (there is no Halt input on the Z-80 chip), the Z-80 program itself can execute a HALT instruction. The 6502 can determine that this has occurred by looking at a bit in the PROGRAMMOVER Status Register. When the Z-80 has halted, it can be restarted with either the maskable (if enabled) or the non-maskable interrupt or it can be reset and started from scratch. Use of the interrupts is explained in the next section.

2.1.3 PROGRAMMOVER Interrupts

While the shared memory architecture of the PROGRAMMOVER allows highly efficient exchange of data between the 6502 and the Z-80, it is not very good for exchanging "event signals" such as "I have finished this task", or "Here's some data", or "Tell me when you're finished". While there are ways of implementing such signalling solely through "mailboxes" in memory, the PROGRAMMOVER also offers bidirectional interrupts to accomplish this.

If the 6502 wishes to signal a running Z-80 program that some more data is available, it can store a message in memory and then interrupt the Z-80 to tell it to read the message. The 6502 knows that the message was received when the Z-80 clears the interrupt. Likewise, the Z-80 can signal the 6502 that it is done by storing a message and then interrupting the 6502. It too can tell that the message was received when the 6502 clears the interrupt. The advantage of using interrupts for signaling is that both processors can be "doing something useful" rather than sitting in a loop testing the "mailbox" location.

2.1.4 Interrupts from 6502 to Z-80

The MTU-130's 6502 microprocessor can interrupt the Z-80 on two different priority levels. The non-maskable interrupt to the Z-80 is the highest possible priority and therefore cannot be locked out by the Z-80 program, just like the "non-maskable" interrupt of the 6502. The principal use of this interrupt would be in conjunction with a "cross monitor" program which would allow the user (through the 6502) to unconditionally interrupt the Z-80 and find out where it was executing and what the registers are. This is analogous to the INT key on the MTU-130 keyboard which unconditionally interrupts the 6502. The non-maskable interrupt may be set by the 6502 writing a one into an interface register bit. The Z-80 will acknowledge a non-maskable interrupt instantly even if it is executing one of the repeat or block transfer instructions (the instruction is actually resumed after the interrupt is serviced!).

The maskable interrupt to the Z-80 is just like the non-maskable interrupt except that the Z-80 program may disable it by executing a DI instruction. Thus depending on the Z-80 program, a maskable interrupt from the 6502 may not be acknowledged immediately, if ever. This is analogous to the maskable interrupt of the 6502 which can be masked off by the program. In addition, the 6502 can determine when the Z-80 acknowledges the maskable interrupt by reading an interface status register.

The maskable interrupt from the 6502 to the Z-80 is implemented in Mode 1. This means that the interrupt logic expects the Z-80 itself to supply the interrupt vector address (it will JSR to location 0038 hex). The Z-80 programmer is responsible for setting Mode 1 with an IM 1 instruction after reset and before interrupts are enabled.

The sequence for using the maskable 6502-to-Z-80 interrupt would be as follows:

1. The 6502 requests a maskable interrupt by writing to an interface register.

2. Assuming that the Z-80 maskable interrupt is enabled, the Z-80 will be interrupted and will enter the maskable interrupt service subroutine.

3. Simultaneous with entering the service routine, the interrupt is cleared. The 6502 can determine when this occurs by reading an interface status register.

4. Since the 6502 is not the only possible source of maskable interrupts, the Z-80 addressable interface status register should be checked to verify that the 6502 is the source of the interrupt and if so, enter the service subroutine for interrupts from the 6502.

5. The Z-80 interrupt service routine performs the appropriate processing such as reading a message in memory.

The 6502 only knows when the Z-80 service routine starts, not when it has been completed. If knowledge of the latter is required, it can be had by having the Z-80 service routine zero a memory location or interrupt the 6502 when it completes. The sequence for the non-maskable interrupt is similar except that there is no significant service delay and therefore there is no provision for the 6502 to determine when the non-maskable interrupt has been acknowledged.

Note that resetting the Z-80 will also clear both interrupt requests. Since resetting the MTU-130 also resets the Z-80, that too will clear both interrupt requests.

2.1.5 Interrupts from Z-80 to 6502

The Z-80 can also interrupt the 6502 but only on the maskable interrupt level. The most significant bit of the byte at Z-80 I/O address $C0 is the interrupt request flag. The Z-80 can freely set and reset it by writing (1=request interrupt) and can determine its current state by reading bit 7 at the same address. The 6502 can only reset it by writing to an interface register. The 6502 can choose to ignore this interrupt in two ways. First it may mask off all interrupts by setting its interrupt disable flag with the SEI instruction. Or it may just mask off the PROGRAMMOVER interrupt by setting an interface register bit.

The sequence for using the Z-80-to-6502 interrupt would be as follows:

1. The Z-80 requests an interrupt by writing $80 to I/O location $C0.

2. Assuming that the 6502 interrupt disable flag is off and interrupts from the PROGRAMMOVER are enabled, the 6502 will be interrupted and will go through its interrupt response sequence eventually ending up in the service routine.

3. The 6502 interrupt service routine checks all possible interrupt sources and determines that the PROGRAMMOVER is the cause of the interrupt.

4. The 6502 service routine performs the appropriate processing such as reading a message in PROGRAMMOVER memory.

5. When the 6502 service routine is finished processing the Z-80 interrupt, it resets the request by writing to an interface register.

6. The Z-80 can determine when the 6502 has acknowledged the interrupt by reading I/O location $C0 and waiting for bit 7 to go to a 0.

Note that resetting the Z-80 will also clear the interrupt request. Since resetting the MTU-130 also resets the Z-80, that too will clear the interrupt request.

2.2 6502 ADDRESSABLE INTERFACE REGISTERS

The PROGRAMMOVER has 2 interface registers that are addressable by the 6502. One is write-only and one is read-only and together they occupy one I/O address; normally $BFB7 in bank 0. This address may be changed to $BFB6 by moving a jumper as described in section 1.4. The bit assignments and functions of these registers are described in the following sections:

2.2.1 6502 Control Register $BFB7

The Control Register is a write-only register addressable by the 6502 at $BFB7. Individual bits control most of the PROGRAMMOVER's functions. This register is cleared to zeroes except for bit 2 whenever the MTU-130 is reset. The function of each bit is described below:

Bit 7 Interrupt from Z-80 acknowledge - Writing a one into this bit will clear the interrupt request from the Z-80. Writing a zero will do nothing.

Bit 6 Enable interrupts from the Z-80 - Writing a one into this bit will allow an interrupt request from the Z-80 onto the 6502's bus. Writing a zero will prevent interrupt requests from the Z-80 from affecting the 6502 bus.

Bit 5 Maskable interrupt to Z-80 - Writing a one into this bit will set the maskable interrupt request to the Z-80. Writing a zero will do nothing. Once set, this bit may be cleared only by the Z-80 acknowledging the interrupt or by resetting the Z-80.

Bit 4 Non-maskable interrupt to Z-80 - Writing a one into this bit will trigger a non-maskable interrupt in the Z-80. Writing a zero will do nothing. Since this interrupt level is non-maskable by the Z-80, it will always be acknowledged immediately. Thus a separate interrupt is generated each time a 1 is written into this bit.

Bit 3 $\overline{\text{Reset Z-80}}$ - Writing a zero into this bit will reset the Z-80 and keep it reset until a one is written. The Z-80 will perform its initialization sequence and start running when a one is written into this bit. This consists of unconditionally jumping to location zero.

Bit 2 Memory Function Enable - Writing a one to this register will enable the PROGRAMMOVER's memory to be accessed by the 6502 in Bank 3 (Bank 2 if a jumper is moved). Writing a zero will disable the 6502 access port. The setting of this bit has no effect on the Z-80 which can continue to run a program. Normally this bit is forced to a one on reset or power-up (i.e., the memory is enabled) but by moving a jumper it can be forced to a zero instead.

Bit 1 -unused-

Bit 0 -unused-

2.2.2 6502 Status Register

The Status Register is a read-only register addressable by the 6502 at $BFB7. Individual bits in this register reveal the complete status of the PROGRAMMOVER. Most of these bits are in reality readbacks from selected bits in the write-only Control register. The status register may be read as often as desired; the act of reading it does not clear or trigger anything. This register will typically read $07 after a reset unless one of the jumpers is moved. Below is listed the meaning of each Status Register bit:

Bit 7 Interrupt request from Z-80 - This bit is a one when the Z-80 is trying to interrupt the 6502. It is actually a copy of the Z-80 addressable 6502 Interrupt Request bit (see section 2.3).

Bit 6 Enable interrupts from Z-80 - This bit is a copy of Control Register bit 6.

Bit 5 Maskable interrupt request to Z-80 - This bit is a one when a Z-80 maskable interrupt is being requested. It returns to a zero when the Z-80 acknowledges the interrupt. This bit is actually a copy of Control Register bit 5.

Bit 4 Z-80 Halt signal - This bit is a zero when the Z-80 is running and a one when it is halted. This bit will be a zero while the Z-80 is in the Reset state.

Bit 3 Readback of Reset to Z-80 - This bit is a copy of Control Register bit 3.

Bit 2 Readback of Memory Enable - This bit is a copy of Control Register bit 2.

Bit 1 -undefined-

Bit 0 -undefined-

2.3 Z-80 ADDRESSABLE INTERFACE REGISTERS

The PROGRAMMOVER has several interface registers that are addressable by the Z-80. Most of these are part of the parallel and serial ports. Since the Z-80 uses special I/O instructions, the addresses of all of these registers are 8-bit Z-80 port addresses. These registers are summarized below and explained in detail in subsequent sections.

2.3.1 Z-80 Status Register (Read-only) $C0

Bit 7 Interrupt request to 6502 (1=interrupt pending)

Bit 6 Interrupt request from 6502 (1=interrupt pending)

Bit 5 Parallel printer busy (0=free, 1=busy)

Bit 4 Parallel printer error (1=error condition such as out of paper)

Bits 3-0 Unused, undefined

2.3.2 Z-80 Control Register (Write-only) $C0

Bit 7 Interrupt request to 6502 (write 1 to request, 0 to stop requesting, the 6502 will zero this bit when it responds to the interrupt)

Bits 6-0 Unused, don't care

2.3.3 Z-80 Parallel Printer Data Register (Write-only) $80

This register holds the 8 bit data for the Centronics compatible parallel printer.

2.3.4 Z-80 Parallel Printer Strobe Register (Write-only) $00

Bits 7-1 Unused, don't care

Bit 0 Strobe to the printer (write 1 to activate and 0 to deactivate the strobe)

2.3.5 Z-80 Serial Port Registers

The following registers are part of the serial I/O port. Refer to section 2.5 for bit assignments in each of the registers. The write-only registers must not be read or garbage data will be written into them.

$40	Write-only	Transmit data register
$41	Write-only	Programmed reset (data is don't care)
$42	Write-only	Command register
$43	Write-only	Control register
$44	Read-only	Received data register
$45	Read-only	Status register
$46	Read-only	Command register readback
$47	Read-only	Control register readback

2.4 PROGRAMMING THE PARALLEL PRINTER PORT

The parallel printer port is designed to be used to interface to printers having a Centronics compatible parallel interface. Not every signal defined by Centronics for their interface is implemented but those that are should allow an effective interface to any modern matrix or daisy wheel printer advertised as having a Centronics compatible interface. This is an output-only port implemented with TTL logic so its use in driving a printer is substantially different from the standard parallel port on the MTU-130. Although the electrical interface itself is standardized, printers vary widely in their programming and operational characteristics. Thus the manual for the particular printer to be driven should be consulted in conjunction with this section.

The most fundamental part of the printer port is the Data Register at $80. This is a write-only register and holds data destined for the printer. The printer does not actually recognize the data until it is "strobbed" by writing into the Strobe Register at $00. Only the most significant bit of the Strobe Register is used in the strobbing operation and it is normally kept in the zero state (it is forced to zero by Reset). To send a byte to the printer, the byte should first be written into the Data Register. Then $80 should be written into the Strobe Register immediately followed by writing $00 to the Strobe Register. Following the strobbing operation, the Data Regsiter may be changed in preparation for the next byte.

Before the printer will accept data however, it must be ready to receive it. Bit 5 of the register at $C0 will indicate the printer's readiness. If this bit is zero, then the printer will accept data. If it is a one, then the printer is busy and will not be able to accept data. The printer program should first check this bit and wait for it to be a zero before sending data to the printer. Typically the printer will go busy briefly after every byte is strobbed into it and go busy longer at the end of a line while it is actually printing. You should consult the interface timing specifications of the printer to determine the timing relationship between Strobe and Busy. In particular, if the delay between Strobe and Busy is long (greater than 5 microseconds, your program may have to delay checking Busy for the next character to allow time for the printer to respond.

Bit 4 of the register at $C0 is available for the printer to indicate an error condition such as out of paper. Not all printers or interface cables may use the error signal however. If not connected, then this bit will always be zero which indicates no error. The actual meaning of this bit being a one will depend on the particular printer and interface cable being used.

2.5 PROGRAMMING THE SERIAL PORT

The serial port on the PROGRAMMOVER is implemented using a 6551 Asynchronous Communications Adapter (ACIA) integrated circuit. Besides performing the usual transmitter and receiver functions, this chip has its own baud rate generator so the baud rate is programmable as well. In addition to these features, the most important modem control signals are implemented for applications involving a telephone data set. The serial port may be used for interface to a serial printer, direct connection to a terminal, or connection to a modem.

The first step in using the serial port is to reset the chip and then set up the various transmission parameters. Reset is accomplished by writing anything to address $41. Two registers are involved in specifying the transmission parameters and speed. The standard setting of the Command Register at address $42 is $0B which gives no parity, independent transmitter and receiver operation (no echo), no receive/transmit interrupts, and asserts the DTR (data terminal ready) and RTS (request to send) modem control signals. The value $05 is the same except that interrupts from the transmitter and receiver are enabled.

The Control Register at address $43 determines the byte format and the baud rate. The standard setting of the upper 4 bits (left hex digit) of this register is $1 which gives 8 data bits, 1 stop bit, and selects the on-chip baud rate generator. The value $9 is similar except for two stop bits which is usually required when talking to a teletype at 110 baud. The lower 4 bits (right hex digit) determines the baud rate. Typical values are $6 for 300 baud, $A for 2400 baud, and $E for 9600 baud. Settings for other rates are shown on the 6551 data sheet.

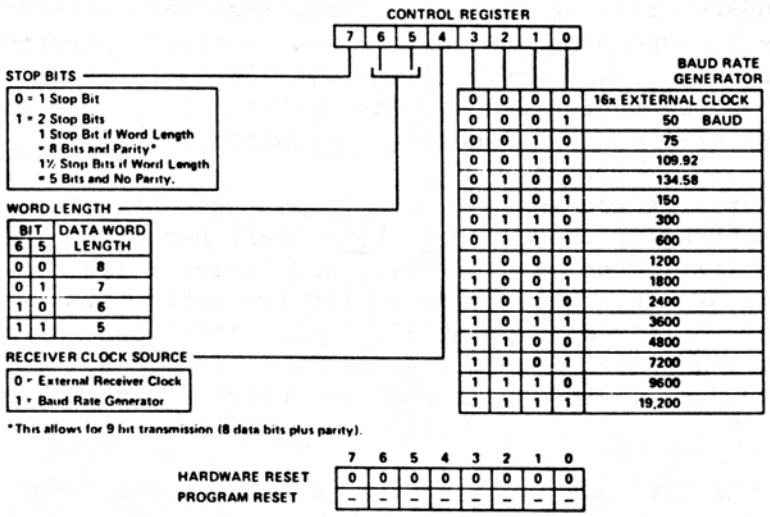
After resetting the chip and setting the Command and Control registers, the serial port is ready for transmitting and receiving. The Received Data Register is at address $44. When read, its contents represent the character last received. The Transmitted Data Register is at address $40. When written to, its contents specify the character to be sent. The chip is ready to transmit a character whenever bit 4 (mask value $10) of the Status Register at address $45 is a one. A character is transmitted simply by performing an OUT $40 which then clears bit 4 until the chip is ready for another character. A character has been received when bit 3 (mask value $08) of the Status Register is a one. Inputting the character from the Received Data Register resets bit 3 to zero until another character is received. (Caution: resetting the chip does not clear the Received Data Register therefore if the status register indicates that a character has been received immediately after reset, it should be read and discarded.)

Interrupts may originate from 4 different sources in the serial interface. One of these is generated by the transmitter when bits 3 and 2 in the Command Register are 0 and 1 respectively and a character has been transmitted. Another is generated by the receiver when Command Register bit 1 is a zero and a character has been received. The remaining two interrupts are generated whenever the Data Set Ready or Data Carrier Detect modem control signals change state and the receiver is enabled. Thus a program using the receiver must be prepared to handle these latter two interrupts or steps must be taken to prevent signal level changes on the DSR and DCD lines. Additionally, when a program has finished using the serial port, it should reset the chip (by writing into address $41), and then writing $02 into the Command Register to prevent spurious interrupts from noise. Unexplained Z-80 maskable interrupts are usually caused by a prior program failing to "idle" the serial port.

The Status Register also has a number of bits for error conditions and the modem control signals. Please refer to the 6551 data sheet below for the meaning of these bits. Note that if nothing is connected to the CTS, DSR, and DCD input pins on the serial connector, all three of these signals will appear to be <u>active</u> because of pullup resistors on the PROGRAMMOVER.

CONTROL REGISTER

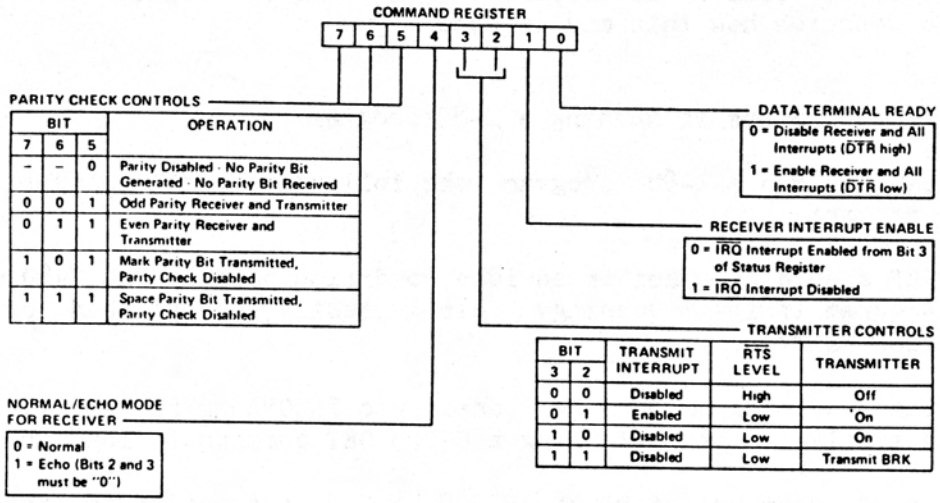
The Control Register is used to select the desired mode for the SY6551. The word length, number of stop bits, and clock controls are all determined by the Control Register.



CONTROL REGISTER WRITE $43 READ $47

COMMAND REGISTER

The Command Register is used to control Specific Transmit/Receive functions



BIT 7	BIT 6	BIT 5	OPERATION
–	–	0	Parity Disabled - No Parity Bit Generated - No Parity Bit Received
0	0	1	Odd Parity Receiver and Transmitter
0	1	1	Even Parity Receiver and Transmitter
1	0	1	Mark Parity Bit Transmitted, Parity Check Disabled
1	1	1	Space Parity Bit Transmitted, Parity Check Disabled

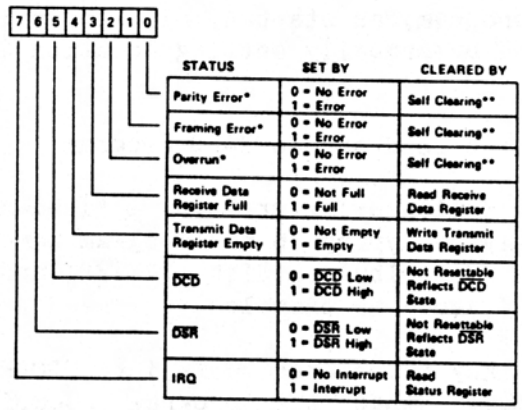
BIT 3	BIT 2	TRANSMIT INTERRUPT	RTS LEVEL	TRANSMITTER
0	0	Disabled	High	Off
0	1	Enabled	Low	On
1	0	Disabled	Low	On
1	1	Disabled	Low	Transmit BRK

	7	6	5	4	3	2	1	0
HARDWARE RESET	0	0	0	0	0	0	0	0
PROGRAM RESET	–	–	–	0	0	0	0	0

COMMAND REGISTER WRITE $42 READ $46

STATUS REGISTER

The Status Register is used to indicate to the processor the status of various SY6551 functions



STATUS	SET BY	CLEARED BY
Parity Error*	0 = No Error 1 = Error	Self Clearing**
Framing Error*	0 = No Error 1 = Error	Self Clearing**
Overrun*	0 = No Error 1 = Error	Self Clearing**
Receive Data Register Full	0 = Not Full 1 = Full	Read Receive Data Register
Transmit Data Register Empty	0 = Not Empty 1 = Empty	Write Transmit Data Register
$\overline{DCD}$	0 = $\overline{DCD}$ Low 1 = $\overline{DCD}$ High	Not Resettable Reflects $\overline{DCD}$ State
$\overline{DSR}$	0 = $\overline{DSR}$ Low 1 = $\overline{DSR}$ High	Not Resettable Reflects $\overline{DSR}$ State
IRQ	0 = No Interrupt 1 = Interrupt	Read Status Register

*NO INTERRUPT GENERATED FOR THESE CONDITIONS.
**CLEARED AUTOMATICALLY AFTER A READ OF RDR OR THE NEXT ERROR FREE RECEIPT OF DATA.

	7	6	5	4	3	2	1	0
HARDWARE RESET	0	–	–	1	0	0	0	0
PROGRAM RESET	–	–	–	–	–	0	–	–

STATUS REGISTER READ $45

2.6 LOADING AND RUNNING A Z-80 PROGRAM

Because of the great flexibility of CODOS and straightforward implementation of the PROGRAMMOVER, no specialized software is necessary for loading and running a Z-80 program. You can actually set things up so that by simply typing in the program name to CODOS, the Z-80 program will be loaded, an interface program on the 6502 will be loaded, and the two of them will start execution. From the program user's point of view, it would be no different from running a regular 6502 program! The sections below describe how this may be done.

2.6.1 Steps in Running a Z-80 Program

In order to load and run a Z-80 program, the following functions must be provided for in some manner:

1. The PROGRAMMOVER should be reset to an idle condition because the Z-80 may wipe out the new program if it is running. This is easily accomplished by writing $07 into location $BFB7.

2. The code of the Z-80 program must be loaded into PROGRAMMOVER memory. Loading the program is simply a matter of using a CODOS GET command to load it.

3. If the Z-80 program does not start at $0000, a JMP instruction to the program must be stored in locations $0000-$0002 in PROGRAMMOVER memory. These 3 locations are addressable by CODOS at 0:3 - 2 and can be loaded as part of the GET in step 2 or set manually with a SET command.

4. In most cases, you will need some regular 6502 code to communicate with the Z-80 program and thus give it virtual access to CODOS and MTU-130 I/O devices. This code can also be loaded with a GET command. It is possible to debug a Z-80 program without an interface program by simply looking at the results in memory with regular CODOS DUMP commands. Note that you can freely do this even while the Z-80 is running!

5. The last step is to start the Z-80 running. The best way is to have code in the step 4 interface program write $0F into location $BFB7 which will start the Z-80 after the interface program has started. If there is no interface program, you may start the Z-80 by manually setting location $BFB7 to $0F.

2.6.2 Setting Up a Load-and-Go Z-80 Program

The following steps may be followed for creating a "load-and-go" Z-80 program file that the user can run merely by typing in the program name. The two diagnostic programs on the CODOS CP/M distribution disk (PMFTEST and PMMTEST) were prepared in this fashion and should serve as examples.

1. Load the Z-80 program and any associated data into PROGRAMMOVER memory. For illustration we will assume that the program occupies Z-80 locations $0400-$623F and has a data table from $C000-$FFFF.

2. Set Z-80 locations $0000-$0002 with a JMP to the Z-80 program entry point. For illustration, assume that the entry point is $0400. Thus you should enter:

```
SET 0:3 C3 00 04
```

3. Load the 6502 interface program into memory. For illustration we will assume that it will start up the Z-80 itself, it resides at $700-$B3F, and its entry point is $700.

4. Set a $07 in some unused memory location. For illustration we will use a SET 0 7 to accomplish this.

5. Now that everything has been set up in memory, a single CODOS SAVE command can be used to create the load-and-go file. For the illustration case, the command would be as follows:

```
SAVE EXAMPLE 700 B3F   0=BFB7 0   0:3 2   400:3 623F   C000:3 FFFF
```

The first block saved is the 6502 interface program with its entry point of $700. The second block saved is the $07 that had been set in 6502 location $0000 in step 4 above but when the file is reloaded will be loaded into $BFB7 which will reset and enable the PROGRAMMOVER. The third block is the Z-80 reset vector and the fourth block is the Z-80 program itself. The last block is the data table for the Z-80 program.

When the program is reloaded and run by typing in EXAMPLE in response to the CODOS prompt, all 5 blocks are reloaded and then the interface program is started at its entry point of $700. It in turn starts up the Z-80 at the appropriate time by writing $07 into location $BFB7. Try doing a GETLOC on the two diagnostic programs for additional examples.

2.7 USING MACASM TO PREPARE Z-80 PROGRAMS

For preparing large Z-80 programs one would probably want to use a Z-80 or 8080 assembler in conjunction with the CP/M operating system. For doing smaller Z-80 programs without these extra cost items, the MTU standard 6502 Macro Assembler can be used quite effectively instead. Another advantage is that dual processor (6502 and Z-80) programs may be written and assembled as a unit with MACASM. For example, the two Z-80 diagnostic programs were done in this manner.

On the MACASM distribution disk (and also the CODOS compatible CP/M distribution disk) is a file called MACROS_8080.A. This file contains macro definitions for all of the 8080 instructions using Intel mnemonics. Code assembled using this macro library is 100% object compatible with the Z-80 processor on the PROGRAMMOVER. Macros are not provided for the unique extended Z-80 instructions. However they may be easily written and added. Since the macro definitions in this file will typically not appear in an assembly listing, they have been formatted to minimize the amount of memory used by MACASM to store them.

In general, you may write a Z-80 assembly language program using this macro library in the same way as if you were using an 8080 assembler. The main difference is that a period must follow every mnemonic and register designator. This is to prevent confusion with the 6502 mnemonics and register designators that MACASM is normally responsive to. For example, the 8080 instruction to copy register A into register E would be written:

```
MOV  E,A
```

if a real 8080 assembler were being used. When using MACROS_8080.A, it would be:

```
MOV. E.,A.
```

instead. The other differencs are that MACASM pseudo-ops should be used instead of Intel pseudo-ops and .DBYTE should be used to generate 16 bit constants in 8080 standard high-byte-first order. Also, MACASM uses $XXXX to denote hexadecimal constants rather than XXXXH.

The standard operating system for Z-80 based computers is CP/M. While not as powerful and convenient to use as CODOS, a large amount of software has been written to run under CP/M because it was the first widely used microcomputer operating system. In most cases, the PROGRAMMOVER board has probably been purchased specifically to run the CP/M operating system. Although full instructions for installing CP/M on the PROGRAMMOVER are supplied with the optional CP/M package, they are summarized below for your information.

PROGRAMMOVER CP/M is a full 64K CP/M with user memory (transient program area) extending from $0100 through $E3FF (56.75K) with no breaks whatsoever. The MTU-130 keyboard and display are used to emulate the operation of an ADM-3A terminal from CP/M's point of view. Blank diskettes may be formatted for CP/M's use with the included FORMAT program that runs under CP/M. CP/M diskettes may be duplicated byte-for-byte with the BACKUP_CPM program which runs under CODOS.

The PROGRAMMOVER CP/M package consists of two diskettes, one which is CODOS compatible while the other is a standard CP/M single-sided, single-density distribution disk. The CODOS disk contains all files necessary to bring up CP/M 2.2 on the MTU-130. The CP/M disk contains the CP/M operating system and the standard CP/M utilities and languages as supplied by Digital Research. For documentation, there is a brief MTU written Installation Manual and the standard Digital Research CP/M manual.

First time use of the CP/M system involves making a backup copy of the CP/M formatted distribution disk and then running a special CODOS program to load the "raw" CP/M system as supplied by Digital Research into PROGRAMMOVER memory. Following this, a special CP/M program is executed to modify CP/M itself to take advantage of the full 64K of PROGRAMMOVER memory available.

Thereafter, to start CP/M, the CODOS disk is placed in drive zero and the CODOS command: BOOT_CPM is entered. After the necessary 6502 resident software is loaded, the user will be asked to remove the CODOS disk from drive zero and insert the CP/M disk to be used. When return is pressed, CP/M itself is loaded from the CP/M disk. The BOOT_CPM command can also be placed in the STARTUP.J file of a user produced "CP/M Boot Disk" so that CP/M is automatically loaded at power-on. In either case, after the A) prompt appears, the MTU-130 becomes a CP/M machine until Reset, Int, or function key 8 is pressed.

3. PRINCIPLES OF OPERATION

This section gives a complete circuit level description of the PROGRAMMOVER hardware. Understanding of this material is not required in order to successfully program the PROGRAMMOVER although it may be useful in resolving detail operation questions that may arise. The description is intended to be sufficiently detailed so that an engineer or experienced technician can readily understand the PROGRAM-MOVER's operation for maintenance purposes.

Reference should also be made to sections 6-10 while studying this section. In particular the discussion is keyed to the schematic diagrams in section 10. The four schematic pages will be referred to by "schematic page numbers" which can be found in the lower right corner of the schematic pages. Conventional logic symbols are used in all cases. Signal names are referred to in all caps and inverse signals (active-low) are designated with overbars. Input signals generally enter a page from the left and output signals leave from the right. The physical schematic location of the source for input signals and all destinations of output signals is designated with 3 characters next to the signal name. The first character is the schematic page number, the second character gives the vertical location on the page, and the last character gives the horizontal location. Such "zone numbers" are also used to identify the location of circuitry being discussed. J1 is the board edge fingers that plug into the MTU-130 bus, J2 is the serial port pin header on the board, and J3 is the parallel port pin header.

3.1 6502 ADDRESS DECODING

Decoding of the 6502 addressable I/O register address is performed in zone B5 of schematic page 4. NAND gates U6-12 and U1 recognize when the 6502 address bus addresses the I/O registers. I/O ENAB, which participates in the gating, goes high when the MTU-130's I/O address space from $BE00-BFFF is enabled and addressed. When the jumper connected to U6-13 is in the "BD0" position, the recognized address is $BFB6. When it is in the "BD1" position (which is standard), the address is $BFB7. The final effect is that U6-12 goes low when either the Control or Status registers are addressed.

NAND gate U13-8 combines the "register addressed" signal discussed above with Phase 2 and R/$\overline{W}$ to produce a signal that goes low when the Status Register is read. NAND gate U13-6 combines "register addressed" and Phase 2 with the inverse of R/$\overline{W}$ and a signal called CNT=5 to produce a signal that goes low when the Control Register is written. CNT=5 becomes active approximately midway through Phase 2 and thus inhibits generating the write pulse until the data to be written is valid on the MTU-130 bus. This prevents glitches in the following Control Register circuit-ry.

Since the PROGRAMMOVER's memory occupies an entire bank, the memory address decoding circuitry in zone C5 of schematic page 1 is quite simple. NAND gate U47-6 combines ADDR 17 with the jumper selected true or inverse version of ADDR 16 and a Board Enable signal from the Control Register. Thus U47-6 goes low when the board is enabled and the desired bank is addressed.

3.2 6502 CONTROL REGISTER

The 6502 Control Register is in zone A3-D3 of schematic page 4. It consists of 4 flip-flops and two sets of gating for a total of 6 significant bits. Starting at the bottom, flip-flop U14-10 can be set from bit 5 of the MTU-130's data bus when clocked by the I/O write pulse described in section 3.1. When set, this flip-flop will generate a maskable interrupt to the Z-80 microprocessor. Note that this flop can only be set from the bus since the $\overline{K}$ input is tied up. It can be reset however through AND-OR-INVERT gate U22-6 which is activated during a Z-80 interrupt acknowledge sequence or when the Z-80 is reset.

Flip-flop U5-5 simply acts as a D-type latch with respect to the I/O write pulse and bus bit 3 and therefore can be both set and reset from the bus. It can also be reset when the MTU-130's System Reset signal is active. This flop is connected directly to the Reset input on the Z-80 so that the Z-80 is reset when it is zero and allowed to run when it is a one.

Next up is U5-9 which is the Board Enable flip-flop. It too acts as a D-type latch with respect to the I/O write pulse and bus bit 2. However through a jumper arrangement it can also be forced to either set or reset when the MTU-130's System Reset signal is active. Normally the "ON" jumper is installed so that the board is enabled after a system reset.

U4-9 is a D-type latch with respect to I/O write and bus bit 6. When set, it enables interrupts from the Z-80 onto the MTU-130 bus. This flip-flop is forced off by an MTU-130 system reset.

In zone 3D are two gates which generate control pulses when the 6502 Control Register is written into. U15-11 produces a pulse when bus bit 4 is a one and the I/O write pulse is present. Each pulse so produced will request a non-maskable interrupt in the Z-80. U45-6 generates a pulse when bus bit 7 is a one and the I/O write pulse is present. It will also produce an output when the Z-80 is reset. This output will reset the Z-80's interrupt request to the 6502.

3.3 6502 STATUS REGISTER

The Status Register is not really a register at all; it is simply a 6 bit tri-state buffer that gates various internal signals onto the 6502 data bus when enabled by the 6502 I/O address decoder described in section 3.1. It is enabled to drive the bus only when selected by the address decoder during read cycles. Four of the bits of the Status Register are just copies of the four Control Register flip-flops described above. Bit 4 reflects the state of the Z-80's HALT signal. Bit 7 indicates the status of the interrupt request from the Z-80. After a reset, which will clear all of these internal signals to zeroes, reading the Status Register should give all zeroes in the upper 6 bits (2-7) and will typically return ones in bits 0 and 1 as the data bus floats to a one level.

3.4 4MHz CLOCK GENERATOR

The memory timing signal generator requires a stable, accurate source of 8MHz. In addition, the Z-80 requires a symmetrical square wave at 4MHz to function properly. Circuitry in zone A5 of schematic page 1 generates the 8MHz signal by multiplying the 1MHz BUS PHASE 2 signal from the 6502 bus by 8 with a phase locked loop. U39-8 is the voltage controlled oscillator in the phase locked loop and is just a classic Schmidt trigger R-C oscillator. The 500 ohm pot (R17) determines the oscillator's free running frequency and is set for a nominal frequency of 8.0 MHz. This simple oscillator is made to act as a voltage controlled oscillator (VCO) by connecting a resistor (2.2K, R15) to the R-C node. A voltage increase at the resistor's free end will slow the oscillator while a voltage decrease will speed up the oscillator. Although the linear VCO range is only 20% or so, this is ample for locking to the fixed crystal-controlled 1MHz frequency of the MTU-130 bus.

The 8MHz oscillator drives U26, which is an 8 bit counter. The lower 3 bits of this counter driver a 1-of-8 decoder (U25) which produces several 125NS wide timing pulses with a recurrance rate of 1MHz. U25-7 is one of these and is connected to U38 which acts as a phase detector for the loop. BUS PHASE 2 is the reference input to the detector. When the loop is locked, the rising edge of BUS PHASE 2 will occur midway during the pulse from U25-7. The phase detector output (U38-13 and test point 1) therefore consists of a 60NS high pulse, 60NS low pulse, and an 875NS floating period (see the timing diagram in section 5). This waveform is averaged by low-pass filter R14 , R13, and C6 to provide the control voltage for the VCO. If the oscillator should speed up, the pulse from U25-7 will come earlier which means that the phase detector's output high time will increase and its low time will decrease thus raising the average voltage across C6. This increased voltage slows down the oscillator to maintain lock. The opposite sequence would occur if the oscillator spontaneously slowed down. The exact phase of the lock between U25-7 and BUS PHASE 2 is influenced by the setting of R17 which is normally adjusted for equal high and low pulse widths at TP1. The 8MHz waveform at U39-8 is decidedly non-symmetrical with a low time about 2.5 times greater than the high time. This non-symmetry is used to advantage in the memory cycle timing generator.

The 4MHz clock to the Z-80 is generated with a separate divide-by-two flip-flop in zone D-6. U52-9 provides a 50% duty cycle inverted clock signal to the Z-80 clock driver circuit in zone D6 of schematic page 3. This clock driver insures that the Z-80 is driven with a clock signal that swings precisely between ground and +5 volts (within .1 volt) with very fast rise and fall times. The clock driver inverts the clock polarity thus producing a true clock on the Z-80's clock pin. Note that the phase (true or inverted) of the Z-80 clock with respect to the bus PHASE 2 signal is randomly determined at power up. This is of no consequence however since the Z-80 can initiate a memory cycle on either edge of the clock depending on the cycle type.

3.5 MEMORY CYCLE TIMING

Memory cycle timing for the 64K dynamic RAM chips is performed by flip-flops U16-6, U8-5, and U8-9 in zone B3 of schematic page 1. Normally all three flip-flops are off when a memory cycle is not being executed. U6-6 will go high when any of the three possible memory cycle requestors (Z-80, 6502, or refresh) requests that a cycle begin. When such a request occurs, the first flip-flop in the chain (U16-6) will be set on the next falling edge of the non-symmetrical 8MHz clock. The output of this flop-flop becomes the row address strobe (RAS) for the memory ICs. The next rising edge of the clock will set U8-9 about 85NS later since it is now conditioned to do so by U16-6. This becomes the column address strobe (CAS) for the memory ICs. The third flip-flop (U8-5) is set on the next falling edge of the clock and thus turns on 125NS after the beginning of the cycle. On the second falling clock edge after the cycle began (i.e., 250NS later), U16-6 is conditioned to turn back off by U8-6. U8-9 then turns off 85NS later thus completing the cycle. Another cycle may be started on the next falling clock edge thus giving 125NS between cycles and a total memory cycle time of 375NS. Note that once the cycle is started, U16-6 will ignore the cycle request input from U6-6 until the entire 375NS cycle is complete.

3.6 MEMORY CYCLE ARBITRATION

On board the PROGRAMMOVER are three "users" of memory cycles which are the Z-80 microprocessor, the MTU-130 bus (6502 microprocessor), and the memory refresher. It is entirely possible for two or even all three of these users to be requesting a cycle at the same time. The needs of these users are quite different as well. For example, the Z-80 is capable of generating almost continuous requests (as frequently as every 750NS) but is able to wait for access indefinitely. The 6502 is not quite so voracious but is unable to wait on memory access at all. Fortunately, it does give several hundred nano-seconds warning before it must have its cycle. While it is running a program, the Z-80 itself requests the refresh cycles. However when the Z-80 is not running, a separate "panic mode" refresh request is generated by separate logic every 16uS. Panic mode refresh can wait for awhile but not indefinitely. These different needs plus the desire to maximize the Z-80's throughput in the presence of other requests rules out a simple priority or round-robbin arbitration scheme. What is actually used is a time dependent priority scheme where the priorities effectively shift during the 1.0uS 6502 bus cycle.

The cycle arbitration logic is in the upper half of schematic page 1. For arbitration purposes, the 1.0uS 6502 bus cycle is divided into an 750NS "6502 dominant" portion and a 250NS "Z-80 dominant" portion. The Z-80 dominant portion of the cycle occurs early in the Phase 1 portion of the cycle while the 6502 dominant portion covers the remainder of Phase 1 and all of Phase 2 (see the Monomeg CPU manual for a description of the 6502 bus cycle). These portions are distinguished by U47-8 and 2 which logically-ORs two adjacent 125NS pulse outputs from the timing decoder U25 which are phase-locked to the 6502 bus cycle.

During the 6502 dominant portion of the cycle, if the memory is addressed by the 6502 address bus (BOARD ADRD from U46-4 is high), the Z-80 and refresh are prevented from taking a cycle through U47-12. Thus the effective priority order is: 6502 highest, Z-80 and refresh never. Note that the actual beginning of a 6502 cycle may be delayed by as much as 375NS by U50-6 conditioned by U25-11 so that it will be run when the 6502 can use it. This gap is necessary so that another user's memory cycle that may have started earlier can finish before the 6502 must have its cycle.

During the Z-80 dominant portion of the cycle the effective priority order is: refresh highest, Z-80 lowest, and 6502 never. U47-8 detects the conditions necessary for allowing a Z-80 cycle to start while U6-8 detects the conditions necessary to allow a refresh cycle.

Each of the three vertically stacked flip-flops (U16-10, U24-6, and U24-10 in zonesC and D3) represent the granting of a memory cycle to a user. Note that all of them are connected "upsidedown" so the term "set" in this discussion means that the flop-flop is physically reset. Only one of these flip-flops may be set at a time, that is, during the memory cycle granted to the user it represents. U6-6 logical-ORs each of the cycle grants together and starts the timing generator running the cycle. Each flip-flop is turned off at the end of a memory cycle by the signal at U7-3.

U24-10 is set when a cycle is granted to the Z-80. It is conditioned to set by U47-8 when the Z-80 wants a cycle and is allowed to have a cycle by the arbitration logic (Z-80 dominant portion of 6502 bus cycle, no "panic mode" refresh request, and no refresh cycle in progress). The complex gating of U45-8 insures that the cycle is started at the proper time for both instruction fetch and regular Z-80 memory cycles. Note that a Z-80 interrupt acknowledge cycle will also trigger a memory cycle. The data read however is ignored since the Mode 1 interrupt protocol is expected to be used.

U24-6 is set when a refresh cycle is granted. It is conditioned to set by U6-8 when a refresh cycle is being requested by U15-6, there is no Z-80 cycle in progress or being requested, and it is the Z-80 dominent portion of the cycle. Note that a refresh cycle may be requested in two ways through OR-NOT gate U15-6. First, it may be requested by the Z-80 ($\overline{\text{Z-80 RFSH}}$) which will normally occur after every Z-80 instruction fetch. Second, it may be requested from 16uS timing counter U26-8 through inverter U21-10. This refresh timing counter is always reset when a refresh cycle is granted. As long as the the Z-80 requests and is granted a refresh before the 16uS timeout interval, the Z-80 will run at full speed with no wait states. If however the Z-80 is not running or the Z-80 program synchronizes itself with the 6502 so that all Z-80 refresh requests and pre-empted by 6502 cycles, the counter will timeout and request a "panic mode" refresh cycle.

U16-10 is set when a 6502 cycle is granted. It is conditioned to set by U50-6 when BOARD ADRD is true and the right timing slot in the 6502 bus cycle, as determined by U25-11, arrives shortly after BUS PHASE 2 becomes true. The arbitration logic has already insured that no other cycle user can be starting or still running a memory cycle at this time.

3.7 REFRESH ADDRESS COUNTER

The refresh address counter is in zone A6 and C6 of schematic page 2. It consists of two 4 bit ripple counters, both inside U12. At the end of a refresh cycle, the counter is incremented by one. The fact that a ripple counter is used does not cause problems because it is incremented at the end of the refresh cycle and therefore has ample time to settle before the next cycle. Since a refresh cycle occurs at least every 16 microseconds (more frequently if the Z-80 is running), the counter will count through 128 states every 2.048 milliseconds or through 256 states every 4.096MS. Thus either 128 cycle or 256 refresh cycle memory ICs may be accommodated. The refresh counter inside the Z-80 could not be used because it is non-functional while the Z-80 is reset.

3.8 MEMORY ADDRESS MULTIPLEXOR

The memory address multiplexor must accept addresses from three different sources and combine them into 8 multiplexed lines for the memory ICs. This in effect defines 6 sources of addresses: high Z-80, high 6502, high refresh, low Z-80, low 6502, and low refresh. The low halves of the address words become row addresses in the RAM chips while the high halves become column addresses. Since the refresh column addresses are not significant, there need be only 5 actual inputs to the memory address multiplexor. This is accomplished with a 2-input multiplexor driving one of the inputs of a 4-input multiplexor.

The address multiplexor is on the left half of schematic page 2. The 2-input "pre multiplexor" is U11 and U19 which are quad 2-input multiplexor ICs. This multiplexor selects between the lower eight Z-80 address bits and the output of the refresh addresss counter. The Z-80 address is normally selected except during an actual refresh cycle which avoids the need for address setup time when a Z-80 cycle is started.

The 4-input main multiplexor is a set of four dual 4-input multiplexor ICs (U9, 10, 17, and 18). The SEL 0 input selects between the lower and the upper half of addresses under the control of RAS RAW which is the row address clock for the memory ICs. The delay network consisting of Inverters U2-10, U2-8, R1, and C5 insure adequate hold time of the row address before switching to the column address during a memory cycle. The SEL 1 input selects between 6502 address and Z-80/refresh address. Again, the Z-80 address is normally selected except during an actual 6502 cycle to allow the Z-80 cycle to start as early as possible. The output of the address multiplexor has sufficient capacity to drive the eight 64K RAM chips directly.

3.9 MEMORY ARRAY AND DRIVERS

Schematic page 2 also contains the memory array and associated drivers. The array itself is extremely simple consisting of a single row of eight 64K dynamic RAM ICs for a total of 64K bytes.

The Row Address Strobe ($\overline{RAS}$) of the memory ICs is driven by inverter U21-12 which will be active whenever any memory cycle is granted. The Column Address Strobe ($\overline{CAS}$) of the memory ICs is driven by NAND gate U50-8. $\overline{CAS}$ is driven only on non-refresh memory cycles to minimize power consumption during refresh cycles which can become rather frequent (about 30% of all cycles) when the Z-80 is running.

In zone A3 is the logic for driving the Write Enable ($\overline{WE}$) signal to the memory ICs. For 6502 cycles, WE is generated during RAS time when LOC R/W is calling for a write (high). For Z-80 cycles, the logic is more complex. Since the actual $\overline{WR}$ output of the Z-80 chip occurs too late to be of use, the absence of the $\overline{RD}$ output is taken to indicate that the Z-80 will be requesting a write. This is indeed a valid indication except during interrupt acknowledge cycles which, as mentioned earlier, will cause a memory cycle to be run. AND gate U7-11 prevents these interrupt acknowledge cycles (which pulse neither RD nor WR) from writing into the memory. Fortunately the M1 (instruction fetch) signal is sufficient to inhibit these spurious writes because instruction fetches are always read cycles.

3.10 Z-80 CHIP AND INTERFACE

The Z-80 CPU chip itself is in the left portion of schematic page 3. It receives the 4MHz clock through a discrete component clock driver circuit (see section 3.4 for a description). RESET is driven directly by 6502 Control Register bit 3. The interrupt request inputs, NMI and IRQ are described in section 3.16. The WAIT input is driven high (no wait) when the Z-80 is granted a memory cycle or when an I/O cycle is requested. This means that the Z-80 will wait indefinitely for a memory cycle to be granted to it but I/O cycles are run with never any waits. The signals for bus request and grant (BUSREQ and BG) are not used.

The Z-80's 16 address lines are connected directly to the memory address multiplexor described in section 3.8. Note that the Z-80 will float its address bus while it is reset. The 8 data bus bits are connected to the memory data bus through bi-directional bus buffer U40. They could not be connected directly because the Z-80 may drive the bus while waiting for memory access thus interfering with a 6502 cycle that may be in progress. The bus request/grant logic built into the Z-80 reacts too slowly to be of use in this case. The other control signals (RD, MREQ, RFSH, M1, and HALT) are connected to pull-up resistors to prevent spurious operations when the Z-80 is reset and floating these lines.

3.11 Z-80 I/O ADDRESS DECODE

Unlike the 6502, the Z-80 has a special set of I/O instructions whose address fields refer to a 256 byte I/O address space separate from the 64K memory address space. During I/O instructions, the Z-80 generates the I/O port address on address lines A0-A7. Except for the serial port, dual 1-of-4 decoder U41 in zone A4 of schematic page 3 decodes the two most significant port address bits to determine which of several Z-80 addressable I/O functions have been addressed. The upper decoder is activated during I/O read (input) cycles while the lower decoder is activated during I/O write (output) cycles. The decoder outputs are clean read enables or write pulses that can be used directly by subsequent circuitry. The chip select inputs on the serial port chip (described in section 3.15) are sufficient to decode its own unique set of I/O addresses.

3.12 Z-80 CONTROL REGISTER

The Z-80 Control Register consists of a single flip-flop (U44-5) in zone A3 of schematic page 3. This flip-flop is actually the interrupt request to the 6502. The clock input is driven by the write pulse associated with port address $C0 as described in section 3.11 above. The D input is connected to bit 7 of the data bus. The Z-80 program can either set or reset this flip-flop at will by performing OUT $C0 instructions with bit 7 of the A register determining the new state. The flop can also be reset by the 6502 INTACK signal.

3.13 Z-80 STATUS REGISTER

The Z-80 status register consists of a 4 bit tri-state buffer (U38-1) in zone B3 of schematic page 3. This is an inverting buffer which drives data bus lines D4-D7 when enabled by input enable associated with port $C0 as described in section 3.11 above. The most significant bit of the status register is the state of the control register flip-flop described in section 3.12 above. Bit 6 indicates that the 6502 is trying to interrupt the Z-80. The other two bits are associated with the parallel printer port described in section 3.14 below.

3.14 PARALLEL PRINTER PORT

The parallel printer circuitry is in zone B3 of schematic page 3. The port consists of 8 bits of output-only data, a strobe output, and two bits of status input. With a suitable cable and programming, this port can drive nearly any printer advertised as having a "Centronics compatible parallel interface". The interface is designed to drive printers with "low power" line terminations of not less than 470 ohms to +5 volts.

The 8-bit data register is U37 which is equipped with internal high current drivers on its outputs. The 8 data inputs are tied to the Z-80's data bus and the clock input is driven from the write pulse associated with I/O port $80. Since the register is positive edge triggered, the data is not latched until the trailing edge of the clock. Note that the data register output is not inverted; if inverted data is required by the printer, the driver software will have to do the inversion with an XRI $FF instruction before it is output. Reset has no effect on the contents of the data register.

The strobe output consists of a single flip-flop (U44-9) and driver (U46-12). The flip-flop clock input is connected to the write pulse associated with port $00 and the D input is connected to bit 0 of the data bus. The strobe output is inverted so writing in a one will produce a low-going strobe signal to the printer. The strobe pulse width is determined by software. System reset will force the strobe to its inactive (high) state.

Two status lines from the printer are tied to inputs on the status register (U38-2 and 4). One of these is always the printer's busy signal while the other can be used to signal an error condition such as out-of-paper. The status register inverts these signals so a low-true signal from the printer (which is standard) will be read as a one in the status register.

3.15 SERIAL PORT

The serial interface is in zone C3 and D3 of schematic page 3. At its heart is a type 6551 Asynchronous Communications Interface Adapter with internal programmable baud rate generator. This is the same chip type as used in the MTU-130's own standard serial port. The baud rate generator requires a 1.84MHz input frequency. Since this is not simply related to any other frequency used in the Programmover, a separate oscillator is used. U20-2 in conjunction with ceramic resonator Y1, R18, and C34 makes a 3.68MHz oscillator. Dual capacitor C4 is required for oscillation and also temperature compensates the ceramic resonator. Resonators (or quartz crystals) for the 3.68MHz frequency are much smaller and less expensive than would be required for the 1.84MHz frequency. Flip-flop U4-2 divides the 3.68MHz oscillator frequency by 2 to produce the required 1.84MHz clock.

Standard serial interface logic levels are -12 volts for a logic 1 and +12 volts for a logic zero. Line receiver and level shifter U42 receives the serial input data, CTS, DSR, and DCD signals and converts then to the TTL compatible levels required by the 6551. R4-R6 connected to +12 volts bias the receiver inputs so that if they are not driven by external signals, they will be "pulled" up to true levels, even in the presence of up to 10 volts of noise. U43 translates the TTL level data, RTS, and DTR signals generated by the 6551 into high voltage ($\pm$12 volts) bipolar signals for the interface. U43 is actually a quad op-amp chosen for its low power consumption and controlled risetime characteristics. R11 and R12-4 (part of an R-pak) set a +2.5 volt input threshold as the op-amp acts like a comparator. R7-R9 set an output impedance of 330 ohms.

Since the 6551 was designed to be connected to a 6502 style bus, some tricks were necessary to get it to run successfully on the internal Z-80 bus. The problem arises because the Z-80 has no bus signal equivalent to Phase 2 and uses separate read and write strobes rather than a single read/write line. Fortunately the 6551 does not require a constant clock on its Phase 2 input; instead it simply performs a chip-select function. AND gate U51-3 detects the existence of an I/O cycle and drives the 6551's PH2 input. M1 is factored in to prevent the serial port from erroneously responding to interrupt acknowledge cycles.

The real chip selects (CS0 and CS1) are connected to A6 and A7 so that the 6551 responds to I/O port addresses $40-$7F. A0 and A1 selects which of the 4 internal registers are selected. A3 performs as the R/W line for the 6551 since there is no other suitable signal with both setup and hold time with respect to the signal used for PH2. Thus from the programmer's point of view, the 4 read/write registers of the 6551 are split into 4 read-only registers at $44-$47 and 4 write-only registers at $40-$43. Trying to read one of the write-only registers will actually cause it to be written into. The port addresses from $48-$7F are not unique and will all map back into $40-$47.

3.16 INTERRUPTS TO Z-80

The PROGRAMMOVER provides for three interrupts to the Z-80; one on the non-maskable level and 2 on the maskable level. The non-maskable interrupt request is just a pulse that is generated whenever the 6502 writes into the 6502 Control Register with bit 4=1. The Z-80 will perform the non-maskable interrupt sequence (a JSR to location $0038) whenever this pulse is generated. See section 3.2 for a description of the 6502 Control Register circuitry that generates this pulse.

The two sources of maskable interrupts are wire-ored together on the IRQ input to the Z-80. One of these is a flip-flop in the 6502 Control register connected through open-collector gate U23-11 in zone A2 of page 4. This flip-flop is set under 6502 program control and is reset by a Z-80 interrupt acknowledge bus cycle or by a Z-80 reset. The Z-80 program identifies the 6502 interrupt by reading the Z-80 Status Register and testing bit 6. See section 3.2 for a description of the 6502 Control Register.

The other interrupt may be generated by the serial I/O port provided that the Serial IRQ Enable jumper is installed. Z-80 programming and external serial port signals control the setting and resetting of this interrupt source. The Z-80 program identifies the serial port interrupt by testing bit 7 of the serial port status register at $45.

3.17 INTERRUPT TO 6502

The PROGRAMMOVER can generate a maskable interrupt to the 6502 by writing a one into the Z-80 Control Register described in section 3.12. The interrupt request flip-flop is U44-5 in zone A3 of schematic page 3. Open-collector NAND gate U23 drives the IRQ line on the MTU-130 bus if this flip-flop is set and the 6502 has enabled interrupts from the PROGRAMMOVER (6502 INTEN is true) by setting bit 6 in the 6502 Control Register. When the 6502 program acknowledges the PROGRAMMOVER interrupt, 6502 INTACK is generated which resets this flip-flop.

3.18 POWER SUPPLY

The power supply regulator is in zone D5 of schematic page 4. Unregulated power at approximately +8 volts from the MTU-130 bus is regulated to +5 volts by an LM340T5 and 4 watt heatsink. C8 prevents oscillation of the regulator. Positive and negative 12 volts for the serial interface is taken directly from the bus.

4. CONNECTOR PIN ASSIGNMENTS

The PROGRAMMOVER has 3 connectors on-board designated J1-J3. J1 is the 44 pin edge finger bus connector. J2 is a 10 pin single row pin header for the serial interface port. J3 is a 26 pin double row pin header for the parallel interface port. The pin assignments for each of these is given below.

4.1 J1 - MTU-130 BUS CONNECTOR

This is the MTU-130 bus connector. It is a set of double-sided edge fingers numbered from 1-22 on the component side and A-Z on the solder side (some letters are skipped). It is directly compatible with the MTU-130 bus and with minor differences, the KIM-1, SYM-1, and AIM-65 busses.

PINS	SIGNALS	PINS	SIGNALS
1	I/O SELECT	A	ADDRESS BUS 0
2	ADDRESS BUS 17	B	ADDRESS BUS 1
3	ADDRESS BUS 16	C	ADDRESS BUS 2
4	IRQ	D	ADDRESS BUS 3
5	--NC--	E	ADDRESS BUS 4
6	--NC--	F	ADDRESS BUS 5
7	RESET	H	ADDRESS BUS 6
8	DATA BUS 7	J	ADDRESS BUS 7
9	DATA BUS 6	K	ADDRESS BUS 8
10	DATA BUS 5	L	ADDRESS BUS 9
11	DATA BUS 4	M	ADDRESS BUS 10
12	DATA BUS 3	N	ADDRESS BUS 11
13	DATA BUS 2	P	ADDRESS BUS 12
14	DATA BUS 1	R	ADDRESS BUS 13
15	DATA BUS 0	S	ADDRESS BUS 14
16	-12 VOLTS REGULATED	T	ADDRESS BUS 15
17	+12 VOLTS REGULATED	U	PHASE 2
18	+8 VOLTS UNREGULATED	V	READ/WRITE
19	--NC--	W	--NC--
20	--NC--	X	--NC--
21	--NC--	Y	PHASE 2
22	GROUND	Z	--NC--

4.2 J2 - SERIAL I/O PORT CONNECTOR

This is a single row pin header. Pin 1 is marked in etch on the solder side.

PINS	SIGNALS	PINS	SIGNALS
1	Clear To Send	6	Data Terminal Ready
2	Data Carrier Detect	7	Request To Send
3	Data Set Ready	8	--polarize--
4	Receive Data	9	--unused--
5	Transmit Data	10	GROUND

4.3 J3 - PARALLEL PRINTER PORT CONNECTOR

This is a dual row pin header with the even numbered pins in one row and the odd numbered ones in the other row. Pin 1 is marked in etch on the solder side.

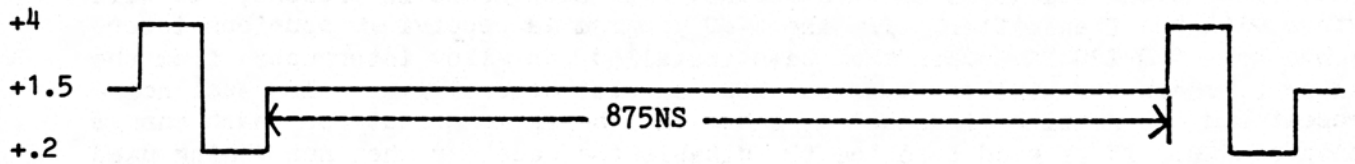
PINS	SIGNALS	PINS	SIGNALS
1	--unused--	14	--unused--
2	GROUND	15	--unused--
3	--unused--	16	--unused--
4	--unused--	17	Data Bit 1
5	--unused--	18	Data Bit 0
6	--unused--	19	Data Bit 3
7	--unused--	20	Data Bit 2
8	Printer Error	21	Data Bit 5
9	--unused--	22	Data Bit 4
10	--unused--	23	Data Bit 7
11	--unused--	24	Data Bit 6
12	--unused--	25	Printer Strobe
13	--unused--	26	Printer Ready

5. ADJUSTMENT AND TROUBLESHOOTING

This section describes troubleshooting techniques and one adjustment on the PROGRAMMOVER board. It is strongly suggested that the adjustment be checked and that the suggestions given be followed prior to contacting MTU about board failures.

5.1 ADJUSTMENT

In order for the PROGRAMMOVER to function properly, the phase-locked loop in the timing generator must be properly adjusted. If the Control and Status registers seem to function normally but reading PROGRAMMOVER memory from the 6502 gives inconsistent results, it is possible that the factory adjustment has been disturbed or has drifted. In particular if U39 or the 5 volt regulator is replaced or the PROGRAMMOVER is operated from externally supplied regulated 5 volt power this adjustment should be checked. To check the PLL adjustment, obtain an oscilloscope, set it for .5uS per divison and look at the signal on test point 1. When in adjustment, this should be a clean "doublet pulse" that rests at approximately +1.5 volts for 875NS, pulses up to about +4 volts for about 60NS, immediately pulses down to +.2 volts for 60NS, and then returns to +1.5 as shown below:



The high and low portions should be equal in width and the repetition frequency should be exactly 1.0MHz. If the waveform is not as shown, remove the seal from R17 and slowly rotate it until synchronization is achieved as evidenced by a stable pattern. Further rotate R17 until the high and low parts of the waveform are equal in duration. Careful "eyeballing" of the waveform is adequate for the adjustment as it is not overly critical.

If the PROGRAMMOVER has never functioned in your system, you should first remove it, thoroughly clean the edge fingers with Scotchbrite or a soft pink pencil eraser, and then try again. If still no luck, try operating it in a different bus slot. If proper operation cannot be obtained, then the board may have been damaged in handling or there is an intermittent connection somewhere. Try reseating each of the ICs into their sockets before continuing.

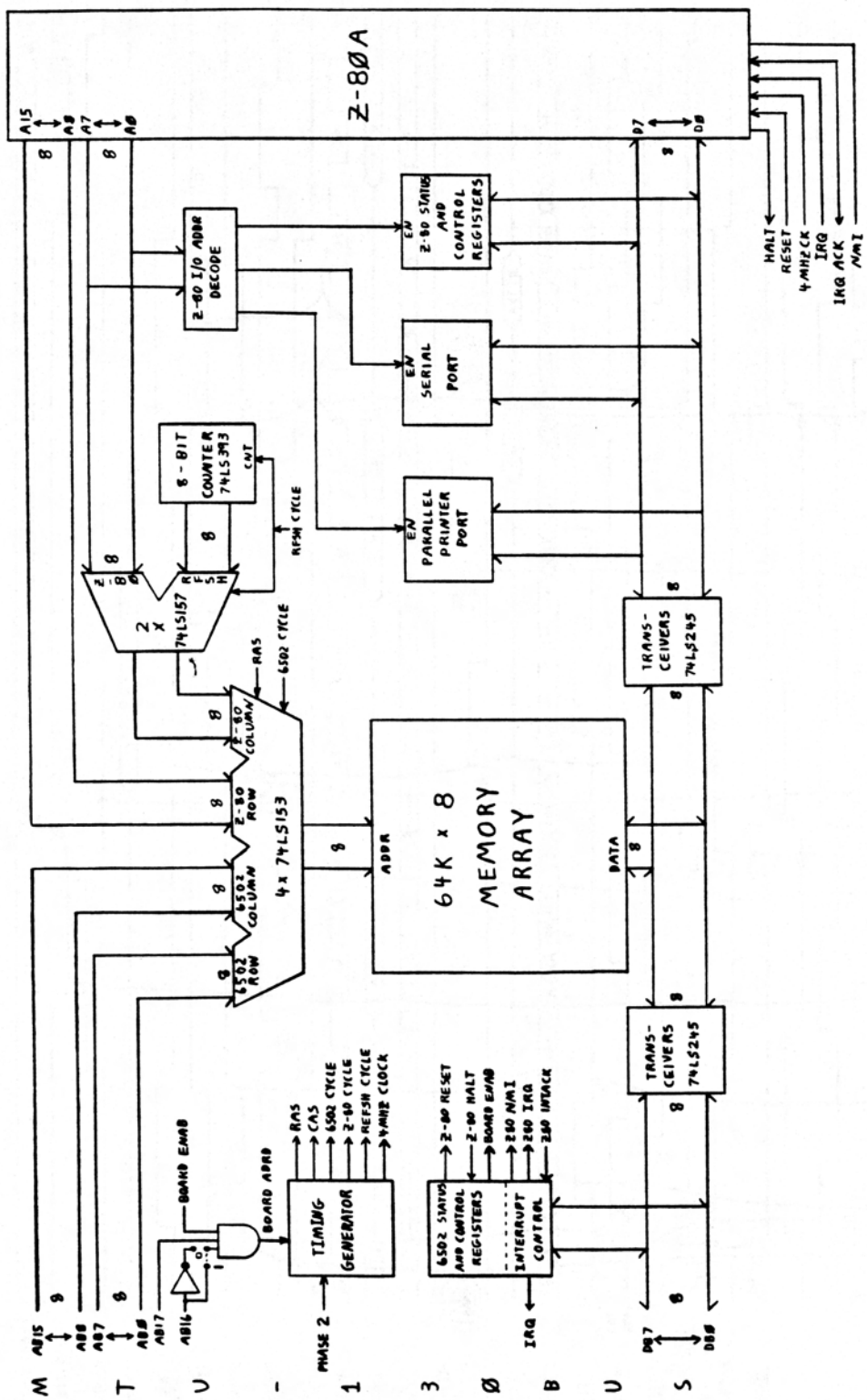
If the PROGRAMMOVER prevents the rest of the system from functioning when it is installed, the first step is to verify that proper power is reaching the circuitry. Install the PROGRAMMOVER in the top card file slot so it can be reached, disconnect the speaker so the keyboard can be moved out of the way, and turn the MTU-130 power on. With a voltmeter or preferably calibrated oscilloscope, connect the ground lead to the screw holding the regulator to its heatsink. Look at the voltage on U8-14 (a 74LS74) which is just to the right of the heatsink. It should read between +4.8 and +5.2 volts and be completely free of ripple. If it reads low or has some ripple, look at the input to the regulator. This should read between about +8.5 and 9.5 volts with just the Monomeg, Disk Controller, and PROGRAMMOVER boards installed (about 0.5 volts lower if a DATAMOVER is also installed). The ripple frequency should be 120Hz (8.3MS between peaks). If the voltage is low (less than +8 on a voltmeter) or the ripple is 60Hz, turn the power off and check for a good connection between the power transformer secondary leads and the 10 pin white nylon connector in the MTU-130 power supply.

If the power seems OK, then check the adjustment in section 5.1. Double-check that the oscillator has not been set to 4MHz or some other sub-harmonic of the correct 8MHz frequency. If the oscillator is functioning, look at U26-8. This should be a positive-going pulse every 16uS. Its presence indicates that memory refresh cycles are being requested and granted properly. If it is a 32uS square wave or has an erratic frequency or is absent, there is a problem in the memory cycle timing generator or the arbitration circuit. If the Programmover keeps the 130 from running and power problems have been ruled out as a possible cause (see above), remove buffers U3 and U48. If this allows the 6502 to run normally, then there is a problem in the 6502 address decoding circuitry.

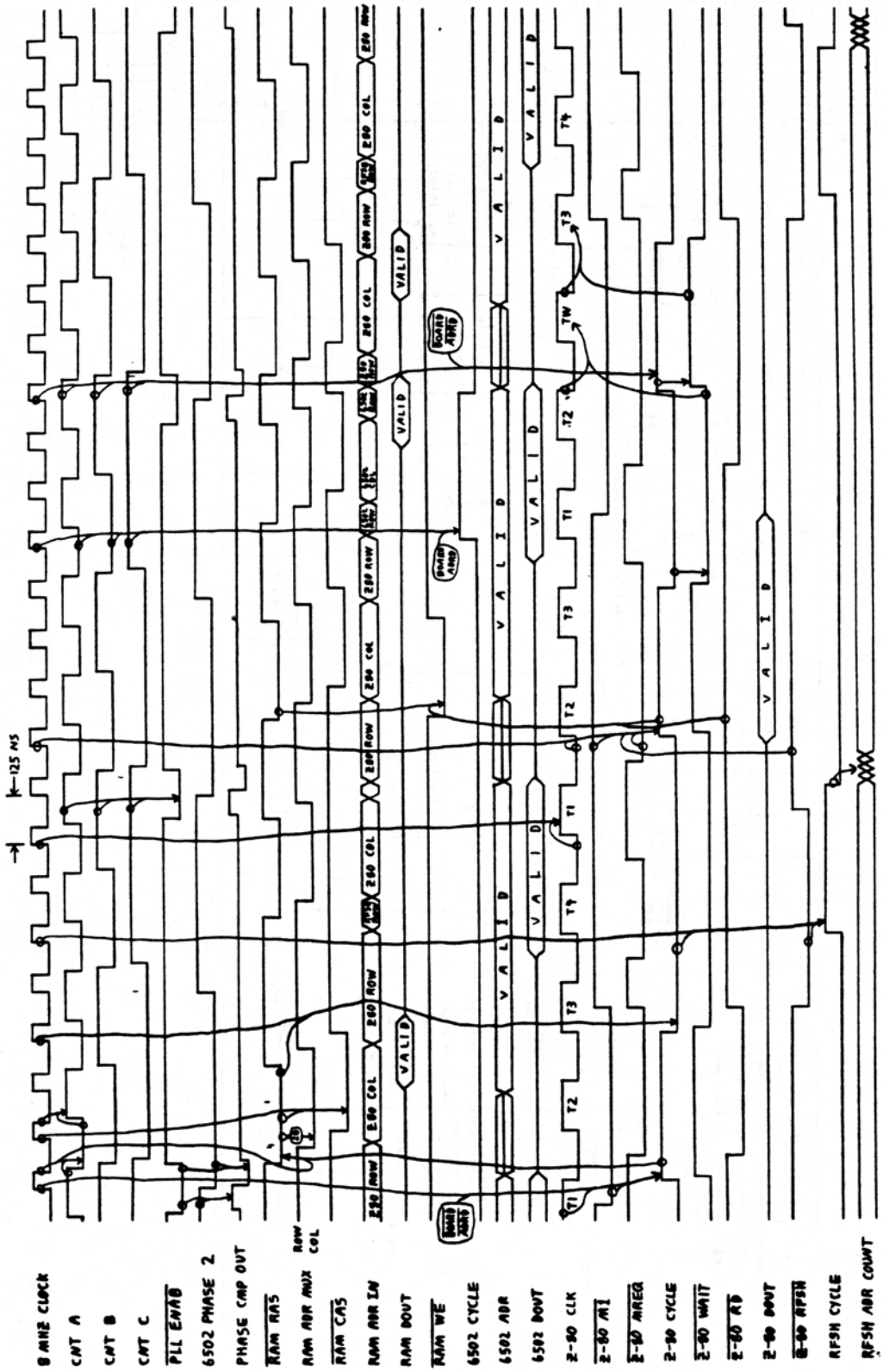
If the memory test program indicates that a single memory IC is defective, the obvious thing is to replace that chip. Nearly any kind of 64K dynamic RAM with an access time rating of 200NS or less can be used. Pin 1 may either be a self-refresh pin or a no-connect.

If problems are experienced with the serial port, check that the interface cable and serial device it is connected to is not putting noise on the CTS (clear to send) line, even pulses as short as 100NS. If such noise is present, it will interfere with the transmitter. If the Z-80 program is receiving spurious interrupts and the SER IRQ EN jumper has been installed to allow interrupts from the serial port, check for similar noise on DCD (carrier detect) line. If such noise is present and the receiver is enabled, there will be an interrupt for each change of state of DCD. It is good practice to disable the receiver when not being used to prevent such interrupts when cables are being connected or external equipment is turned on or off.

6. PROGRAMMOVER BLOCK DIAGRAM



PROGRAMMOVER TIMING

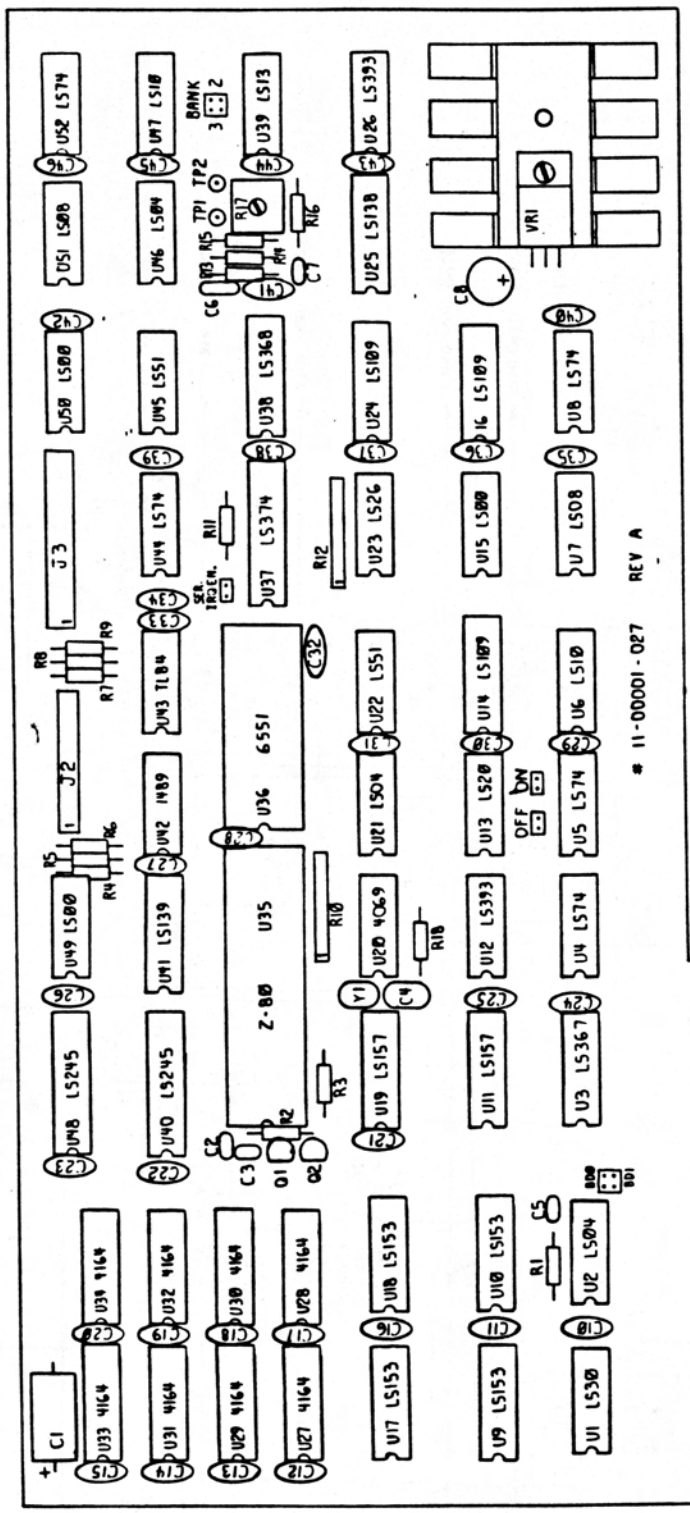


8. PROGRAMMOVER PARTS LIST

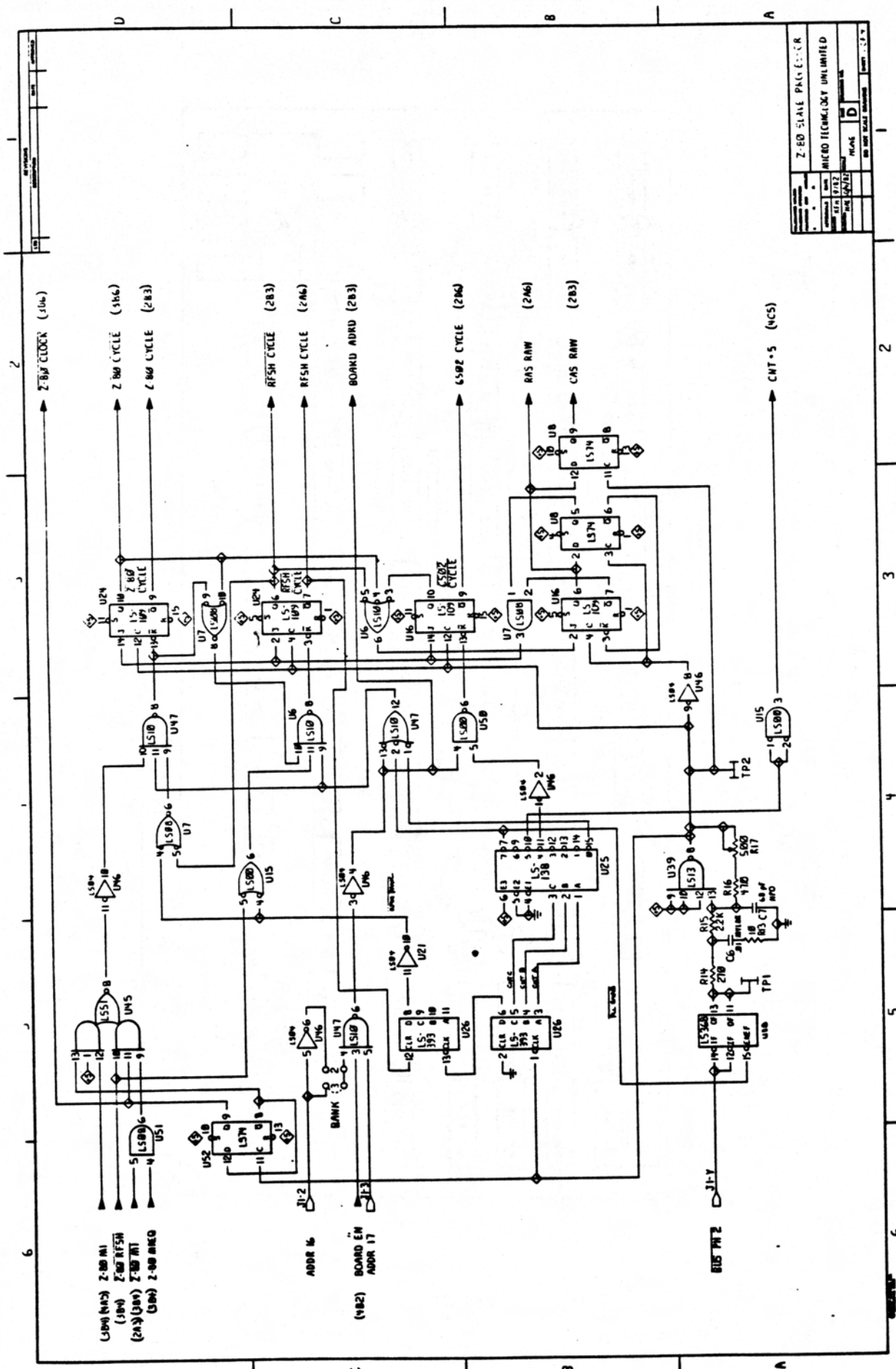
QTY	COMPONENT	COMPONENTS DESIGNATION
3	74LS00	U15,49,50
3	74LS04	U2,21,46
2	74LS08	U7,U51
2	74LS10	U6,47
1	74LS13	U39
1	74LS20	U13
1	74LS26	U23
1	74LS30	U1
2	74LS51	U22,45
5	74LS74	U4,5,8,44,52
3	74LS109	U14,16,24
1	74LS138	U25
1	74LS139	U41
4	74LS153	U9,10,17,18
2	74LS157	U11,19
2	74LS245	U40,48
1	74LS367	U3
1	74LS368	U38
1	74LS374	U37
2	74LS393	U12,26
1	1489	U42
1	4069	U20
8	4164	U27-34
1	6551	U36
1	TL084	U43
1	Z-80A-CPU	U35
26	SOCKET,14PIN,PC	U1,2,4-8,12,13,15,20-23,26,39, 42-47,49-52
21	SOCKET,16PIN,PC	U3,9,10,11,14,16-19,24,25, 27-34,38,41
3	SOCKET,20PIN,PC	U37,40,48
1	SOCKET,28PIN,PC	U36
1	SOCKET,40PIN,PC	U35
37	CAP,Z5U,.05UF,12V	C10-46
2	CAP,NPO,68PF,12V	C5,7
2	CAP,NPO,100PF,12V	C2,3
1	CAP,MYLAR,.01UF,100V,AXIAL	C6
1	CAP,ELECT,100UF,16V,VERT	C8
1	CAP,ELECT,220UF,6.3V,AX	C1
1	RES,1/4W,5%, 10 OHM	R13
1	RES,1/4W,5%,270 OHM	R14
1	RES,1/4W,5%,330 OHM	R7,8,9
1	RES,1/4W,5%,470 OHM	R16
1	RES,1/4W,5%,680 OHM	R1
1	RES,1/4W,5%, 1KOHM	R2
2	RES,1/4W,5%,2.2KOHM	R3,15
4	RES,1/4W,5%, 10KOHM	R4,5,6,11
1	RES,1/4W,5%, 1MOHM	R18
2	RES,1/4W,5%, 10KOHM,SIP	R10,12
1	RES,TRIMPOT,SQ,500 OHM	R17

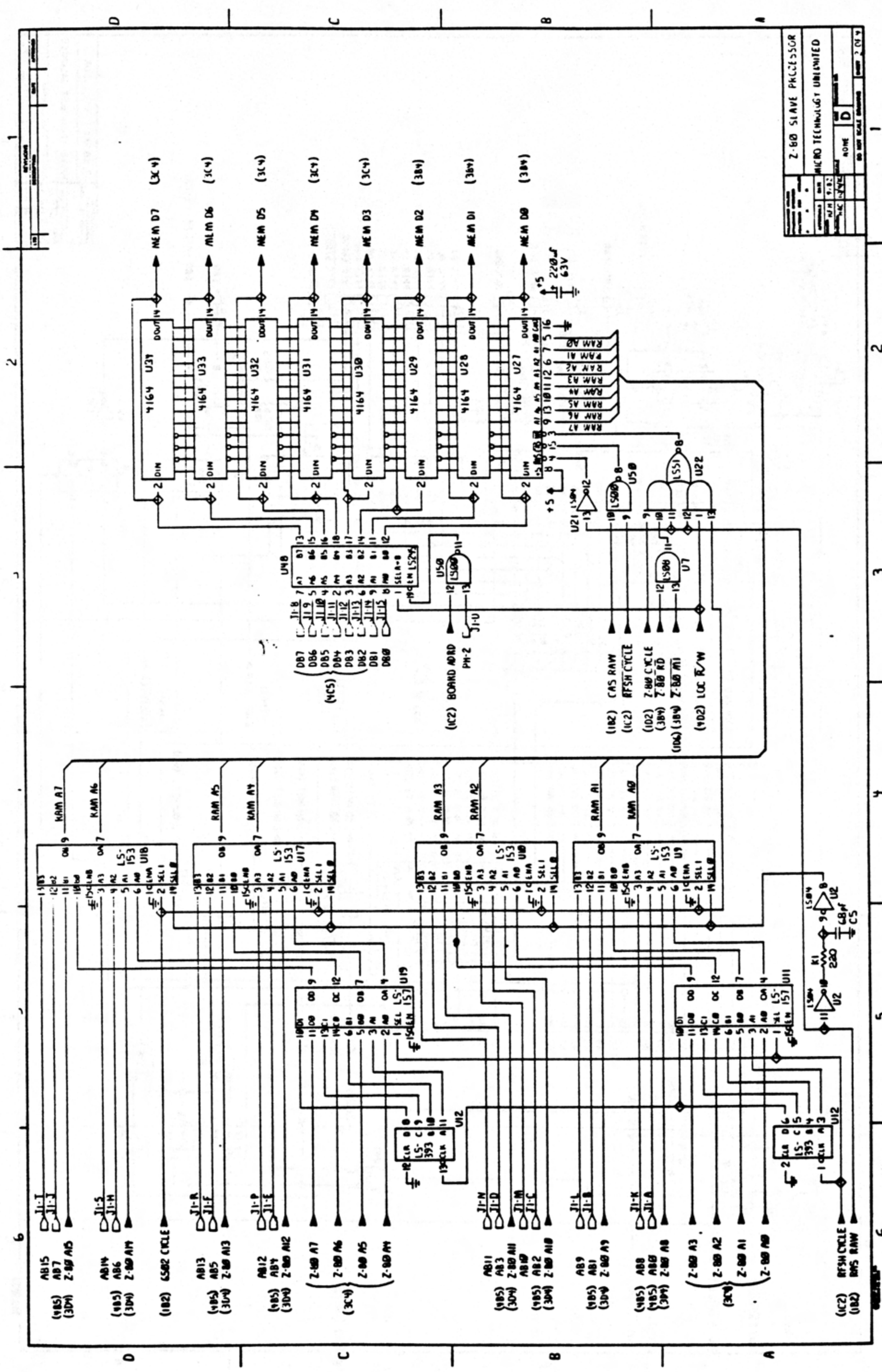
QTY	COMPONENT	COMPONENTS DESIGNATION
1	RESONATOR/CAP,3.68MHZ	Y1,C4
1	TRANS,PN3646	Q2
1	TRANS,PN4916	Q1
1	VOLT REG, LM340T5	VR1
1	HEATSINK,4W	H1
1	CONN,10PIN,SIL	J1
1	CONN,26PIN,DIL	J2
2	HEADER,4PIN,2ROW	
3	HEADER,2PIN	
2	SCREW,4-40X3/8",PH	
2	NUT,HEX,4-40	
1	PCB,Z-80	

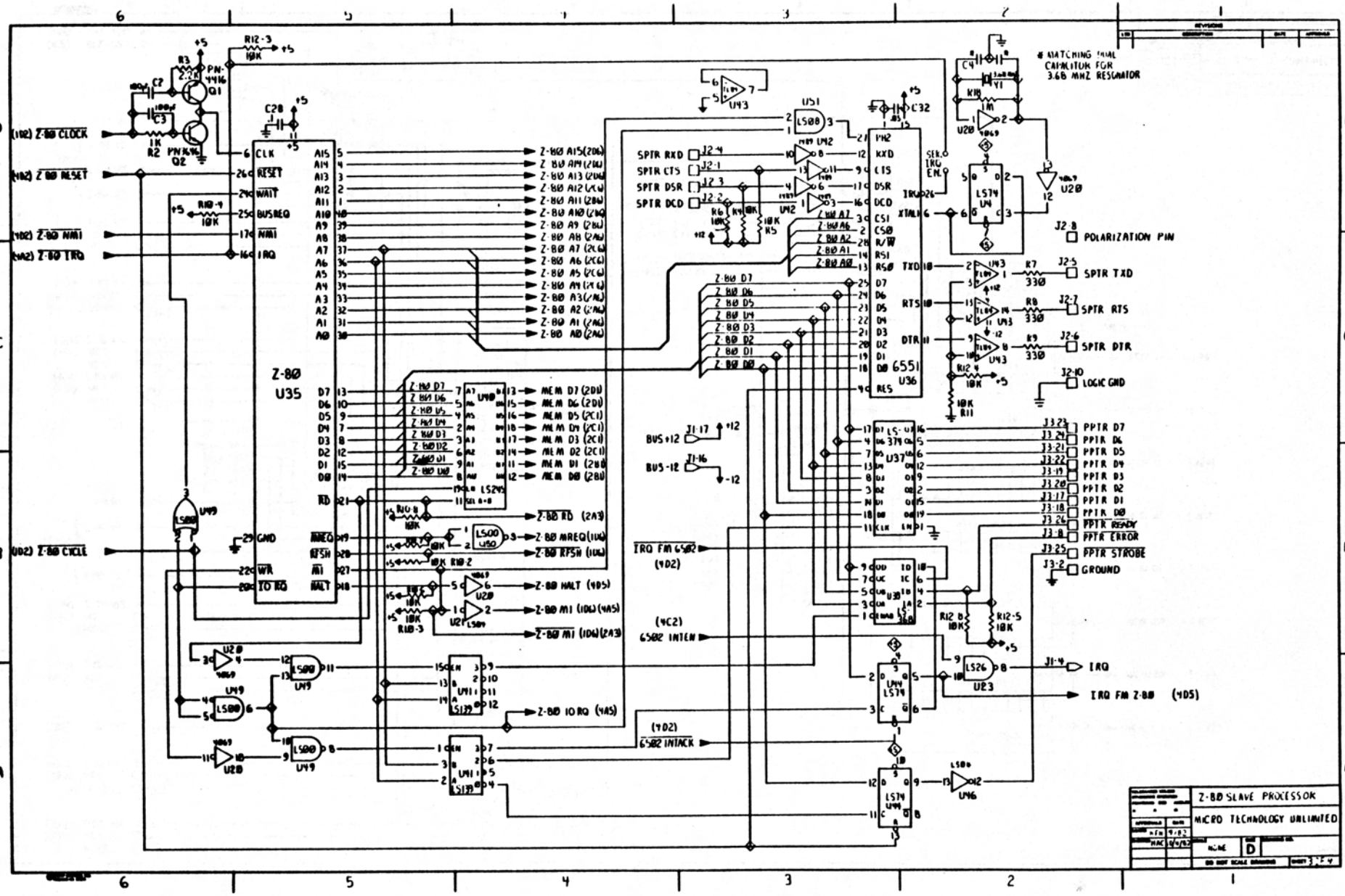
PROGRAMMOVER PARTS LAYOUT

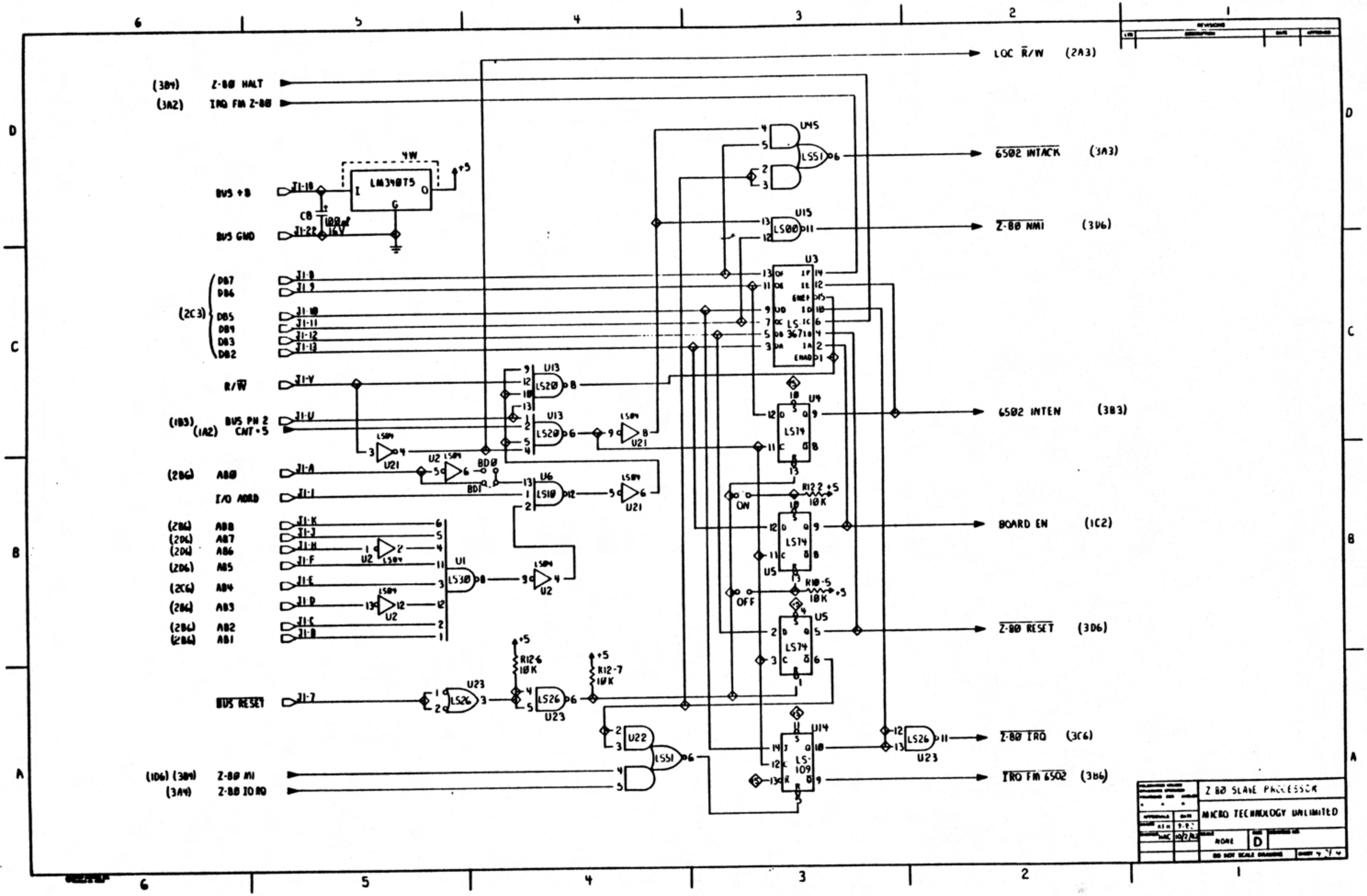


PROGRAMMOVER SCHEMATIC DIAGRAMS




Z-80 SLAVE PROCESSOR
MICRO TECHNOLOGY UNLIMITED
MEM D7
MEM D6
MEM D5
MEM D4
MEM D3
MEM D2
MEM D1
MEM D0
4164
RAM A0
RAM A1
RAM A2
RAM A3
RAM A4
RAM A5
RAM A6
RAM A7
CAS RAW
RFSH CYCLE
Z-80 CYCLE
Z-80 RD
Z-80 M1
LOC R/W
BOARD ADRD
PH-2
6502 CYCLE
RAS RAW


Z-80 SLAVE PROCESSOR
MICRO TECHNOLOGY UNLIMITED
Z-80 CLOCK
Z-80 RESET
Z-80 NMI
Z-80 IRQ
Z-80 CYCLE
Z-80 A15(20H)
Z-80 A14(20H)
Z-80 A13(20H)
Z-80 A12(20H)
Z-80 A11(28H)
Z-80 A10(28H)
Z-80 A9 (28H)
Z-80 A8 (2AH)
Z-80 A7 (2CH)
Z-80 A6 (2CH)
Z-80 A5 (2CH)
Z-80 A4 (2CH)
Z-80 A3 (2AH)
Z-80 A2 (2AH)
Z-80 A1 (2AH)
Z-80 A0 (2AH)
Z-80 D7
Z-80 D6
Z-80 D5
Z-80 D4
Z-80 D3
Z-80 D2
Z-80 D1
Z-80 D0
MEM D7 (20H)
MEM D6 (20H)
MEM D5 (2CH)
MEM D4 (2CH)
MEM D3 (2CH)
MEM D2 (2CH)
MEM D1 (28H)
MEM D0 (28H)
Z-80 RD (2A9)
Z-80 MREQ (10H)
Z-80 RFSH (10H)
Z-80 HALT (9D5)
Z-80 M1 (10H)(4A5)
Z-80 M1 (10H)(2A9)
Z-80 IORQ (4A5)
SPTR RXD
SPTR CTS
SPTR DSR
SPTR DCD
SER.O IRQ EN.
IRQ
XTAL
TXD
RTS
DTR
SPTR TXD
SPTR RTS
SPTR DTR
LOGIC GND
POLARIZATION PIN
PPTR D7
PPTR D6
PPTR D5
PPTR D4
PPTR D3
PPTR D2
PPTR D1
PPTR D0
PPTR READY
PPTR ERROR
PPTR STROBE
GROUND
BUS +12
BUS -12
IRQ FM 6502 (9D2)
6502 INTEN (9C2)
6502 INTACK (9D2)
IRQ FM Z-80 (9D5)
MATCHING TRIAL CAPACITOR FOR 3.68 MHZ RESONATOR



LOC R̄/W (2A3)
(3B4) Z-80 HALT
(3A2) IRQ FM Z-80
6502 INTACK (3A3)
Z-80 NMI (3D6)
6502 INTEN (3B3)
BOARD EN (1C2)
Z-80 RESET (3D6)
Z-80 IRQ (3C6)
IRQ FM 6502 (3B6)
BUS +8
BUS GND
LM340T5
(2C3) DB7 DB6 DB5 DB4 DB3 DB2
R/W̄
(1B3) BUS PH 2
(1A2) CNT+5
(2B6) AB0
I/O ADRD
(2B6) AB8
(2D6) AB7
(2D6) AB6
(2D6) AB5
(2C6) AB4
(2B6) AB3
(2B6) AB2
(2B6) AB1
BUS RESET
(1D6) (3B4) Z-80 M1
(3A4) Z-80 IORQ
ON
OFF
BD0
BD1
R122 +5 10K
R10-5 10K
R126 10K
R12-7 10K
Z 80 SLAVE PROCESSOR
MICRO TECHNOLOGY UNLIMITED
NONE
D
DO NOT SCALE DRAWING