RICE UNIVERSITY

A MICROPROCESSOR SYSTEM DESIGNED FOR USE IN THE LABORATORY

by

NIGEL D. WAITES

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE:

Dr. J. Robert Jump, Director
Professor of Electrical and Computer Engineering

Dr. J. Bartlett Sinclair
Associate Professor of Electrical and Computer Engineering

Dr. Joel H. Cyprus
Lecturer in Electrical and Computer Engineering

Houston, Texas
May, 1988

A MICROPROCESSOR SYSTEM DESIGNED FOR USE IN THE LABORATORY

NIGEL D. WAITES

RICE UNIVERSITY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

Abstract

The goal of the project was to design a low cost microprocessor system for laboratory use. A general purpose microprocessor board was conceived that could be used in a card type system or with the special purpose mother board, which would give the microprocessor board protection from the students, and which would conveniently distribute signals through breadboard terminals for connection to external circuitry.

This thesis focuses on the development of the system, both in terms of hardware and software.

The project's goals have all been achieved; forty microprocessor systems were built, and they are presently being used in the laboratory. The bundled hardware and software package gives the user access to the equivalent of a small development system that provides a user-friendly environment, at a fraction of the cost of any commercially available development system.

Acknowledgments

I wish to thank my advisor Dr. J. R. Jump for his support throughout the project. I also wish to thank Dr. J. D. Wise for his invaluable input.

Special thanks are in order for Hubert Daugherty, who designed the Printed Circuit Board layout and administered the purchasing of all the components. The systems were assembled with help from Edward Smith. I thank him for his help. I would also like to thank Dan Louge for providing feedback on using the early versions of the software.

On behalf of Rice University I would like to thank the following companies for making valuable donations to the project:

Motorola Semiconductor, Inc.

Texas Instruments Incorporated

Hewlett Packard

Finally, I would like to thank Major E. Carter for his software contribution.

TABLE OF CONTENTS

Chapter 1

Introduction

A digital design course is offered as part of Rice University's Electrical and Computer Engineering curriculum. This course teaches digital logic design and microprocessor interfacing.

For the last several years the course has been taught using a Z80-based microcomputer. The 8-bit Z80-based microcomputer is now dated due to the arrival of various 16-bit microprocessors. Faculty members in the Electrical and Computer Engineering Department were aware of this fact and decided that it was desirable to develop a new system.

The original system consisted of a metal chassis with plug-in modules. The metal chassis provided power to the modules from an external power supply. The power supply and microprocessor unit were connected to the chassis via cables. The modules provided breadboarding areas for interfacing to the microprocessor. Other modules provided switches and light emitting diodes.

This project was undertaken in an attempt to replace the existing Z80-based system with a system using the Motorola MC68000 family of microprocessors. The MC68000 family has widespread usage in industry, and MC68000 assembly language programming is taught widely in educational environments. The system consists of two printed circuit boards, one for the microprocessor and its support chips, the other for various buffer circuits. The microprocessor board is a stand alone unit that can be used for any general purpose project requiring the functions of a microprocessor. The buffer board provides the interface between the laboratory user and the microprocessor board, and can only be used in conjunction with the microprocessor board.

Much was learned from the existing Z80 microprocessor system, which clearly had some weaknesses in the laboratory environment. In the design of the new system these problems were recognized and eliminated as far as possible.

The material presented in the following chapters describes both the hardware design and software implementation. The second section describes the system hardware, with explanations of the design strategies. The third chapter describes the software incorporated with the system. The fourth chapter evaluates the cost involved in the development of the system. The fifth chapter is a User Guide which describes how the system is employed. The sixth chapter draws conclusions on how the system performs, and discusses areas in which further software development could improve the system.

Chapter 2

Hardware Design

2.1 Design Issues

The existing Z80 system was retired, but the prototyping equipment used in conjunction with the system was retained. This included breadboards, switch modules and a metal chassis. The old system had a separate microprocessor board and power supply which were connected to the chassis via cables. The ribbon cable connections had been very unreliable. Therefore, one major goal of the new system was the elimination of as many mechanical connections as possible. This was achieved by having both the power supply and microprocessor system connect directly to the chassis.

2.2 Power Supply

The power supply's physical size was one major issue in the selection choice, as it had to be less than two inches tall to fit into the chassis. The power supply also had to meet certain functional requirements including low heat dissipation (as it is enclosed in a metal box with little ventilation), good regulation, and most importantly short circuit immunity. The power supply had to furnish +5 volts at 5 amperes, as well as +12 volts and -12 volts, each at 1 Ampere.

A linear supply was ruled out due to both size and heat dissipation. Various switch mode power supplies were tested, with all but one failing the short-circuit test, contrary to manufacturer's claims. The supply finally chosen is the Power General 3050-1A, as it was the only one which satisfied all the above requirements. This particular supply must be preloaded with a minimum of one ampere drawn from the +5 volt output to maintain correct regulation. This is achieved inside the chassis by a 5-ohm resistor (10 watt) connected from +5 volts to ground.

2.3 Physical Size

The new microprocessor unit plugs directly into the chassis alongside the breadboard modules. To allow adequate breadboarding space, the microprocessor unit's size was constrained to ten inches by five and three- quarter inches. The buffer (mother) board can utilize the full ten by five and three-quarter inches; however, to allow for easy access to the breadboard connectors, the microprocessor board must be no wider than three and one-quarter inches. This immediately places a severe physical constraint on the complexity of the microprocessor board.

2.4 Microprocessor Board

The basic goal of the design was to produce a module with the microprocessor , memory, and peripherals on one printed circuit board. The module also needed to be used in either a stand alone mode or in conjunction with the buffer board.

The entire circuitry had to fit onto a two-sided printed circuit board (PCB) measuring three and one-quarter inches by ten inches. The width constraint on the PCB forced the designer to reduce the chip count to a minimum, as physically there is very little space. This size restriction immediately pointed to the MC68008 microprocessor which has 48 pins as opposed to the 64 pins of the MC68000 microprocessor. The MC68008 is a 16-bit microprocessor with an 8-bit data bus permitting the system memory to be only 8 bits wide instead of the 16 bits required by the MC68000. This significantly reduces the minimum memory chip count. The MC68008 has a 20-bit address bus as opposed to the 24-bit address bus of the MC68000. However, a twenty bit address bus gives one megabyte of address space, which is more than adequate for this type of system. Furthermore, MC68008 maintains full software compatibility with the MC68000.

The microprocessor unit has two uses. Firstly it is to be used by students to learn both interfacing and programming techniques. Secondly, it is to be used as a general purpose microprocessor board. Both serial and parallel peripheral devices were desired features to give maximum flexibility. The Motorola MC68681 Dual Channel UART was chosen for the serial device, as this device is simple to use, robust , and inexpensive. The Dual Channel UART allows one channel to be permanently assigned to communication with the host machine or terminal, while the second channel is available for general use. The MC68681 provides simplified interfacing to the MC68008 and offers the following features:

1. On chip dual baud rate generation, up to 38k baud.
2. Programmable baud rate, via an on-board timer.
3. An on-board counter/timer which can be used to generate one shot pulses or square waves with variable duty cycles.

4. A local internal loopback diagnostic feature, allowing software diagnostics to check both communication channels.
5. Programmable handshake lines (CTS and RTS) on both channels.
6. Several general purpose input/output lines.

In the selection of the parallel port device the Rockwell R6522 Versatile Interface Adapter was found to be a flexible and cost effective part. The R6522 is fundamentally an MC6800 part, so extra logic is required to interface it to the MC68008. The R6522 provides the following features:

1. Sixteen fully programmable input/output lines.
2. Two counter/timers with programmable output pins giving either one-shot or square-wave output.
3. Four input lines capable of generating interrupts on either positive or negative transitions.
4. An 8-bit shift register, which can be used for shifting in or out under control of various internal and external clock options.
5. An Interrupt Status register (for polling) and an Interrupt Mask register, which allows full control of interrupt generation.

These peripheral devices are 'glued' to the microprocessor via some discrete logic in the form of 74LS chips, and two PAL devices.

2.5 Buffer Board

The buffer board's prime objective is to protect the expensive chips on the microprocessor board. The buffer board has one hundred breadboard terminals that allow easy connections to be made to the microprocessor board. The buffer board, as its name suggests, buffers the majority of the bus signals, allowing continual short circuiting of bus signals with no permanent damage resulting.

One of the biggest problems with the existing Z80 system was that the parallel ports were damaged very easily, and there was no direct and effective way of diagnosing the problem. The buffer board has circuitry which allows full testing of the parallel ports, and in conjunction with the system software a diagnostic check can be run at any time.

The buffer board also contains four seven-segment LED displays which can display the hex digits 0-F, giving the ability to display 16-bit quantities.

The buffer board connects directly to the chassis via five banana jack connectors (which are keyed, so that correct insertion is assured) which deliver power to the system. Power and bus signals connect to the microprocessor board by three ribbon cables carrying a total of one hundred twenty signals.

The buffer board has two DB25 connectors for RS232 connections.

2.6 Hardware Implementation

2.6.1 Clock Speed

The MC68008 is designed to be run from a clock at a speed no faster than 8MHz and no slower than 4MHz. The obvious choice is 8MHz, giving maximum CPU performance, provided that the memory system is able to respond quickly enough. The MC68681 must also be driven from an external clock, and its clock rate must not exceed 4MHz. The internal baud rate generation inside the DUART is carried out by a divider chain with taps for specific baud rates. The divider chain is designed for use with a 3.6864MHz clock input. This gives standard baud rates such as 1200, 2400, 4800, 9600 and others. The MC68681 DUART is very flexible, as the clock can be driven by either a TTL output or directly from a crystal. To alleviate the need for a second crystal and hence reduce cost, the MC68000 CPU, which requires a clock, is driven from a 7.3728MHz clock package's TTL output. This is divided by two to generate the 3.6864MHz clock required by the DUART. The division is carried out by U19 which is a D flip-flop with its negated output connected to the D input, effectively making a T flip-flop. Note that when a TTL clock is being fed into the DUART pin X2 (pin 33) must be tied to ground. The schematics for the microprocessor board are shown in Figure 1.

Note that the E (ENABLE) clock from the MC68008 is the system input clock divided by ten, and hence the E clock runs at a frequency of 737.28kHz.

2.6.2 Address Space Management

The MC68008 has a linear address space of one megabyte (*i.e.*, twenty address lines). The MC68000 family of microprocessors support two modes of operation, 'user' mode and 'supervisor' mode. These modes are displayed on the FC0-2 (Function code lines) which can also be used in the address decoding logic to give an effective four megabyte address space. For simplicity the bottom half of the address space (0-$7FFFF) is used for on-board addressing, while the upper half-megabyte ($80000-$FFFFF) is used for

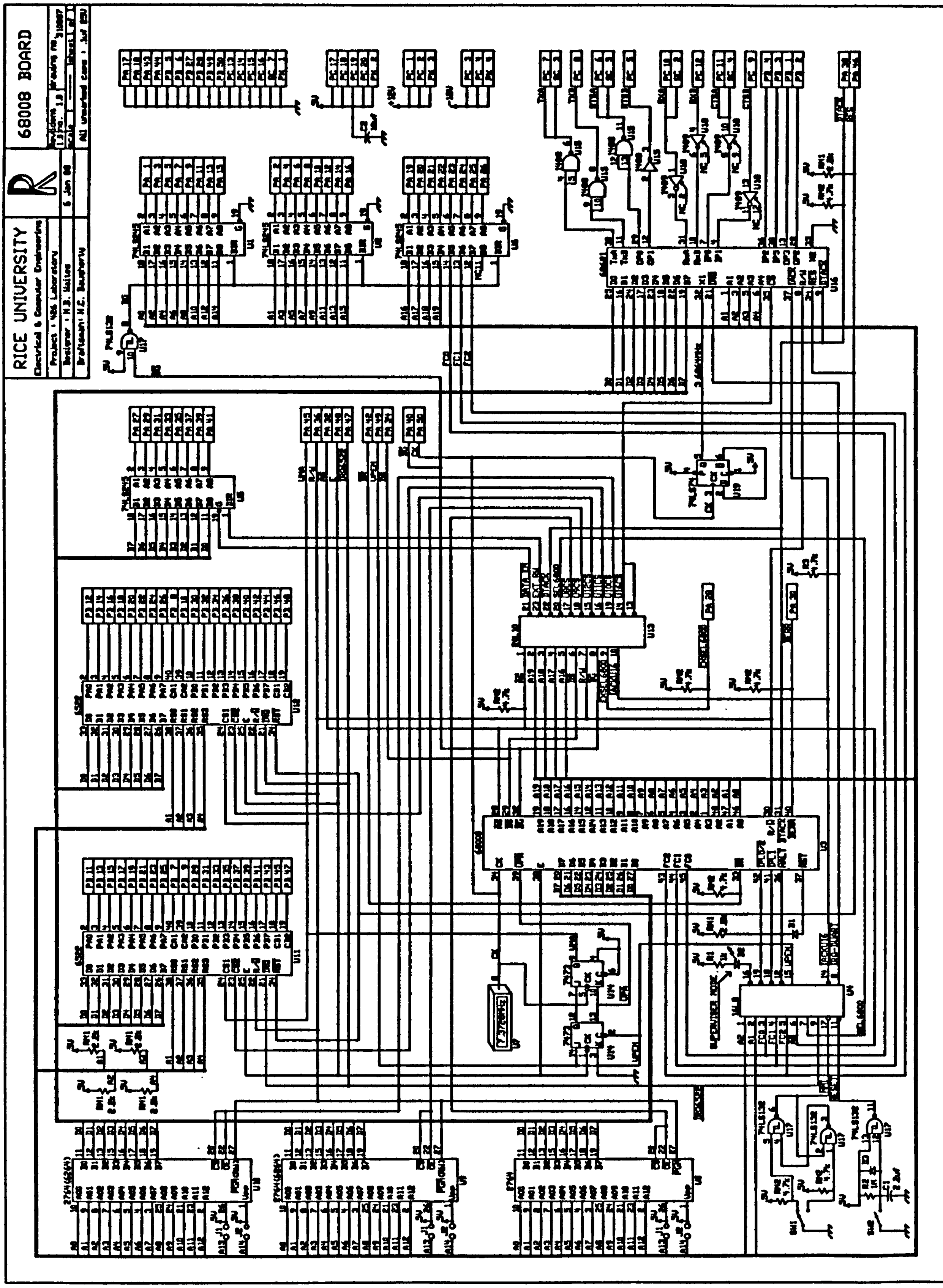

RICE UNIVERSITY
Electrical & Computer Engineering
68008 BOARD

Figure 1

off-board addressing.

This means that each device can be allocated a large block of address space, as all the on-board devices have to fit within a half-megabyte address space. The largest common denominator was used. This is determined from the memory size. The largest memory chip available in a 28-pin DIP is a 64k EPROM. Each device is therefore allocated a 64k memory block. Allocating such large blocks to I/O devices seems rather wasteful, but it simplifies the address decoding logic significantly. Each device is allocated a 64 kbyte

block in the bottom half-megabyte. This means there are 8 addressable devices, and that these devices are selected by decoding address lines A16-A19.

The address decoding is implemented using a programmable array logic device (PAL).

2.6.3 Memory Interfacing

System memory consists of three sockets which are wired for 6264s (8 kbyte static RAM) or 2764s (8 kbyte EPROM). The access time for the RAM or EPROM chips should be 250 nanoseconds or faster, as the no-wait state bus access is around 290 nanoseconds, and the chip selects take approximately 35 nanoseconds to propagate.

Note that each memory chip is mapped into a 64 kbyte block, and hence it folds eight times. The three memory blocks are mapped as follows:

0-$FFFF Memory chip U8

$10000-$1FFFF Memory chip U9

$20000-$2FFFF Memory chip U10

The address decoding is done via the PAL 20L10 (U13). The equations for the three memory blocks are:

CS0=AS*/A19*/A18*/A17*/A16

CS1=AS*/A19*/A18*/A17*A16

CS2=AS*/A19*/A18*A17*/A16

/ - is the NOT operator. For more information on PALASM equations see Appendix A.

AS is defined as an active low signal.

CS0-2 are active low outputs on the 20L10.

The memory chip selects become active when the address has been set up and is stable, indicated by the assertion of AS, and by the correct combination of address lines. When a read cycle occurs, pin 27 of the decoded memory chip is high (due to R/W being high), and both CS and OE go low, enabling the CPU to read data from the memory. Note, however, that when a write cycle occurs, pin 27 of the memory chip is low (*i.e.*, R/W is low). At first glance it appears the memory chip is trying to output data (Output enable is low) and write data (Write enable is low) simultaneously. However, WE overrides OE and the write occurs. This feature allows each socket to contain either an EPROM or a RAM chip.

The MC68008 boots itself from address locations 0-7, and hence the bottom 8k (CS0) must be either EPROM or battery backed-up RAM. If battery backed-up RAM is used, it can be written to freely. The ability to write into the bottom 8 kbytes has an undesired effect when the socket contains an EPROM. Data would be output from the CPU, when the R/W line is low. This means that PGM is driven low. When PGM is low the device is in the program mode and the data pins become inputs. While the EPROM is in this state, Vpp however, is held at +5 volts, which is far less than the 21 volts applied for several milliseconds which is required for programming. This undesired write cycle thus has no effect on the EPROM, even though the EPROM appears to be in a program mode cycle. This connection scheme however does allow a battery backed-up RAM to be used for both reading and writing.

The DTACK signal for the memory is generated by the 20L10 PAL. This will be explained later in the DTACK generation section 2.7.3.

The microprocessor board has jumper connections to assist in upgrading the memory sockets to use 62256 or 27256 (32 kbyte RAM or ROM) chips. The 32 kbyte packages, unfortunately, have slightly different pinouts, as a consequence after the modifications are carried out, only 32k chips can be used, and the EPROM's and RAM's are longer interchangeable. When the modifications are made, therefore, each socket must be designated to either a RAM or a ROM chip. Details on this modification are given in Appendix C.

2.6.4. The MC68681 DUART

The Motorola MC68681 Dual Asynchronous Receiver Transmitter (DUART) is designed primarily for use with the MC68000 family of microprocessors. The DUART runs from a 3.6864MHz clock derived from the system clock. The MC68681 has a standard bus interface consisting of address lines, data lines and a chip select. The DUART generates its own DTACK using its external clock. Note that the DUART sometimes inserts wait states, as it is not capable of running no-wait state bus cycles. The interrupt sequence will be discussed in the Interrupt structure section 2.8.

The MC68681 has two complete asynchronous communication channels, with the ability to be configured via software to handle RS232 handshaking using RTS and CTS. To conform to RS232 standards the signals have to be converted from TTL levels to RS232 levels on the output lines and vice-versa for the input lines. The RS232 levels require a TTL 'low' level to be converted to plus 3-15 volts and a TTL 'high' level to be converted to minus 3-15 volts. Several chips will perform this operation. The most cost effective are the 1488 line driver and the 1489 line receiver, which provide four gates per package. The DUART requires four output lines (TXA,TXB,RTSA,RTSB) and four input lines (RXA,RXB,CTSA,CTSB) to be translated. The 1488 and 1489 are therefore fully utilized. These chips also provide the correct electrical characteristics for the RS232 specification in terms of capacitance and resistance. Note the power requirements for each chip. The receiver requires only +5 volts, whereas the line driver requires +12 and -12 volts. The RS232 level converters are mounted on the microprocessor board, allowing the microprocessor board to be connected directly to a terminal via the 'SC' header block.

2.6.5 The R6522 VIA

The microprocessor board has two R6522 Versatile Interface Adapters, which provide parallel input and output. The R6522 is a 6500 family part, which is a predecessor of the MC6800 family. The MC68008 interfaces to the R6522's via the MC68008's ability to emulate an MC6800 bus cycle.

To assist in the MC6800 bus interface, the MC68008 first provides an Enable (E) clock. The E clock for normal speed 6500 parts must be less than or equal to 1MHz. The E clock generated by the MC68008 is the processor clock divided by ten, and therefore the E clock runs at 737.28kHz. The MC68008 has only 48 pins as opposed to the MC68000's 64 pins. One of the missing pins is part of the MC6800 interface. This signal must therefore be generated externally.

The MC6800 and MC68000 bus cycles co-exist, with the appropriate bus cycle being run according to the type of device selected. The MC68008 has an input called Valid Peripheral Address (VPA). This input is asserted whenever an MC6800 device is being selected. VPA is generated by decoding of the appropriate addresses. When an MC6800 bus cycle is selected, an output from the 16L8 PAL, Valid Peripheral Enable (VPEN) becomes true. This enables the flip-flop U14, which outputs a high on Q after the E clock goes low. The output NOT Q from the flip-flop signals to the MC68008 that it should execute an MC6800 bus cycle, the bus cycle being requested by making the VPA input low. After one processor clock cycle, the second flip-flop's output Q goes high. This output signal is Valid Memory Address (VMA). Note, this is internally generated on the MC68000. VMA indicates that the MC68008 has internally synchronized itself for an MC6800 bus cycle. The MC6800 bus cycle uses the positive pulse from the E clock to synchronize its data transactions between the CPU and the peripheral. The flip-flop circuitry stops any incomplete positive pulses from propagating through via assertion of VPA by the flip-flop on the falling edge of the E-clock.

When VMA is asserted, the MC6800 peripheral device is now guaranteed of a full positive E pulse, with the negative edge of the E clock capturing the data and terminating the cycle. After the negative edge of the E clock has occurred, AS is deasserted. VPEN then becomes negated. This clears the flip-flop and removes VPA. Note that during every MC6800 bus cycle DTACK must be held high for the complete bus cycle.

Each R6522 device must be enabled when VMA is asserted to ensure correct operation. This is accomplished by using the two chip selects that each R6522 possesses. The active low chip select is generated by simple address decoding in the PAL. Therefore this chip select is active immediately after AS is asserted. The other chip select is active high and is tied to VMA. The R6522 is then selected at the correct time due to the combination of its two chip selects.

2.6.6 A Special Characteristic of the R6522

The R6522 Versatile Interface Adapter was actually originally designed to be used with the 6500 family of microprocessors. One of the major characteristics of the 6500 family is that the control and address busses are always driven. They are never allowed to float.

During testing of the prototype microprocessor board the R6522s were found to exhibit strange behavior. The parallel output lines would change states randomly when the processor executed the STOP instruction, which floats the address, data and control lines. At first the problem was unclear, as some chips exhibited this phenomena, and others worked correctly. The outputs would sometimes change several seconds after the STOP instruction had been executed. This behavior was very mysterious, since the devices changed their outputs despite the fact that their chip selects were negated. After much experimenting it was discovered that in certain circumstances it was possible to change the outputs by shorting the floating address lines to ground.

At this point the manufacturer was contacted, and an application engineer explained politely that this was an undocumented feature of the part. However, Rockwell had sold the R6522 to Apple for use in the Macintosh, and during that transaction a Rockwell engineer discovered that pull-up resistors on the address bus fixed the problem. Rockwell also decided to fix the part. The new part is called the R65NC22. The microprocessor board has pull-ups allowing original R6522 parts to be used, although all the systems built contain the new part.

2.6.7 Peripheral Register Addressing

All registers on both the R6522s and the MC68681 appear on even address boundaries. This maintains compatibility with a normal MC68000 design, which functionally means that successive registers can be read or written to using the MOVEP instruction. It is important to notice that there is a distinct physical difference between the connection of the MC68008 and MC68000 to peripheral devices. The MC68000 has sixteen data lines, and only eight are connected to peripheral devices. The MC68008 has only eight data lines. These are used in conjunction with the A0 address line to emulate the MC68000.

At first glance it is not clear what the functional differences are, however let us consider the move instruction which has an undesirable effect when executed on certain peripherals.

Consider the instruction: MOVE.W #$FF,PERIPHERAL_REG

First let us consider what happens on a MC68000 system. If the bottom eight data lines are connected to the peripheral device and the chip-select depends on the Lower Data Strobe (LDS), the peripheral register will have $FF written into its register. The top byte (zero in this example) is put on the top eight data lines and it is ignored by all devices. However, on the MC68008 the top byte is written into the register (*i.e.*, zero), the address line A0 is set to '1', and $FF is written into the register. This causes two problems. Firstly, many peripheral devices need a substantial recovery time between bus transactions, and secondly, some devices have internal register addressing triggered from external addresses (the MC68681 has the MR1-2 registers which are a classic example). The double bus cycle will cause the register to flip before the correct data is written into the register.

The conclusion is that if one intends to write code which is to be MC68008/MC68000 compatible, care must be taken to ensure correct addressing of peripherals. This can be achieved, among other ways, by using the move byte instruction rather than the move word instruction illustrated above.

2.7. 'Glue' Logic

The microprocessor, memory and peripherals are glued together with two programmable array logic devices, U4 (a PAL16L8) and U13 (a PAL20L10).

2.7.1. Reset and Supervisor Mode Logic

System reset during power-up is generated by an RC circuit. This is cleaned up by the 74LS132 two input NAND Schmitt trigger. The output of the Schmitt trigger is fed into the 16L8. RESET to the CPU is an output from the PAL. The equations which determine the RESET output are:

HALT=RESET

HALT.TRST=RESET

(For a brief description of PAL equation syntax, see Appendix A)

When RESET (PAL input) is high, HALT goes low. The tri-state output is also enabled when RESET is high. HALT therefore goes low whenever RESET is high, and whenever RESET is low the output is in high impedance. The output HALT is pulled up so when the output is in the high impedance state and no signal is driving the HALT line, the line is high. The HALT output is effectively simulating an open collector gate, this is necessary as the HALT signal on the MC68008 is bidirectional.

Note the exact connection of the HALT line, the HALT pin and the RESET pin (RESET is also a bidirectional line). On power-up (or master reset via depression of the optional switch SW2) both the HALT and RESET pins must be taken low to ensure a system reset. The HALT output goes low, taking HALT low. The diode D1 allows current to flow out of the RESET input which brings RESET low. The diode D1 is Germanium. These diodes have a slightly lower forward drop than silicon diodes, to ensure that RESET stays below 0.8 volts. The peripheral chip's RESET inputs are connected to the RESET input on the MC68008. This allows the software 'RESET' instruction to reset the peripheral devices. When the 'RESET' instruction is executed the 68008's RESET line goes low, but the diode D1 blocks current from the HALT pin, and the HALT pin remains high. If a double bus error occurs, the microprocessor drives the HALT line low, which brings RESET low and the MC68008 is reset.

The microprocessor board has one LED which indicates when the processor is in Supervisor mode. This signal is generated by the appropriate decoding of the FC0-2 lines within 16L8, with the qualification of AS. The equation is:

SUPERV=/FC0*FC1*FC2*AS+FC0*/FC1*FC2*AS

2.7.2 Address Decoding Equations

The address decoding is implemented in the 20L10, with the devices mapped as follows:

CS0	$0-$FFFF	RAM/ROM	U8
CS1	$10000-$1FFFF	RAM/ROM	U9
CS2	$20000-$2FFFF	RAM/ROM	U10
CS3	$30000-$3FFFF	MC68681	U16
CS4	$40000-$4FFFF	R6522	U11
CS5	$50000-$5FFFF	R6522	U12

The equations for these outputs, and the equations for both PAL's are given in Appendix A.

2.7.3 DTACK Generation

The three RAM/ROM sockets require that Data Acknowledge (DTACK) be asserted. This is performed by the 20L10. The DTACK signal is an open collector signal. The PAL implements this signal similarly to the HALT output. The equations for DTACK are:

DTACK=DS*CS0+DS*CS1+DS*CS2

DTACK.TRST=DS*/ICS3*/CS4*/CS5*/A19

When any of the ROM/RAM chips is selected, the DTACK output goes low. Otherwise it is driven high. The tri-state output is enabled when the DUART and the R6522s are not selected and the address is in the bottom half-megabyte.The DUART supplies its own DTACK, and the R6522s use MC6800 bus cycles which do not involve DTACK.

2.7.4 MC6800 Bus Cycle Selection

The MC6800 bus cycle is selected when the signal SEL6800 goes high. This occurs when either CS4 or CS5 is low (*i.e.*, selecting either R6522), or when EXTSEL6800 is low. The equation for SEL6800 is:

SEL6800=CS4+CS5+EXTSEL6800

The 16L8 has active low outputs, so this equation is negated using DeMorgans law, and the equation in the PAL is:

/SEL6800=/CS4*/CS5*/EXTSEL6800

The SEL6800 output is connected to the 20L10 PAL which generates VPEN. This signal generates VPA through the flip-flop which informs the MC68008 that a MC6800 bus cycle should be executed. The VPA signal from the flip-flop ensures the MC68000 is correctly synchronized for the bus cycle.

The equation for VPEN is:

VPEN=AS*(SEL6800+IACK6800)

IACK6800 will be explained in the interrupt management section 2.8. The 16L8 PAL also has active low outputs, hence the equation must be inverted:

/VPA=/AS+/SEL6800*/IACK6800

2.8 Interrupt Management

The MC68008 differs from the MC68000 in that it only has two effective interrupt lines. The MC68000 has three interrupt input lines, defining seven levels of interrupts and a no interrupt condition. On the MC68008, the IP0 and IP2 interrupt lines are internally tied together allowing only level 2, 5 and 7 interrupts to be generated.

The 16L8 encodes the three sources of interrupts into the two pins required by the MC68008 with the following equations:

IP0_2=IRQ_DUART + IRQSW

IP1=IRQ6522*/IRQ_DUART+IRQSW

The IP0, IP1, and IP2 lines are active low, the 16L8 provides active low outputs. The level 7 interrupt is non-maskable. This feature is used as a 'soft' reset, allowing registers to be dumped. The NMI input is driven from the debounce circuit consisting of U17. When NMI becomes active, both IP0_2 and IP1 (Level 7 interrupt) go low, regardless of any other device interrupts. When the MC68681 requests an interrupt, IP0_2 goes low and a level 5 interrupt is requested. If the R6522 also requests an interrupt simultaneously, the NOT IRQ_DUART term causes the R6522 interrupt to be ignored until the MC68681 removes its request. This effectively means that the three interrupts have a priority scheme, with the NMI interrupt having the highest priority, the MC68681 second highest priority, and the R6522 the lowest priority. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. The R6522 interrupt output pins are open collector gates, allowing the two R6522 interrupt pins to be connected together. The same signal is also run out to the breadboard connector allowing an external source to generate level 2 interrupts. If multiple interrupt sources are used, software polling may be required to determine the source.

When the MC68008 responds to an interrupt, it places the interrupt acknowledge code on the FC0-FC2 lines. The interrupt level is placed on the address lines A1-A3 (A0 is not used, as the MC68000 does not possess an A0 pin) with the level being displayed with positive logic.*i.e.*, a level five interrupt will drive A1 and A3 high, and A2 low.

The 16L8 PAL decodes the acknowledge cycle and forces the processor to use either vectored or autovectored interrupts as appropriate. The 16L8 only uses the address lines A1 and A2 as inputs. A3 is not decoded, since IP0 and IP2 being internally tied together forces A1 to always be equal to A3 in the interrupt acknowledge cycle.

The autovectored interrupts occur for level 7 and level 2 interrupts. The MC68008 executes an autovectored acknowledge cycle when VPA is asserted. The signal IACK6800 is asserted when either a level 7 or 2 acknowledge cycle is decoded with the following equation:

IACK6800=FC0*FC1*FC2*/A1*A2 + FC0*FC1*FC2*A1*A2

The VPA signal is asserted by:

VPA=AS*(SEL6800+IACK6800)

The vectored interrupt is passed across the bus from the DUART after it receives an interrupt acknowledge. This is generated by the equation:

IACK_DUART=FC0*FC1*FC2*A1*/A2*AS

When the MC68681 receives the interrupt acknowledge, it places the vector number on the data bus and then asserts DTACK when it is ready. The MC68008 uses the vector number to decide where to pass program control.

2.9 External Addressing and DMA support

The system address space organization allows external devices to be addressed in the top half-megabyte only. The bottom half-megabyte is used by on-board devices. The address and data lines are buffered using 74LS245s to protect them from external maltreatment.

In normal operation, when the CPU is addressing on-board peripherals, the data bus is not passed through the 74LS245, as the buffer is disabled. (With the data bus buffer disabled, data from the CPU instruction fetches cannot be seen on the external connector. A version of the 20L10 PAL called FADDN.DAT has the data bus enabling configured differently. It enables the data bus buffer to output data to the external connector whenever the bottom half-megabyte of the address space is accessed. This PAL, however will not support certain multiprocessor applications, as the shared data bus is always driven.) The data bus buffer is enabled by a signal called DATA_EN, which is generated by the PAL.

The DATA_EN signal becomes true when either the top half-megabyte is addressed and the bus cycle is not an MC68681 interrupt acknowledge cycle, or when a bus request has been granted and the external bus master addresses the bottom half-megabyte of the address space. The buffer board also has a Parallel Interface Adapter which is addressed at locations $70000-$7FFFF. The DATA_EN signal is also true when these locations are addressed. The Parallel Interface Adapter is mapped in the bottom half of the address space, freeing the whole top half-megabyte to be used externally. The PAL equation to generate DATA_EN is:

DATA_EN=A19*AS*/IACK_DUART * (/BG)

+BG*AS*/A19

+/A19*A18*A17*A16*AS*/BG

If DMA operation is desired, the bus master must not assert AS when A19 is high. This is simply achieved by putting AS and A19 through an OR gate. (This is not required if the term /BG is added as shown in the parentheses.) If the 'FADDN.DAT' PAL is used, certain applications of shared data bus schemes cannot be used, as the data bus is continually driven. Alternatively, a second buffer could be placed after the data bus buffer, and external logic could be used to control it.

The data bus buffer's direction control is selected by a signal called EXT_RW, which is also generated by the 20L10 PAL. This signal is effectively generated by a controlled inverter. When the MC68008 is the bus master, the R/W signal is applied directly to the direction input of the buffer, with a MC68008 read enabling the A-to-B transfer and a write enabling the B-to-A transfer. When an external device becomes the bus master, an external bus master read requires a B to A transfer which is generated by inverting the R/W signal. The controlled inverter is generated in the PAL by the following equation:

EX_RW=BGACK*RW

+/RW*/BGACK

where BGACK controls the inversion, since its assertion means that an external device is controlling the bus.

The address lines are also buffered. These buffer directions are reversed when an external bus master takes over the bus. The BGACK line is inverted by the NAND gate U17 to drive the direction input of the address buffers. The buffer directions are dependent upon the BGACK signal. This signal must be asserted at the correct time. The MC68008 asserts the BG signal after the assertion of the BR signal. Note, however, that BG is asserted during a processor bus cycle. The BGACK signal should only be applied after the CPU's bus cycle has finished. A simple circuit for accomplishing this task is shown in Figure 2.

The flip-flops are not enabled until BR becomes true. After BR becomes true and the MC68008 finishes the bus cycle, which is indicated by the rising edge of AS, the first flip-flop's output becomes true. The second flip-flop delays the assertion of BGACK until one clock cycle later, giving the MC68008 time to float all of its control and bus lines.

Note: For correct DMA operation the modifications shown in Appendix C must be present on the MC68008 board.

Suggested Bus Request Circuitry

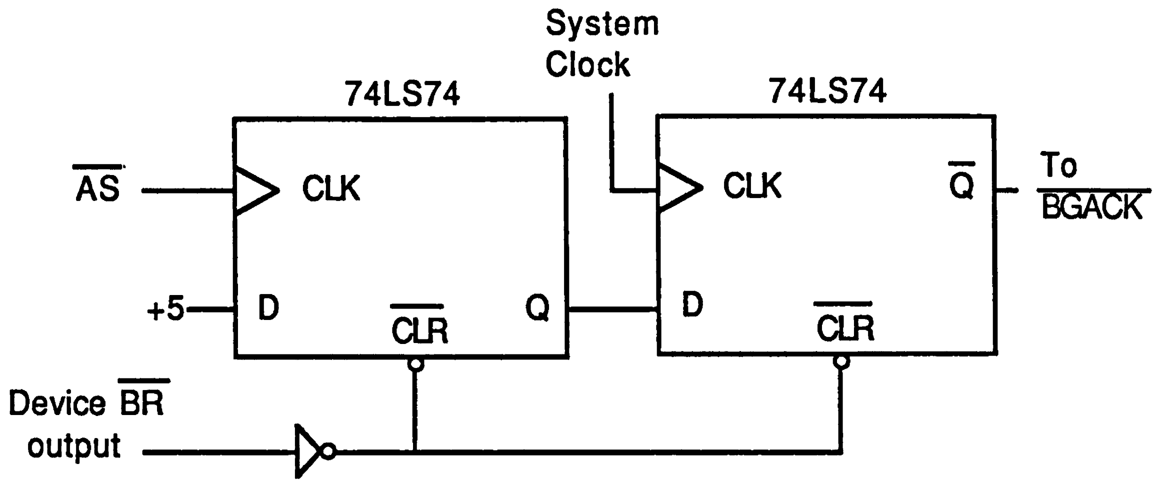


Figure 2

2.10 Buffer Board

2.10.1 Design Criteria

The buffer board is specifically designed to interface to the microprocessor board, providing easy access to the microprocessor signals. Any system built for use in a laboratory environment needs to be robust and provide a facility for fault detection. The buffer board attempts to provide both of these attributes.

2.10.2 Buffering

The microprocessor board buffers both the address and data busses with 74LS245 transceivers. These devices are capable of indefinite short-circuiting and are extremely robust. The major bus signals are buffered via a 74LS244 which has similar characteristics to the 74LS245. The use of the 74LS244 rules out the possibility of implementing a DMA interface using the buffer board. However, DMA transfers can be implemented using the microprocessor board without the buffer board. Schematics for the buffer board are shown in Figure 3.

2.10.3 Parallel Port Diagnostics

The Z80 system had two parallel ports which were prone to failure. An external cable was used to jumper the two ports together to allow a loopback test to be carried out. This arrangement, however, did not provide reliable testing. Furthermore testing was carried out only after the user suspected that one of the ports had failed.

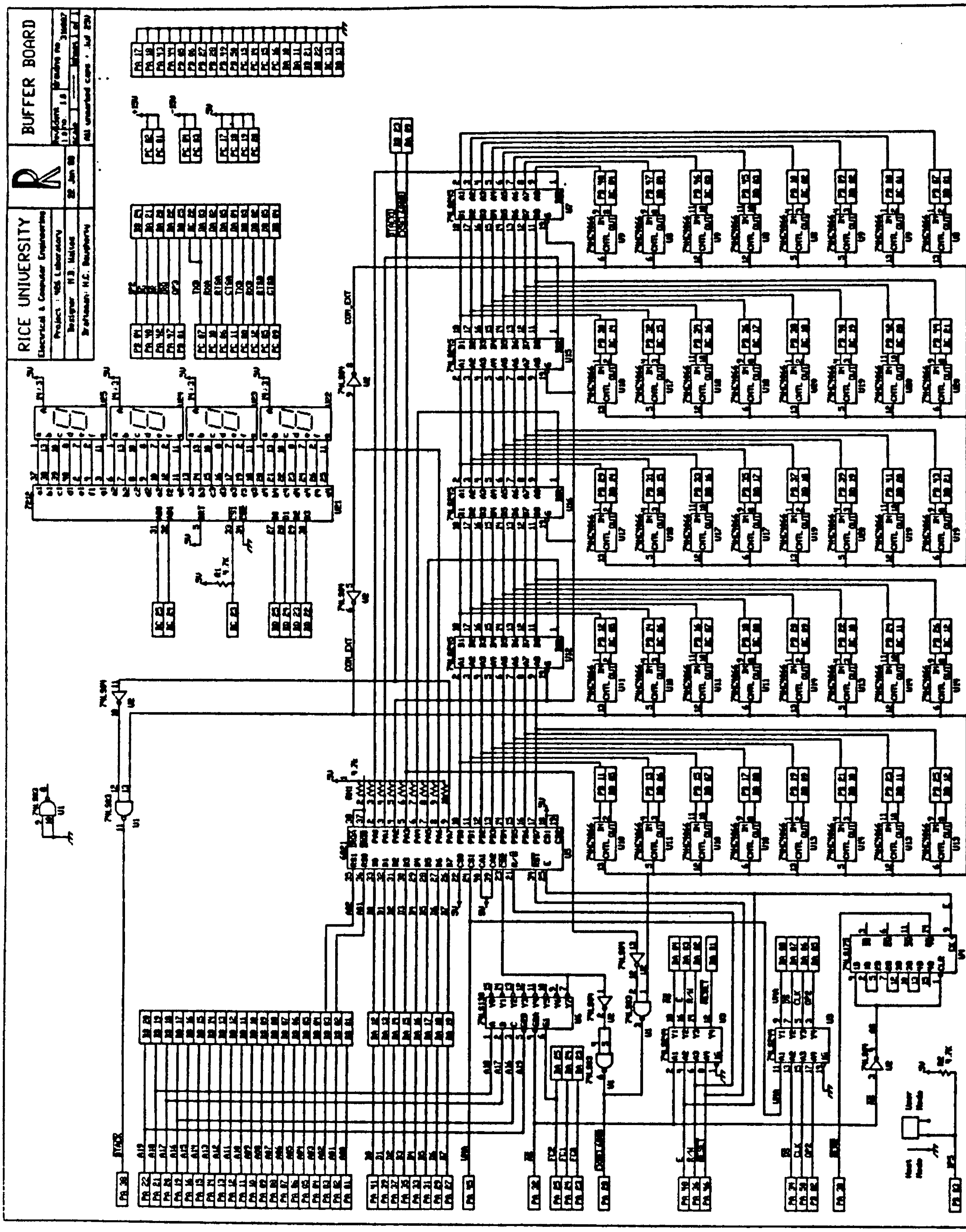


Figure 3

The major problem with the parallel ports lies with the input and output lines being rather fragile electrically. Continuous short-circuiting of these lines often causes permanent damage. Two approaches can be used to correct the problem, each having its own merits.

The first solution involves protecting the devices by buffering each pin before it reaches the external connector. The buffering provides protection for the parallel ports, and the buffers themselves are close to indestructible. This solution is probably the most elegant, although there are implementation problems. Each parallel port has two registers, a Data Direction Register and an Input/Output register. The data direction

register determines which pins on the port are assigned to inputs or outputs. Each pin can be programmed to be either. To allow the port to be used with no restrictions on which bits can be inputs or outputs, individually controlled transceivers would need to be used. The scheme would entail capturing the writes to the Data Direction Register in a latch, and then using the output from the latch to control the direction of the transceiver. A similar scheme would be used to capture writes into the ACR Auxiliary Control Register to control the direction of the handshake lines CA1,CA2,CB1 and CB2. This technique would be very robust and is completely transparent to the user software, as the hardware captures normal writes to the R6522.

The scheme was not implemented, however, due to the difficulty in acquiring transceivers with individual direction control. Texas Instruments make a device called the 74LS449 which provides exactly the desired function. However, T.I. is the only source of this part. One of the major goals in designing the system was to make sure all the parts used were readily available and could be replaced easily within the next several years.

The second solution is to provide an on-board means of checking the parallel ports. The strategy works on the principle that if the parallel ports cannot be protected, then at least we should be able to detect a parallel port failure easily and without external circuitry.

The loopback test must first isolate the signals connected to the breadboard connectors, as the key to the strategy is that diagnostic tests can be executed without the need for any disconnection of user circuitry. The isolation is implemented by the use of the 74HC4066 (now called the TLC4066, and found in the

Telecommunications and Linear Handbook). This device provides four analog switches per package. When the switch is turned off, a resistance of about 100 megohms is seen across the terminals of the switch, providing the necessary isolation for the loopback test.

The 74HC4066 has an additional desirable feature. When the switch is turned `on` the resistance across the terminals is approximately 30 ohms, which essentially provides a current limiting resistor in the case of short circuiting. During test the outputs of the R6522 were capable of continuous short circuiting through the resistor. Thus, the 74HC4066 provides both device protection and isolation for diagnostic testing. The resistor is also small enough to be insignificant when connected to TTL loads. For example even when ten normal TTL loads are being driven, the low level still remains below 0.8 volts.

Once the 74HC4066's are switched 'off', the R6522's are isolated and loopback testing can be carried out. After the testing is finished, system software must switch the 74HC4066s `on`. This is transparent to the user, but if the appropriate software is not executed , no signals will propagate through the switches. This situation can occur when using the 'Emulate' program. This problem is mentioned in the User Guide.

Diagnostic testing should provide clear and conclusive evidence on the status of a particular device. A simple loopback test does not necessarily isolate the exact cause of the failure, although an error will be detected. This problem becomes evident when the loopback function is implemented, as one port is used as an output. Data is written to the output port and read back on the input port. If the data read is different from that written, then either the output port is bad, or the input port is malfunctioning, but the user cannot determine which is causing the problem.

To attain a more exact result from the test, the four parallel ports are bussed together during loopback testing. This means that each test has several different sources of results, and an exact port and bit can be isolated.

To control the testing and switch the various devices, an MC6821 Parallel Interface Adapter (PIA) was incorporated into the buffer board. The PIA is mapped into the address space at $70000 via the 74LS138. The 74LS138 is only enabled when FC2 is high, meaning that the PIA can only be accessed in Supervisor mode. The PIA is a MC6800 device, and therefore it must use a MC6800 bus cycle when accessed. This bus cycle is selected by asserting EXTSEL6800. The 20L10 PAL also contains an equation with a term which enables the data bus transceivers when an address in the range $70000-$7FFFF is generated.

The MC6821 contains two parallel 8-bit ports, designated in the data sheet as ports A and B. Port A is used to control the enabling of the 74HC4066s, to select the direction of the 74LS245's and to sample for continuous assertion of both DTACKU and EXTSEL6800.

The pull-up resistors on the A port are used to switch the relevant signal lines to disable the 74HC4066s and 74LS245s immediately after RESET. After a hardware RESET, the PIA sets both port A and port B to input mode. The pull-ups force these lines high, and, via the inverters, the enables are negated.

The enables are also used to disable the propagation of DTACKU and EXTSEL6800. The processor executes the diagnostics, and then enables the 74HC4066s, allowing both DTACKU and EXTSEL6800 to propagate through the open collector NAND gates.

Port B is connected to the parallel port bus, giving yet another port to return the status of the parallel bus.

The Parallel port diagnostic software is described in Section 3.3.2.

2.10.4 Bus Error Generation

The MC68008 allows an exception to be invoked when an address is generated by the microprocessor which fails to make any peripheral device respond.

The BERR signal is generated via a 74LS175 (U4). The assertion of AS removes the active clear, enabling the flip-flops. Data is clocked through the flip-flops which are simply connected in series. The BERR signal will be asserted after three E clock cycles or approximately 4 microseconds, giving more than adequate time for any peripheral device to respond.

2.10.5 LED Display

The buffer board has four seven-segment LED's which can display hexadecimal digits. An Intersil ICM7212MI LED driver chip directly drives four seven-segment displays without the need for series resistors. The ICM7212MI interfaces with common anode LED's only.

The ICM7212MI provides a microprocessor type interface. Two address lines select the display to be changed, four data bits select the digit to be displayed, and a chip select enables the data to be latched and displayed. All the ICM7212MI LED control signals are available on the breadboard connector (the connectors pin-outs are shown in Appendix E). The ICM7212MI can be used via a microprocessor bus interface, via connection to one of the R6522s, or directly from external circuitry.

An LED diagnostic program is listed in Appendix B. The listing shows a simple program to drive the LED displays. Instructions on interfacing the ICM7212MI to the R6522 are also given.

2.10.6 Mode Selection Jumper

The system has two modes of operation. A jumper is provided on the buffer board to select the mode of operation in which the system executes. The jumper either shorts the bit 5 input on the MC68681 to ground when in 'host' mode or to 5 volts when in 'user' mode. During initialization, the CPU checks the status of bit 5 on the MC68681 to determine which mode of operation is selected. The modes of operation are explained in Section 3.3.

2.10.7 PCB Design

Due to fabrication costs the PCB was limited to a two layer board. The layout was drafted using an IBM PC running SMARTWORK, a low cost PCB layout program. The program allows interactive editing and simple auto-routing. After the layout is completed, it is plotted onto film which can then be directly photographed and reduced by the PCB fabricators.

During the development of the system another program called HIWIRE (both programs are written by a company called 'WINTEK') was purchased. This program has a schematic capture editor, with utilities for generating netlists both from the schematic capture program and the PCB layout program. A further utility allows comparisons of netlists. Using these software utilities, PCB layouts can be verified against their schematics.

Chapter 3

Support Software

3.1 Software Overview

The digital designer's work environment consists of the MC68008 system connected to a serial port on a Sun workstation. The Sun workstation is used to edit, assemble and communicate with the MC68008 system. The user communicates to the MC68008 system through a special program which effectively provides a development station environment.

3.2 The MC68000 Assembler

The assembler is a single pass program written in the 'C' language. The original source code was acquired from Vanderbilt University, where it had been originally released on a Gould Unix-based system. The code was extensively modified to run under UNIX BSD 4.2, as many UNIX features have changed since the original release of the Assembler in 1984.

The original definition of the Motorola MC68000 family assembly language defined in the "16-bit Microprocessor User's Manual" has changed significantly. The assembler was modified to meet the current standard. The assembler was also modified to allow for full 32-bit addressing, as the original assembler only handled 16-bit addressing.

The output code generation was also updated to use 32-bit Motorola S-record format, as opposed to 16-bit Motorola S-records. The output can be directly downline loaded to most EPROM programmers for the burning of EPROMs.

3.2.1 Implementation of the One pass Assembler

The assembler makes a single scan through the source file and then produces the output code. Labels which are undeclared during the pass are stored in a symbol table, and their values are filled in when they become known. After all the source code has been scanned, the unknown label values are backpatched. Backpatching is accomplished by storing unknown labels in a backpatch table. The backpatch table stores the symbol name, the number of bytes to be filled in during backpatching, and the address at which backpatching must occur. When the total source file has been scanned, all label values are known and the symbol table is complete. At this stage all the values of the unknown labels can now be inserted.

The output code is written into a random access file with each record number corresponding to the absolute address of the byte. The random access file provides a convenient way for providing the holes to allow backpatching to be easily implemented.

The UNIX file system provides a non-standard random access feature which the code generator relies upon. If the first statement in a source code is:

ORG $100

The assembler executes an 'lseek' call to move the file pointer to location $100. Even though, when the call is made, the file is of zero length, the UNIX filing system automatically pads in the appropriate number of bytes (*i.e.*, in this example the 'lseek' function will return a pointer to $100, and the file will appear to be $100 bytes long). The code generation could be modified so it does not use the random access file. Instead, the intermediate code could be stored in memory using calls to malloc(). This method of housekeeping would require some overhead in code segment management, but it would increase the assembler's speed. It would also make it easily transportable, as it would no longer be dependent on irregular UNIX filing system features.

The assembler was extensively modified to enable it to run on either UNIX or the IBM PC with one set of source files. Microsoft 'C' does support the 'lseek' function as described above, nevertheless many other modifications were necessary to make the assembler run on the PC. Integers default to 16 bits on the PC. Therefore all variables are now specifically declared to be either 16 or 32 bits. The assembler also uses bit-fields which are declared in the reverse order on the PC. This is managed through the use of conditional compile statements. Note, however, that bit-fields are extremely system dependent. Byte swapping is also a problem on the Intel 8086 machines. This is also managed with a conditional compile.

The assembler will also run on a Commodore Amiga and Apple Macintosh. The 'lseek' function on these systems does not behave like the UNIX implementation, which therefore restricts source programs to using no more than one ORG statement.

The one pass assembler has certain limitations in the use of addressing modes. All addressing modes can be used, only if all the constants involved in the effective address calculation are known as the text is scanned. For example the following instruction is valid:

```
move.b    $1000+$2,D0
```

The address could be declared before the instruction and the instruction would still be valid:

```
myloc equ $1000
move.b    myloc+$22,D0
```

However, the following piece of code will produce an assembly error, although it is perfectly valid assembly code:

```
move.b  myloc+$2,D0
myloc equ $1000
```

In practice these forms of addressing modes do not arise too often, and therefore they do not hinder the usefulness of the assembler.

One form of addressing which always uses a displacement offset from a particular address does cause considerable problems. If the programmer attempts to write completely relocatable code, meaning that all program variables are accessed by the Program Counter plus a displacement value, the assembler will fail to compile the source code. If all the constants pointing to variables are declared before any actual code is scanned, however, the assembler will correctly produce the output code. An example program is shown in Appendix B. The memory test program is completely relocatable and uses Program Counter displacement addressing.

It should also be noted that the listing file generated by the assembler does not necessarily contain the correct hex code, as any backpatched code will not be displayed correctly. The assembler was modified to generate a symbol table to assist in alleviating this problem.

Instructions on how to use the assembler are given in Section 5, the User Guide.

3.3 The MC68008 Monitor Program

The MC68008 system contains an 8 kbyte EPROM which stores the entire system software. The monitor program is an adaption and extension of a program called 'VUBUG', which was acquired from Vanderbilt University.

The monitor program can run in two modes, user or host mode. The buffer board contains a jumper which selects the mode. The jumper block is clearly marked on the printed circuit board. When the system is set in user mode, a terminal can be connected directly to the serial port A, and the user communicates directly with the microprocessor. This mode of operation is best suited for use with a PC running a terminal emulation program, as the monitor can read programs sent to it via an ASCII upload from the terminal emulation program. A typical terminal emulation package is 'Procomm'. 'Procomm' is particularly suitable, as a DOS shell can be opened, an edit and assemble phase can be carried out, and the output from the assembly can then be uploaded and tested.

In the host mode, the monitor program communicates with a program called 'Emulate' which provides a friendly environment to develop software. The monitor and 'Emulate' program send messages to each other via the serial link, and the MC68008 is controlled purely by the 'Emulate' program running on the UNIX system. Both the monitor and 'Emulate' programs have user documentation in the User Guide.

In host mode the monitor and 'Emulate' programs communicate using a small number of basic primitives. These are listed below:

1. Send contents of the next X bytes from location Y in the MC68008 board's memory to the 'Emulate' program.
2. Receive X bytes from 'Emulate' and place them in memory starting at location Y.
3. Execute a program on the MC68008 system.
4. Break into a program running on the MC68008.
5. Single step the MC68008.
6. Insert a breakpoint at location X.
7. Send contents of the MC68008 registers to the 'Emulate' program.
8. Receive contents of the MC68008 registers from the 'Emulate' program.
9. Run the self-test diagnostics.

Certain messages sent to the MC68008 expect acknowledgement messages, as the monitor program catches all MC68008 exceptions. For example, when the "put to memory" primitive is used, the monitor attempts to execute the command issued. If it fails due to a bus error or address error, an appropriate message is sent to 'Emulate' indicating the address of the memory error. If the command is successful, the monitor responds with a "command OK" acknowledgement. The exact syntax and the argument format for these primitives is listed in Appendix F. For more information on the message system see the source code file 'monitor.c', the module which implements the communication protocol.

3.3.1 Monitor Implementation

After reset or power-on reset, the monitor program first executes the memory and DUART diagnostics. Once the system has passed the diagnostics, all system variables are initialized. A routine called 'init' initializes the DUART. It also initializes system flags and sets up the circular input buffer. After the DUART initialization software is executed, the interrupts are enabled, allowing characters to be received over the serial line. The software decides which mode is selected by reading bit 5 of the input port on the DUART. If this bit is low, then the software selects the host mode; otherwise, it selects the user mode. A flag called 'outmod' disables all output from being transmitted when set to '1'. This is used in host mode to stop echoing of characters and other output which is not needed by the 'Emulate' program. For example, when in user mode, upon system reset, the monitor displays a "hello" message on the terminal screen. When the system is running in host mode this is not sent as 'outmod' is set to '1'.

For all future discussion let us assume we are in user mode. After the initial "hello" message is displayed, a "!" prompt is written to the terminal. The monitor then waits for a character to be typed on the terminal keyboard. The serial receiver driver is interrupt driven, and uses a circular buffer of sixteen characters. (Note also that the MC68681 DUART has a 4-byte on-board FIFO buffer). Each time a character is received by the DUART it is echoed and placed into the circular buffer. The monitor program removes characters from the buffer by calling a routine called 'getch'. This routine returns the first character received when one or more characters are in the buffer. Otherwise, it waits until a character is placed into the buffer. The serial driver supports the XON-XOFF protocol when transmitting to avoid terminal buffer overruns (the terminal or terminal emulation program being used should have XON-XOFF enabled to ensure correct operation).

The serial driver also treats the control-C character in a special manner. Whenever a control-C is received, the 'rstrt' routine is executed. This routine preserves all the registers, stops execution of any user programs, and places the user back in command mode. The 'rstrt' routine also resets all peripheral ports via the software instruction 'reset'. It then reinitializes the DUART for correct operation. When the NMI switch is depressed on the MC68008 board, 'rstrt' is also called. The control-C and the NMI switch use the same software to recover from a program abortion. As the NMI switch is connected to the interrupt level which cannot be disabled, the NMI is effectively the system RESET switch. Sending a control-C to the Monitor will stop the program, provided the interrupt vector in the DUART, and the DUART's communication characteristics have not been corrupted due to a runaway program. The control-C character stops the Monitor about 99% of the time in practice, as the microprocessor rarely runs wild for very long, since a bad instruction, bus error or address error tend to cause the program to halt.

Once the character has been fetched from the input buffer, it is checked against the command table. The command table consists of pairs of words, the first containing the character, and the second containing the word which points to the start of the routine to be executed. A word is perfectly adequate for this function, as a word can be used to point to an any address in the bottom 64k, and all the routines are in an 8k Eprom mapped at location $0. If a second EPROM were placed in the third memory socket and extra commands were added to the command table, any routines in the third EPROM could not be called directly, as this EPROM is mapped at $20000-$2FFFF which is not within the bottom 64k.

All the functions invoked by user commands run in the MC68000's Supervisor mode, allowing the user to modify the upper bits of the status register without encountering privilege errors. Normally user programs execute in the MC68000's user mode. If application programs run in user mode, software reliability is improved, as both the user program and the monitor have separate stack frames. User programs can of course run in Supervisor mode, using the 'super' trap call, which sets the processor into Supervisor mode.

The monitor provides exception handling for all the exceptions defined by Motorola. The exception handlers print messages indicating which exception occurred and any other relevant information.

Most of the interactive user commands acquire addresses from the keyboard and then execute some monitor function (*e.g.*, display memory requires a start and end address). The monitor provides several calls to fetch bytes, words and long words. These routines can be called by user programs through the use of the traps provided in the User Guide.

The number fetch routine checks input characters to ensure they are valid hex digits. If they are not, the number fetch routine displays a message indicating invalid text was entered and returns to the caller. The number fetch routine correctly handles the 'Delete' character, but all other non-alphanumeric characters cause the routine to exit.

The monitor leaves all interrupts enabled when user programs are executing. The DUART is used for communication with the terminal using channel A. The channel A receiver interrupts are handled by the monitor which stores characters received into the input buffer. If any other DUART function interrupts the monitor, control is passed to location $1102C. Only eight bytes are reserved at $1102C. Therefore, a jump instruction to the user's handler should be inserted. Similarly, control from the level 2 interrupt is passed to location $11034. The level 2 interrupt has a default handler installed which resets all the interrupts on the R6522. The monitor also has routines which will provide a circular buffer for received characters from the channel B port. An example of how these are used is given in Appendix B.

The monitor uses indirect subroutine calling to allow data to be read from either channel A or B. This effectively means that the number fetch routines can be used with both channel A and B. To use these routines, the user calls them directly (as opposed to using the TRAP), placing the address of the routine which is to be executed in A0 *i.e.*, the specific I/O handler. The user routine should then return the character fetched in D0.

The monitor source is available for modification. Its location is given in Appendix B.

The monitor source is written in the old Motorola source code. The instructions use the same syntax as the present "Motorola Standard", but unfortunately the addressing modes are different. To recompile the monitor source code, one must use the old version of the Motorola assembler. This is available only on UNIX-based machines. The monitor and assembler were developed simultaneously. Unfortunately, this meant that during the development phase the initial assembler was used to assemble the monitor.

The monitor program reads Motorola S-records. These S-records must use 4-byte addressing. Both assembler programs produce output which can be read by the monitor. The S-record is slightly adapted by the assembler to provide an easy method of setting the program counter. When an end record is sent (an 'S7' record), the address following it is the value to be loaded into the program counter. This value is generated during the assembly by the programmer specifying a label after the end statement. The format for the Motorola S-record is shown in Appendix D.

3.3.2 Monitor Diagnostics

The MC68008 system has several diagnostic routines which provide help in tracing system faults.

On power-up, the system RAM is checked with a routine which only uses registers and does not require a stack frame. If the memory check fails, the Supervisor LED flashes with a frequency of about 2Hz and a mark-space ratio of 1:1. If the memory check passes, a DUART loopback test is executed.

The DUART test connects each channel's receiver and transmitter together. Characters are transmitted and the received data is verified against the data sent. The code for this test is rather difficult to understand, as the DUART does not respond quickly to commands when asked to operate in loopback mode. The code has many software delays to allow for unknown chip characteristics, which were discovered by trial and error. The symptoms of this strange behavior are random results, whereupon the diagnostic test may run correctly only about 50% of the time. The final code has proven to be reliable and effective in testing the loopback mode. If this test fails, the Supervisor LED is flashed at approximately 1Hz with a mark-space ratio of 1:4. If either the memory or DUART diagnostics fail, the error is considered disastrous, and the Supervisor LED flashes indefinitely. Power should be removed, the offending IC should be replaced, and power should be reapplied.

When the system boots without errors, the Supervisor LED is illuminated for about half a second,and then it remains off until user commands are issued.

The parallel port diagnostic is executed whenever the 'Emulate' program is started. The parallel port diagnostic first tests the MC6821 for correct operation. Certain registers in the MC6821 are written with data, and the MC6821 is then read for correct verification of these registers. Note the MC6821 is interfaced to the microprocessor board via the 74LS245 (U5), and the MC6821's data bus is common to the data bus provided on the breadboard connector. These reads check to see if the data bus is continually being driven by external user circuitry. If the bus is continuously being driven, it has no effect on any other system components as they are isolated through the 74LS245. If the reads provide incorrect data, the MC68008 monitor sends a message to the 'Emulate' program, and the 'Emulate' program displays a message to the user indicating that the data bus is being driven at the wrong times. If the read test is passed, the parallel port diagnostic is executed.

The parallel port diagnostic firstly checks the data bits on the PA and PB ports of each R6522. This test is done in five stages. Initially Port B on the MC6821 becomes the talker (it is set in output mode) and the four R6522 ports are listeners (set in input mode). The 74LS245s are enabled, and their directions are set so that the data propagates from PB0-7 through to U7- A1-A8. The enabling and direction setting is controlled by the PA port on the MC6821. Two data bytes ($55 and $AA) are written sequentially on the MC6821 PB port, and the R6522 ports are read and verified against the written data.

The next stage involves making one of the R6522's the talker. The other R6522s remain listeners, and the MC6821 port PB is also made a listener. This is repeated until all the R6522 have been talkers. If one or more errors occur, the diagnostic reports a failure to the 'Emulate' program. The diagnostic software does not attempt to compute which port failed. It simply displays all the information in the form of which port wrote data, the value of the data, and the values read on the other ports. Typical output is shown in Figure 4. The bad port in this example always reads the wrong data when the $55 data byte is broadcast giving the vertical column under UY-A. When the bad R6522 port (UY-A) writes $55, all the other ports see $57, and this makes up the horizontal row. On analyzing the data read and written, it is clear that bit 1 of port A of chip U12 is open, and U12 should be replaced. Once the data bits have been tested, the interrupt lines CA1, CA2, CB1 and CB2 are checked. These lines are connected to the A-side of U7 and are driven using the PB port of U12 (the R6522). Each pin is programmed to cause an interrupt when a positive edge is applied to it. The positive edge is generated by software using the output port. If any pin fails the interrupt test, a message is displayed indicating which chip failed, either U11 or U12.

After the parallel ports have been checked, both DTACKU and EXTSEL6800 are sampled using port A on the MC6821. If either one is low, a message is displayed informing the user of which line is continuously being driven low. This check catches incorrect wiring of external circuits or incorrect logic design.

If all the above tests are carried out without errors, the monitor returns a message to the 'Emulate' indicating that the diagnostics were passed. If any errors occur, these are sent to 'Emulate', and 'Emulate'

Figure 4

Diagnostic Report Follows:

Parallel Port Diagnostics (UX -x ->U11 UY -x ->U12)

Control Port	UX -A	UX -B	UY -A	UY -B
Wrote 55	Read 55	Read 55	Read 57	Read 55
Wrote AA	Read AA	Read AA	Read AA	Read AA
Read 55	Wrote 55	Read 55	Read 57	Read 55
Read AA	Wrote AA	Read AA	Read AA	Read AA
Read 55	Read 55	Wrote 55	Read 57	Read 55
Read AA	Read AA	Wrote AA	Read AA	Read AA
Read 57	Read 57	Read 57	Wrote 55	Read 57
Read AA	Read AA	Read AA	Wrote AA	Read AA
Read 55	Read 55	Read 55	Read 57	Wrote 55
Read AA	Read AA	Read AA	Read AA	Wrote AA

Interrupt OK

Interrupt OK

Interrupt OK

Interrupt OK

Interrupt OK

Interrupt OK

Interrupt OK

Interrupt OK

Emulator aborting program.

displays them on the terminal screen. If the system is being used in user mode, all error messages are also written to the terminal.

3.4 The 'Emulate' Program

The 'Emulate' program attempts to provide the user with a friendly environment in which to develop programs. The program provides an environment similar to that of an HP64000 development workstation. The 'Emulate' program is a modification of the 'Z80 Emulate' program which was originally written by Dr. J. D. Wise.

When the program is executed, it first establishes a communication link to the MC68008 via the serial port. Once the link is established, the 'Emulate' program issues a command to the MC68008 to execute the parallel port diagnostics. The MC68008 executes the diagnostics. If any errors are found, the 'Emulate' program displays them and aborts; otherwise, it displays the main menu.

The menu allows various functions to be selected using single key strokes,and an overall command line is built up. Once the command line is ready for execution, the user types a return character to invoke the command. The 'Emulate' program allows in-line loading of code, displaying of memory and registers, disassembly, insertion of breakpoints, single stepping and symbol table look-up. The 'Emulate' program also intercepts and displays all exceptions generated by the MC68008.

'Emulate' allows the user to communicate with the MC68008 directly using the serial port, during the execution of user programs. The emulation software effectively makes the UNIX terminal become a terminal connected to the system's serial port. The monitor also contains traps which send and receive characters via the serial port, so implementation of interactive programs is very simple.

'Emulate' is available for both Sun workstations (effectively, any machine running UNIX) and IBM PCs. The UNIX version is written entirely in 'C' and should be easily transportable to most UNIX implementations. (The only compatibility problems are in the use of UNIX I/O calls which differ slightly with each specific implementation. 'Emulate' presently runs under HP-UNIX, PYRAMID BSD 4.2, and Sun UNIX BSD 4.2.) The IBM version is written mainly using Microsoft 'C', although the serial communication routines are written in assembly language. The IBM PC version requires a fully compatible PC for correct operation.

For instructions on how to use 'Emulate' see the User Guide in chapter 5.

Chapter 4

System Cost

During the R and D and production, careful emphasis was placed on minimizing the cost of the system. The total cost of the system's R and D was approximately $2,000, of which about $500 was spent on testing various power supplies.

The cost of producing the forty complete systems was $13,500 excluding R and D. It is important to note that approximately $6,000 of this expenditure was on power supplies for the system. These calculations do not include labor costs, although they do include printed circuit board and system assembly costs.

Overall, this means that each system cost approximately $380 including R and D. Of the $380 spent on each system, about $150 accounts for the power supply, the rest being charged to the actual hardware. A parts list is given in given in Section 4.1.

The system also includes a substantial amount of software which transforms the microcomputer into a useful development system. In reality, the software development cost would be a considerable factor in the overall cost of the system.

4.1 Bills Of Materials

Microprocessor Board Bill-of-Materials

site	part	value	manufacturer	supplier	specification
C1		2.2uf	Panasonic	Digi-Key	tantalum, >= 10v
C2		10uf	Panasonic	Digi-Key	tantalum, >= 10v
D1	ECG-109		Sylvania	fvd<=2v, piv>10v	
D2	Red LED		Panasonic	Digi-Key	any LED ok
D3	1N4148				alternate: 1N914
PA	3433-6202	3M	Novell	50 pin ribbon	
PB	3433-6202	3M	Novell	50 pin ribbon	
PC	3428-6202	3M	Novell	20 pin ribbon	
R1	1k			Digi-Key	1/4watt, 20%
R2	1M			Digi-Key	1/4watt, 20%
R3	4.7k			Digi-Key	1/4watt, 20%
RN1	2.2k		Panasonic	Digi-Key	Pull-ups to 5v on pin 1
RN2	4.7k		Panasonic	Digi-Key	Pull-ups to 5v on pin 1

site	part	value	manufacturer	supplier	specification
U1	74LS245		National	Digi-Key	
U2	74LS245		National	Digi-Key	
U3	MC68008PS		Motorola	Active	
U4	16L8		MMI	Quality	any 16L8 ok
U5	74LS245		National	Digi-Key	
U6	74LS245		National	Digi-Key	
U7	OSC 7.3728Mhz			FOX (F100 series)	Active
U8	2764		Intel	PROM 8k x 8	
U9	2764(6264)		Intel	PROM(RAM) 8k x 8	
U10	2764(6264)		Intel	PROM(RAM) 8k x 8	
U11	6522		Rockwell	Active	use 65NC22 if avail.
U12	6522		Rockwell	Active	use 65NC22 if avail.
U13	20L10		MMI	Quality	any 20L10 ok
U14	74LS73		National	Digi-Key	
U15	DS1488		National	Digi-Key	RS232 level tx
U16	MC68681P		Motorola	Active	
U17	74LS132		National	Digi-Key	
U18	DS1489		National	Digi-Key	RS232 level rx
U19	74LS74		National	Digi-Key	

Buffer Board Bill-of-Materials

site	part	value	manufacturer	supplier	specification
BA	Breadboard Conn.			A.P. Products	Marshall
BB	Breadboard Conn.			A.P. Products	Marshall
BC	Breadboard Conn.			A.P. Products	Marshall
BD	Breadboard Conn.			A.P. Products	Marshall
DA	RS232 Conn.		Tex-Techs,Inc.	Digi-key	
DB	RS232 Conn.		Tex-Techs,Inc.	Digi-key	
RN1		2.2K	Panasonic	Digi-Key	Pull-ups to 5v on pin 1
U1	74LS03		National	Digi-Key	
U2	74LS04		National	Digi-Key	
U3	74LS244		National	Digi-Key	
U4	74LS175		National	Digi-Key	
U5	MC6821P		Motorola	Schweber	
U6	74LS138		National	Digi-Key	
U7	74LS245		National	Digi-Key	
U8	74HC4066		T.I.	Schweber	
U9	74HC4066		T.I.	Schweber	
U10	74HC4066		T.I.	Schweber	
U11	74HC4066		T.I.	Schweber	
U12	74LS245		National	Digi-Key	
U13	74HC4066		T.I.	Schweber	
U14	74HC4066		T.I.	Schweber	
U15	74LS245		National	Digi-Key	
U16	74LS245		National	Digi-Key	
U17	74HC4066		T.I.	Schweber	
U18	74HC4066		T.I.	Schweber	
U19	74HC4066		T.I.	Schweber	
U20	74HC4066		T.I.	Schweber	
U21	ICM7212IM		Intersil	Schweber	

site	part	value	manufacturer	supplier	specification
U22	7SEG LED		H.P.	Schweber	Common Anode
U23	7SEG LED		H.P.	Schweber	Common Anode
U24	7SEG LED		H.P.	Schweber	Common Anode
U25	7SEG LED		H.P.	Schweber	Common Anode

4.2 Fabrication

The Printed Circuit Boards were made using a program called Smartwork. Smartwork allows the PCB to be designed interactively on an IBM PC. The final artwork is plotted on an HP plotter using vellum film. The artwork was constructed into a PCB by a company called HEDCORE. The PCB is manufactured from the photographed artwork. HEDCORE's address is:

5514 Mitchelldale

Houston, TX 77092

The boards were stuffed and assembled (*i.e.*, wave soldered) by a company called M. & R. , their address is:

4910 Wright Rd

Suite 100

Stafford, TX 77477

CHAPTER 5

USER GUIDE

5.1 The Rice MC68008 Computer System

5.1.1 Introduction

The Rice MC68008 microcomputer system was designed specifically for use in the ELEC 426 course, which teaches the fundamentals of digital system design. The system is comprised of a twin board microcomputer based around the Motorola MC68008 (an MC68000 with an eight bit data bus), a breadboarding chassis containing a switched mode power supply, breadboards for prototyping circuits, and switch modules furnishing LED outputs and switches. The user communicates to the microcomputer by using software on the host computer (SUN 3) which in turn communicates via an RS232 link to the microcomputer.

5.1.2 System Organization

The MC68008 is a single board microcomputer which contains a Motorola MC68008 microprocessor running at 7.3728 MHz, 8 kbytes of EPROM, and 8 kbytes of static RAM, two Rockwell 6522 parallel port devices, a Motorola 68681 DUART (Dual Universal Asynchronous Receiver Transmitter), RS232 level converters, and discrete logic ('glue') which allows these devices to talk to each other. The MC68008 sits on top of the mother board, and connects to the mother board via 3 ribbon cables.

The mother board contains buffers and circuitry which aid internal diagnostics, four breadboard type connectors which contain all pertinent signal lines, four seven-segment hexadecimal displays, and two DB25 connectors located at the rear of the board.

5.1.3 Address Space Management

The memory map of the MC68008 is shown in Table 1. Note that the MC68008 is purely memory mapped and therefore, all I/O devices are in the memory address space. On the MC68008 all devices are allocated a 64k byte address block (the top four lines A16-A19 are decoded). This means that the I/O devices have folded memory addresses; *.e.g.*, the transmit buffer on the DUART is located at $30006, $30026, $30046, *etc.* The bottom half of the address space is already assigned (addresses $0-$7FFFF) and cannot be used by the developer. The top half of the address space (addresses $80000-$FFFFF) is unused and should be used for development (see note 1).

Table 1

Memory Map

8K bytes system Eprom	$0-$FFFF
8K bytes RAM	$10000-$1FFFF
8K bytes RAM or Eprom (unused)	$20000-$2FFFF
M68681 DUART	$30000-$3FFFF
R6522 Parallel Port	$40000-$4FFFF
R6522 Parallel Port	$50000-$5FFFF
SYSTEM USE ONLY	$60000-$7FFFF
USER ADDRESS SPACE	$80000-$FFFFF

Note: The System uses locations $11000-$11500 for internal housekeeping. The user must not write to these locations. Communication to the host is done via channel A on the DUART. Channel B is available for use by the user.

5.1.4 I/O Registers

The system has two parallel ports and a DUART, which are selected as shown above. For compatibility with the MC68000 the devices are wired so that registers are addressed on even boundaries.

The exact location of each register is shown below:

DUART Registers		R6522 Registers (X)	
MR1A	$30000 *	ORB	$40000
SRA	$30002 *	ORA	$40002
CRA	$30004 *	DDRB	$40004
RBA/TBA	$30006 *	DDRA	$40006
IPCR	$30008	T1C-L	$40008
ISR	$3000A	T1C-H	$4000A
CUR	$3000C	T1L-L	$4000C
CLR	$3000E	T1L-H	$4000E
MR1B	$30010	T2C-L	$40010
SRB	$30012	T2C-H	$40012
CRB	$30014	SR	$40014
RBB/TBB	$30016	ACR	$40016
IVR	$30018	PCR	$40018

OPCR	$3001A	IFR	$4001A
OPR(SET)	$3001C	IER	$4001C
OPR(RESET)	$3001E	ORA	$4001E

Registers marked with a '*' are for system use only.

The other R6522 (Y) is located with a base address of $50000, and its registers are addressed identically to those of (X).

5.1.5 Interrupt Structure

The MC68008 has three levels of interrupts available. The highest level (which is Non-maskable) is used for the soft reset and is invoked by pressing the push button switch located on the MC68008 board. The second level is used for DUART interrupts and should never be disabled via software. The lowest level is used by the R6522 parallel port chips and may be disabled via software. (This means the user must only set the interrupt mask to either level 0,1,2,3, or 4. Note, however, that on the MC68008 only levels 0,2,5,7 are of any relevance as IPL0 and IPL2 are internally tied together.) The default software environment has all interrupts enabled, so unless one specifically wants to disable and re-enable interrupts, the interrupt mask need not be altered (Programs must be running in Supervisor mode to alter the mask). Three lines are connected to the lowest level interrupt, the two R6522 interrupt lines and an external interrupt signal which is brought out to the connector. If one intends to use more than one source of interrupt generation then one must poll to determine the source.

Table 2

Interrupt Structure

Priority Levels

Priority Level	Associated Device	
Level 7 (NMI)	NMI Switch	Highest level
Level 5	DUART	↓
Level 2	R6522 and external interrupt line	Lowest level

Notes

1. Care must be exercised when interfacing devices into the memory map. During an interrupt acknowledge cycle the user must ensure that his devices are not selected. See the MC68008 manual for more details.

2. DTACKU has a propagation time of 48 nanoseconds. This timing requirement is important due to the asynchronous setup time of DTACK. (see timing requirement 47 in the MC68008 Databook.)

3. **USER PROGRAMS MUST INITIALIZE THE STACK POINTER !!**

5.2 THE MC68000 ASSEMBLER

5.2.1 Introduction

The MC68000 cross assembler is a simple, one pass assembler that runs on any machine which supports 'C'. It accepts as input Motorola source code as defined in the Motorola "16 bit Microprocessor Handbook". Certain other pseudo-instructions are also accepted. These are described below.

5.2.2 Running the Assembler

The assembler is run by typing:

asm filename

The assembler will accept any filename, with any extension, but extensions must be specified. When executed, the assembler will produce two output files:

hex - this contains the hexadecimal machine code which can be directly loaded into the emulator or resident monitor.

hex.sym - this contains a symbol table which is useful for debugging.

The assembler supports the following options. They are entered as flags on the command line in the usual UNIX fashion:

-o name - this explicitly names the output file to 'name' rather than defaulting to 'hex'

-l - this option produces a listing file which is sent to the standard output device (normally the terminal). To place the listing into a file use redirection,

i.e., asm -l file >listing

-c - the -c option adds object code to the output listing. Note the -c option automatically invokes the -l option.

-s - the -s option overrides internal boundary alignment. When either string or byte constants are specified, the assembler by default realigns the current location counter to an even value if it was odd at the end of the string or byte declaration. This is annoying if the user wishes to mix bytes and strings together as he must count the number of string characters to ensure an even number. If the -s option is specified, then this internal boundary alignment is switched off and anything can be declared on any boundary. (If care is not used address, however, errors will be generated, as the MC68000 must read word and long word values from an even boundary.) An assembler pseudo-instruction is provided to assist in this alignment, when using the -s option. At any point in the source code the instruction 'align' can be inserted, to realign the current location counter to an even value.

5.2.3 Addressing Modes

The MC68000 assembler supports all the regular MC68000 addressing modes as shown below:

Addressing Mode	Mnemonic
Dn	Data register direct
An	Address register direct
(An)	Address register indirect
(An)+	Address register indirect with post-incrementing
-(An)	Address register indirect with pre-decrement
d(An)	Address register indirect with displacement
d(An,Xn)	Address register indirect with index and displacement
label	Absolute word or long
$integer	Absolute

d(PC)	Program counter with displacement
d(PC,Xn)	Program counter with index and displacement
Immediate	Immediate data

Where:

n denotes the integers 0-7 for registers

X denotes either an address or data register

label is any program label and will result in either a long or word offset as appropriate. Note: labels must be declared with a ':' after the label name.

For example: here: bra here

A long offset is always assumed for a forward reference. Labels may have simple arithmetic associated with them. Simple implies addition and subtraction only.

here+2 is valid

here*2 is invalid

Immediate operands are specified by preceding the value with a '#' and have the following syntax:

#$integer	- hex constant
#integer	- decimal constant
*	- current location counter

<u>Examples</u>

```
move.b    #$FF,D0     ; move hex FF to D0
move.b    #255,D0     ; move hex FF (255 decimal) to D0
move.b    $4(A0),D0   ; move contents of location pointed to by
                      ; A0+4 to D0
```

Pseudo-Instructions

The following pseudo instructions are supported:

Mnemonic	Arguments	Use
org	effective address	set location counter to address in argument
equ	effective address	equate symbol with address in argument
dc	count,constant	declare count locations with value constant
db	count,constant	declare count bytes with value constant
ds	"string"	declare a string
end	effective address	terminate assembler input and set intial program counter to effective address on load
align	none	align current location counter to an even boundary
exit	none	trap 0

; this trap forces an exit to the monitor. All user programs should use this trap to return
; control to the monitor

getb	D0	trap 1

; reads a hex byte (two ASCII hex digits, then converts to real hex) from the keyboard,
; returns it in D0

getw	D0	trap 2

; reads a hex word from the keyboard, returns value in D0

getl	D0	trap 3

; reads a hex long from the keyboard, returns value in D0

writb D0 trap 4

; writes the byte value in D0 onto the screen translating into ASCII hex digits first

; for example, if D0 contains A5 this routine will write A5 onto the terminal screen

writw D0 trap 5

; writes the word value in D0 onto the screen translating into ASCII hex digits first

writl D0 trap 6

; writes the long value in D0 onto the screen translating into hex digits first

getch D0 trap 7

; read a character from the keyboard return ASCII value in D0. This read is a blocked read,

; the routine will return only after a character has been typed

writs A0 trap 8

; this routine writes a string onto the screen. A0 is passed to the routine containing the

; address of the first character of the string. The string must be terminated by a byte

; containing zero

writc D0 trap 9

; writes the value of D0 onto the screen. No ASCII translations takes place. For example, if

; you pass this routine with a value of 65 (decimal), it will print an 'A' on the screen

crlf none trap 10

; this routine simply writes a newline on the terminal screen

super none trap 11

; this trap places the MC68008 into supervisor mode allowing execution of privileged

; instructions

The traps are self explanatory in that all arguments are passed and returned in D0 with the exception of crlf which has no argument, super which has no argument, exit which has no argument and writs in which A0 passes the address of the string to be written. All registers except the registers containing the parameters and D0, are saved by the trap handlers.

General Notes

Important: All instructions must be in lower case and registers must be in upper case. Symbols are also case sensitive.

move.b	D0,D1	Legal
MOVE.B	D0,D1	Illegal
move.b	d0,d1	Illegal

The assembler allows the user to specify byte, word or long instructions by adding ".b", ".w" and ".l" to any instruction which allows different attributes. Branch instructions may specify ".s" for a short (byte) displacement. The pseudo instruction 'dc' also supports attributes, so words and long words can be specified by using ".w" and ".l", respectively. String constants must be defined by surrounding quotes and are constrained to thirty bytes. The backslash character allows control characters to be inserted into strings. The allowable characters are shown below:

\r - carriage return

\t - tab

\n - linefeed

\c - formfeed

\0 - ASCII value zero. This is important, as it is used to terminate strings for printing by the trap 'writs'.

The assembler also has built in some constants which are preassigned to assist in program development. These are listed below:

URAM	- $10000	This is the address of the first free space for user programs.
USTK	- $1102C	This is the address of the recommended user stack space.
INTI	- $11034	The address of the location in which the Level 2 interrupt jumps to.
INTD	- $11026	The address of the location in which the Level 5 (DUART)

interrupt jumps to when a DUART interrupt occurs. Note that when an RRDYA interrupt occurs control is NOT passed to this location.

These are used just as any other constant would be used. *e.g.*, org URAM
Both INT1 and INTD must be jump instructions to the user's specific handler, as only 8 bytes are reserved for each.

An Example Program

```
        org     $10000              ; user ram
start:  lea     $1102c,A0           ; load User stack pointer via A0
        move    #2,D7               ; set count for loop to 2
lab:    lea     mess,A0             ; put address of first char in string
        writs                       ; write string onto screen
        dbf     D7,lab              ; repeat loop until D7 is $ffff ( -1)
                                    ; i.e., we will loop 3 times even though
                                    ; we loaded 2 into D7
        exit                        ; pass control back to monitor
mess:   ds "Hello there\n\r\0"      ; define string, terminate with 0 for
                                    ; writs
        end     start               ; set PC to label `start', so after loading PC
                                    ; will already be loaded with the starting
                                    ; address of the program
```

5.3 OWLBUG

5.3.1 Introduction

The Rice University MC68008 system has a ROM resident monitor called Owlbug. Owlbug is a simple and easy-to-use monitor which allows interactive debugging of programs. Owlbug offers the ability to modify registers, to insert breakpoints, and to single step instructions, and to trace programs. It also provides full exception handling of processor errors, such as address and bus errors.

5.3.2 Using the Rice MC68008 system and Owlbug

The Rice MC68008 system can be run using any Personal Computer which has terminal emulation software. The Rice MC68008 system uses the DB25 connector closest to the ribbon cables to communicate to the terminal. This DB 25 connector is configured as a computer which means that the pinout is as follows:

Pin 2	Receive Data
Pin 3	Transmit Data
Pin 7	Logic Ground

Once a suitable cable is connected from the PC to the MC68008, the PC should be booted and the terminal emulator executed. When the terminal emulator is running the data communication parameters should be set to:

1) 9600 Baud

2) 7 bits

3) 1 Stop bit

4) Even parity

5) Select (if available) XON/XOFF protocol. This will stop Owlbug overrunning the PC's internal receive buffer.

Now switch on the MC68008 system. A message saying "OWLBUG Monitor Version 5.5 Jan 88" should appear and then a prompt of "!". At this stage Owlbug is ready to accept commands.

The MC68008 assembler is available running under UNIX, MSDOS (IBM PC compatibles), Macintosh and AmigaDos. Suggested terminal emulators for these systems are:

MSDOS - Procomm, Kermit.

Mac - Versaterm, Macterminal.

Amiga - VT100, Handshake.

Instructions

Notice the following abbreviations:

<sp> - a space

<cr> - a carriage return

addr - a hexadecimal address ranging from $0 to $FFFFFFFF (leading zeros are not needed)

xx - a hexadecimal byte (leading zeros are not needed)

yy - a two letter string

Owlbug is case insensitive.

Commands supported by Owlbug are listed below:

Modify memory

Command	Action
m<cr>	Start memory mode.
m<sp>addr<cr>	Start memory mode at addr.

Once in memory mode the following commands are available:

Command	Action
.addr	Set pointer to addr.
=xx	Store value xx at address in pointer.
,xx	Increment pointer and store xx.
+	Increment pointer.
-	Decrement pointer.
q	Exit memory mode.

Display memory

Command	Action
d<cr>	Display the next eighty bytes from memory pointer.
d<sp>addr<cr>	Display the next eighty bytes from address addr.
d<sp>addr1,addr2<cr>	Display memory from addr1 to addr2 inclusive.

Notice that the memory pointer is saved, so the following command sequence would render eighty bytes to be displayed from address $1000, and then a further eighty bytes would be displayed from address $1050.

d<sp>1000<cr>

d<cr>

Load Program

Command	Action
l<cr>	Start program load. The loading program expects to see a

Motorola S-record format supporting 32 bit addressing. The end record start address is loaded into the program counter ready for the user to run the program. If the MC68008 is being used in stand alone mode, the procedure for downline loading is:

1) Type 'L<cr>' The MC68008 will respond with a message saying 'Loading...'.

2) Select the menu item on your terminal emulator for ASCII upload, and type in the file name.

3) The MC68008 should respond with 'Load done' and it will display the value of the program counter.

If for any reason this fails, type Ctrl-C, and the MC68008 will respond 'Stopped'. Then retry the above procedure.

l<sp>addr<cr>	Start program load, but offset each record by addr bytes. The value for the Program Counter which is sent in the end record is also offset by addr.

Programs compiled using the MC68008 assembler 'asm' will download into Owlbug via the load command. The author is not aware of any other assemblers that produce the right output code for direct loading into Owlbug, although a simple filter is available for the Commodore Amiga to allow output from the Metacomco assembler and linker to be translated into Motorola S-record format.

Register Modification and Examination

Command	Action
r<sp>yy	Start register mode, and set register to be modified to yy, where yy may be either: SR, PC, D0-D7,A0-A7.
.yy	Set register to be modified to yy.
=addr	Set register to addr. Notice that the SR register only takes a 16-bit value (in the range $0-$ffff).
<cr>	Display all register values.

Breakpoint Instruction

Command	Action
b+addr	Insert breakpoint at address addr.
b-addr	Delete breakpoint at addr.
b<cr>	Display all breakpoints.
b#	Delete all breakpoints.

N.B. The present monitor supports only two breakpoints.

Running Programs

Command	Action
g<cr>	Start program from address stored in Program counter.
g<sp>addr	Start program from address addr.

USER PROGRAMS MUST INITIALIZE THE STACK !

This can be done by the following code:

```
lea USTK,A7    ; USTK is an assembler constant pointing to a user stack area.
```

Tracing and Stepping programs

Command	Action
t+	Start trace. This turns tracing on. When the program is executed via g, tracing will be displayed.
t-	Switch trace mode off.
s-	Switch single step mode off.
s+	Switch single step mode on. Similarly to t+, the g command must be used to actually single step the program.

Copy memory

Command	Action

c<sp>addr1=addr2,addr3<cr> Copy memory from locations addr2 through addr3, starting at location addr1.

Help can be obtained by typing a '?' at the prompt.

5.3.3 Error handling

Owlbug supports error handling for all the MC68008 exceptions, which include address and bus errors, illegal instructions, privilege violations, divide by zero, CHK, TRAPV, and spurious interrupts. Appropriate messages are written to the terminal via the specific trap handlers. On an exception Owlbug first saves the user registers, and then prints the appropriate message. The registers can then be viewed via the register command.

5.3.4 Stopping Owlbug

As Owlbug's input is purely interrupt driven, programs can be stopped by typing Ctrl-C. Owlbug responds with a message 'Stopped', and similarly to the exception handling, all the registers are saved and can be viewed via the register command. In some cases runaway programs may destroy the interrupt vector and Ctrl-C may have no effect. In these situations Owlbug can be stopped by pressing the NMI switch on the MC68008 board. This will cause a non-maskable interrupt to be generated. The contents of the registers will be saved and Owlbug will return with the message 'Stopped'. The latter procedure should never fail to bring Owlbug back to life, but if for some unknown reason it fails, cycle power on the MC68008.

5.4 USING THE MC68000 EMULATE PROGRAM

5.4.1 Introduction

The MC68000 Emulate program provides a pleasant environment in which one can communicate with the Rice MC68008 system. It facilitates in the downline loading and debugging of programs for the Rice MC68008 system. It was designed to Emulate the HP64000 "Emulate" function as closely as possible given the implementational constraints.

5.4.2 Running the Emulator

The emulator is executed by typing: em

The emulator immediately attempts to establish communication with the Rice MC68008 system. If it cannot establish communications due to the MC68008 system being switched off, it will display a message requesting that the MC68008 system be switched on. When the MC68008 is switched on and the communication link established, internal diagnostics are run. If the system fails its internal diagnostics, it will display a diagnostic report on the terminal. Hopefully, the diagnostics should aid the instructional staff in fixing the system. Once the system has passed the internal diagnostics, a menu of commands will appear at the bottom of the screen. Only a subset of the available commands on the HP64000 are available in Emulate.

5.4.3 Interaction with Emulate

The command syntax for Emulate is essentially drawn from the HP64000. As a command is processed, a menu on the bottom line of the screen indicates the options available at this level. The allowable options are represented in two forms:

1) <letter>=<name> - *e.g.*, d=display, typing <d> will cause <display> to be added to the command line.

2) <ADDR> or <FILE> - a numeric (hexadecimal of course) or a file name may be entered.

As the command is processed through the various levels of menus, it appears on a line above the menu. The various menu levels can be represented by a tree structure. When the command is complete, it can be executed by typing RETURN. Backspacing over a command word moves the user back one level higher in the tree, and control-U returns the user to the root level.

5.4.4 Numeric Input

All addresses and data values are in hexadecimal. The only exception, is the number of steps command, which uses decimal for user convenience.

Commands

d=display

The display command is used to view the contents of memory locations and registers.

m=memory

The display memory command displays the contents of the specified memory locations.

i=io port

The display i/o port command displays a single memory mapped location.

<ADDR>

If a numeric quantity is entered, it is taken to be the beginning of the range of locations to be displayed. If none is given the default is the previous value.

m=mode

This allows the user to select one of the two modes below. If this is omitted the default mode is absolute.

a=abs

If absolute mode is specified, memory contents as hexadecimal numbers and their equivalent ASCII characters are displayed on the right of the screen. All control and non-printable characters are displayed as '.'.

m=mnem

If mnemonic mode is specified, memory contents are displayed as MC68008 instructions.

r=registers

The display registers command displays the contents of the MC68008 registers .

e=end

The end emulation command returns control to UNIX, the host operating system. Note when you type 'e' the message 'end emulation' appears and to execute the command (as with all commands) you must hit return.

l=load

The load command loads the specified UNIX file into the memory of the Rice MC68008 system. The file must be in four byte address S-record format. If the source file included a label as part of the end statement, the value of this label will automatically be loaded into the MC68008 Program counter. The program is then executed by selecting the run command and typing <return>.

<FILE>

The file prompt is asking for a file name. Type in the relevant file name.

m=modify

The modify command allows you to set the contents of memory locations or registers.

m=memory

The modify memory command sets the contents of the specified address to the given value. The previous contents of the location are displayed in brackets after the address. Only single locations may be changed.

r=register

The modify register command sets the contents of the specified register to the given value. The previous contents of the register are displayed in brackets. Either upper or lower case letters may be used to select the registers.

i=io port

The modify i/o port command allows a single memory mapped location to be changed. This is useful for changing the contents of peripheral devices, as this command does not read the location first, as with the modify memory command.

r=run

The run command executes a MC68008 program and allows an optional breakpoint to be added.

f=from

From allows the user to specify the address at which execution is to begin. If this is omitted (RETURN is typed), the current value of the program counter in the MC68008 is used.

u=until

Until allows the user to specify a breakpoint address in which control will be passed back to the monitor when this address is executed. The address obviously must be in RAM, it must be the **first byte of an instruction** and on the MC68000 it must be **an even address.**

s=step

The step command allows single stepping and tracing of MC68008 programs. After each step all the registers and the next instruction are displayed on the terminal.

<# STEPS>

The number of instructions to step before returning control to the monitor may be specified (in decimal). If this is omitted a single instruction is executed. When control is returned to the monitor, an additional number of steps may be specified. Typing a space will cause Emulate to execute a single instruction, and pressing return places you back at the top command level.

f=from

From allows specification of the address at which stepping is to begin. If this is omitted the current value of the program counter is used.

v=value of

This command allows you to find the value of a symbol used in the source code. Note when the hex file is loaded, symbol table information is loaded via a file with the same name as the hex file, but with an extension of.sym. If you rename the hex file and you wish to use symbol table information you must also rename the.sym file.

<name>

Type in the name of the symbol and the value will be displayed on the screen.

Notes about the 'Emulate' Program.

Programs running under Emulate can be stopped by typing control-C. This stops the program in the MC68008, and Emulate returns a register dump to the user. One side effect of a control-C is that the MC68008 runs a software 'RESET' instruction which resets all the peripheral ports. Both the registers and memory are unaffected by the 'RESET' instruction.

Also, the software which enables the 74HC4066 switches is only executed when entering 'Emulate'. If the user switches off the MC68008 system and does not terminate the 'Emulate' program, when power is reapplied to the MC68008 system the 74HC4066 switches will still be disabled. To ensure this does not happen, always exit 'Emulate' when powering down the MC68008.

Chapter 6

Conclusions

The MC68008 system has now been used successfully in the laboratory for approximately two months. The system has proved to be reliable and robust in the laboratory environment. A few very minor bugs were discovered in the software. This is inevitable with any new system. All the known software bugs have been fixed and the software appears to be solid.

The MC68008 system provides a low cost development station in which both hardware and software can be developed. The MC68008 system allows maximum hands on experience in the laboratory environment, as dozens of MC68008 systems can be bought for the cost of a single commercial development workstation. The system is very easy to use, which is extremely important, in contrast to many development systems which require large amounts of documentation and laborious manual reading. All the system software is documented in about ten pages of text.

As the developer uses the Sun workstation for file management and editing, most developers already feel comfortable in using a system editor (of their choice) and file manipulation commands. The 'Emulate' program is menu driven, so that very little system dependent knowledge is required to use the whole system.

6.1 Further Software Development

There are, as always, several ways in which the software could be enhanced. Some possibilities are discussed below.

The present assembler has no object module representation, so code cannot be linked together. This means that all programs must be written in assembly language. The ability to link software could be achieved by writing a utility to convert a Sun executable image into a Motorola S-record format. This would allow high level language programs written in 'C' and 'Pascal' to be converted and run on the MC68008 system. Alternatively the load module in 'Emulate' could be altered to read Sun executable images directly. The 'Emulate' program could be added to indefinitely, with features like in-line assembly, a shell command, multiple windowing, macro files, and numerous other features. The 'Emulate' program presently drives a Sun workstation console, as well as the HP2391A terminals. Other terminal drivers could be added (the Sun console emulates a VT100 terminal protocol). Use of the Sun console allows easy access to edit, assemble and Emulate shells. Windowing software would also be easily implemented on a Sun workstation.

Overall, the MC68008 system has proved to be ideally suited to laboratory use, providing essential software development features. The system is low cost and easily maintainable. The system can be used in two modes, either with the host software or stand alone. The stand alone software offers similar features to those available via 'Emulate' although they are less friendly. A tiny Forth compiler has successfully been ported to the MC68008 system, using the stand alone software.

BIBLIOGRAPHY

Coffron, James. Using and Troubleshooting the MC68000. Reston, VA: Reston Publishing Company, 1983.

Harman,Thomas L. and Barbara Lawson. The Motorola MC68000 Microprocessor Family: Assembly Language, Interface Design, and System Designs. Englewood Cliffs, NJ: Prentice-Hall, 1985.

Jaulent, Patrick. The 68000 Hardware and Software. Basingstoke, Hampshire (U.K.): Macmillan, 1985.

Kane,G. 68000 Microprocessor Handbook. Berkeley, CA: Osborne/McGraw-Hill, 1981.

Monolithic Memories. Programmable Logic Handbook. Santa Clara, CA: Monolithic Memories, 1985.

Motorola. Data Sheets for the MC68000 and MC68681. Austin,Texas: Motorola, 1985.

Motorola. MC68000 8-/16-/32-Bit Microprocessors,Programmer's Reference Manual. Englewood Cliffs, NJ: Prentice-Hall, 1986.

Rockwell. Data Sheets for the R6522. Newport Beach, CA: Rockwell, 1987.

Schwaderer,W. C Wizard's Programming Reference. New York, NY: Wiley Press, 1985.

Appendix A: PALASM Syntax and PAL Equations

The PALASM syntax is defined by the M.M.I. corporation. The equations used for the development of the two PALs in this system were written using PALASM 2.

PALASM allows the designer to write equations using meaningful signal names. Firstly, PALASM requires certain fields to be filled in, such as the title, the pattern, the revision, the author, the company, the date and finally the chip name. Secondly, the designer defines names for all the pins on the PAL in the heading section, starting from pin 1 and ending with pin 20 or 24 according to the size PAL. Names are assumed to be positive if not specified otherwise. If the user wishes to specify an active low input or output, a '/' must precede the signal name. The word EQUATIONS must appear next, followed by the output pin equations. If the PAL is implementing purely sequential logic, the equations are written in the form:

x=y*z+h*j

where the '*' means logical AND and the '+' means logical OR. If a registered PAL function is used, the '=' sign is preceded by a ':', to indicate the use of a register. The equations are written in a similar manner to the header with a '/' being the NOT operator. The tri-state outputs are defined by their own equation, which consists of the signal name plus the extension '.TRST'. For example, see DTACK on the 20L10.

Comments can be included in the equations by the use of a ';' character.

For more information on PALASM see the M.M.I. Programmable Array Logic handbook.

```
TITLE ADDRESS DECODING PAL
PATTERN ADD.DAT
REVISION 1.2 (C) JULY 87
AUTHOR N.D.WAITES
COMPANY RICE UNIVERSITY
DATE JUNE 13 87


CHIP ADD_DECODER PAL20L10

/AS A19 A18 A17 A16 /DS RW /BG /EXTSEL6800 /IACK_DUART NC GND
/ICS3 /CS3 /CS5 /CS4 /CS0 /CS1 /CS2 SEL6800 /DATA_EN /DTACK  EXR_W VDD

EQUATIONS
CS0=AS*/A19*/A18*/A17*/A16        ; CHIP SELECTS I.E.,  $00000
CS1=AS*/A19*/A18*/A17*A16         ; $10000
CS2=AS*/A19*/A18*A17*/A16         ; $20000
CS3=AS*/A19*/A18*A17*A16          ; $30000
CS4=AS*/A19*A18*/A17*/A16         ; $40000
CS5=AS*/A19*A18*/A17*A16          ; $50000
/SEL6800=/CS4*/CS5*/EXTSEL6800    ; DETERMINES WHEN A 6800 DEVICE IS
                                  ; SELECTED.
                                  ; USED TO GENERATE VPEN WHICH DRIVES VPA
/EXR_W=BG*RW                      ; THIS GENERATES A READ/WRITE DIRECTION
       +/RW*/BG                   ; FOR THE DATA BUFFER (74LS245 - U5)
                                  ; WHEN BG IS FALSE WE WANT NORMAL R/W AS
                                  ; GENERATED BY THE 68008, WHEREAS WHEN
                                  ; BG IS TRUE WE DESIRE R/W TO BE
                                  ; INVERTED THIS IS EQUIVALENTLY
                                  ; (BGN= BG ACTIVE LOW)
                                  ; READ (HIGH) =
                                  ;       (NOT BGN)*R/W + BGN*(NOT R/W)
                                  ; WRITE (LOW) =
                                  ;       BGN*R/W + (NOT BGN)*(NOT R/W)
DATA_EN=A19*AS*/IACK_DUART        ; ENABLE BUFFERS WHEN CPU GOES OFFBOARD
                                  ; 1.2 UPDATE : ADDED IACK_DUART TO STOP
                                  ; U5 DRIVING THE BUS DURING AN IACK
                                  ; CYCLE IN WHICH THE UART PASSES A
                                  ; VECTOR TO THE CPU.
                                  ; NOTE : WHEN EITHER OF THE 6800 IACK
                                  ; CYCLES OCCUR AUTOVECTORING IS USED,
                                  ; THE DATA BUFFER IS ENABLED AND DATA
                                  ; IS PRESENTED TO THE CPU.
                                  ; HOWEVER, THE CPU's DATA BUS IS IN HIGH
                                  ; IMPEDANCE STATE AND THEREFORE IGNORES
                                  ; THE DATA.
           +BG*AS*/A19            ; ALLOW DMA TRANSFERS TO ACCESS
                                  ; MEM,VIAs,UART.
                                  ; NOTE : USER CAN ONLY PLACE DEVICES IN
                                  ; TOP HALF OF THE ADDRESS SPACE
                                  ; $80000-$FFFFF.
        +/A19*/BG*A18*A17*A16*AS  ; ALLOWS OFFBOARD ADDRESSING OF BLOCK
                                  ; $70000-$7FFFF WHICH IS USED FOR
                                  ; ADDRESSING MOTHER BOARD  DEVICES.
```

```
                                    ; THIS FREES THE WHOLE OF THE TOP 512K
                                    ; ADDRESS SPACE FOR USE BY THE USER.

DTACK=DS*CS0+DS*CS1+DS*CS2          ; GENERATES DTACK FOR THE ROM/RAM's.
                                    ; DTACK IS NOT USED TO STRETCH THE BUS
                                    ; CYCLE WITH A 7.3728MHz CLOCK.THE
                                    ; READ/WRITE CYCLE IS AROUND 300
                                    ; NANOSECONDS, HENCE THE CPU RUNS FLAT
                                    ; OUT. 200 NANOSECOND ROM/RAMs WORK OK
                                    ; AS PAL DECODING TAKES AROUND 35nS,
                                    ; 250nS WILL PROBABLY WORK OK TOO.
                                    ; NOTE : DTACK IS GENERATED EXCLUSIVELY
                                    ; FOR ADDRESSES 0-2FFFF HEX. HOWEVER,
                                    ; DTACK IS DRIVEN HIGH THROUGH ADDRESSES
                                    ; $30000-$7FFFF

DTACK.TRST=
    DS*/ICS3*/CS4*/CS5*/A19         ; NOTE : DTACK IS AN OPEN COLLECTOR
                                    ; SIGNAL WHICH IS SIMULATED BY THE PAL
                                    ; BY ENABLING THE OUTPUT WHEN WE WISH TO
                                    ; DRIVE THE LINE ACTIVE (I.E., LOW),
                                    ; ELSE IT REMAINS TRI-STATED.
                                    ; 1.1 UPDATE : ORIGINALLY THE TOP 512K
                                    ; WAS TO HAVE A FULL SPEED DTACK
                                    ; GENERATED BY THE PAL. A DISABLE LINE
                                    ; WOULD HAVE ALLOWED THE USER TO EXTEND
                                    ; THE BUS CYCLE.
                                    ; HOWEVER, DUE TO THE LACK OF CONNECTOR
                                    ; PINS THIS WAS DROPPED,AND THE MORE
                                    ; USEFUL VPA SIGNAL WAS PROVIDED TO THE
                                    ; USER. NOW ALL OFFBOARD DEVICES MUST
                                    ; GENERATE THEIR OWN DTACK SIGNALS.
                                    ; (OPEN COLLECTOR, OF COURSE)
                                    ; ONBOARD FULL SPEED DTACK IS GENERATED
                                    ; FOR ADDRESS RANGES : $00000-$2FFFF
                                    ; (ROM/RAM)
                                    ; N.B. : DURING AN IACK (INTERRUPT
                                    ; ACKNOWLEDGE) CYCLE THE CPU OUTPUTS
                                    ; A4-A19 HIGH, IF A DEVICE IS MAPPED
                                    ; INTO THE TOP OF MEMORY THEN YOU MUST
                                    ; DECODE THE FC0-FC2 SIGNALS TO
                                    ; DETERMINE WHETHER THE CPU IS EXECUTING
                                    ; A NORMAL MEMORY ACCESS OR AN INTERRUPT
                                    ; ACKNOWLEDGE CYCLE.
```

```
PALASM XPLOT, V2.12 I - M.M.I. INTERNAL RELEASE (2-JUL-1985)
(C) - COPYRIGHT MONOLITHIC MEMORIES INC., 1984,1985

Title     : ADDRESS DECODING PAL             Author  : N.D.WAITES

Pattern   : ADD.DAT                          Company : RICE UNIVERSITY

Revision  : 1.2 (C) JULY 87                  Date    : JUNE 13 87


PAL20L10
ADD_DECODER

                   11 1111 1111 2222 2222 2233 3333 3333
     0123 4567 8901 2345 6789 0123 4567 8901 2345 6789

   0 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
   1 ---- ---- ---- ---- ---- X--- -X-- ---- ---- ----
   2 ---- ---- ---- ---- ---- -X-- X--- ---- ---- ----
   3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
   4 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   5 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   6 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   7 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

   8 -X-- ---- ---- ---- -X-- ---- ---- --X- --X- --X-
   9 ---- ---- ---- ---- -X-- ---- ---X ---- ---- ----
  10 ---- ---- ---- ---- -X-- ---X ---- ---- ---- ----
  11 ---- ---- ---- ---- -X-X ---- ---- ---- ---- ----
  12 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  13 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  14 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  15 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

  16 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  17 X--X ---- ---- ---- ---- ---- ---- ---- X--- ----
  18 -X-X ---- ---- ---- ---- ---- -X-- ---- ---- ----
  19 -X-X X--- X--- X--- ---- ---- X--- ---- ---- ----
  20 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  21 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  22 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  23 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

  24 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  25 ---- ---- ---- ---- ---- ---- ---- X-X- --X- ----
  26 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  27 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  28 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  29 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  30 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  31 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

  32 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  33 -X-X -X-- X--- -X-- ---- ---- ---- ---- ---- ----
  34 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

```
35 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
36 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
37 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
38 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
39 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

40 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
41 -X-X -X-- -X-- X--- ---- ---- ---- ---- ---- ----
42 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
43 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
44 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
45 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
46 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
47 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

48 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
49 -X-X -X-- -X-- -X-- ---- ---- ---- ---- ---- ----
50 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
52 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
53 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
54 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
55 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

56 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
57 -X-X X--- -X-- -X-- ---- ---- ---- ---- ---- ----
58 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
60 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
61 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
62 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
63 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

64 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
65 -X-X X--- -X-- X--- ---- ---- ---- ---- ---- ----
66 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
67 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
68 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
69 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
70 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
71 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

72 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
73 -X-X -X-- X--- X--- ---- ---- ---- ---- ---- ----
74 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
75 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
76 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
77 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
78 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
79 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

TOTAL FUSES BLOWN: 940

```
TITLE 68008 INTERRUPT GLUE
PATTERN INT.DAT
REVISION 1.0b (C) JULY 87 INTERRUPT ACK OUTPUTS TRUE LOGIC ON A1-A2 (NOT
INVERTED !)
AUTHOR  N.D.WAITES
COMPANY RICE UNIVERSITY
DATE JUNE 13 87

CHIP INT_GLUE PAL16L8


A2  A1 FC0 FC1 FC2 /AS SEL6800 /IRQ_DUART /IRQ6522 GND
RESET /HALT /IACK6800 /IACK_DUART VPA /SUPERV /IRQSW /IPL1 /IPL0_2 VDD

EQUATIONS
IPLO_2=IRQ_DUART                          ; IPL0_2=DUART INT or
       +IRQSW                             ; NOTE: INT1 ONLY BECOMES
                                          ; ACTIVE WHEN IRQSW IS ACTIVE
                                          ; OR WHEN IRQ6522 IS ACTIVE
                                          ; AND IRQ_DUART IS NEGATED
                                          ; I.E. WE HAVE PRIORITIES

IPL1=IRQ6522*/IRQ_DUART                   ; IPL1=DUART INT or NMI switch
     +IRQSW

IACK_DUART=FC0*FC1*FC2*A1*/A2*AS          ; DUART HAS PRIORITY 5

/VPA=/AS+/SEL6800*/IACK6800               ; VPA = AS*(SEL6800+IACK6800)

IACK6800=FC0*FC1*FC2*/A1*A2               ; NMI PRIO 7, VIA PRIOR 2
         +FC0*FC1*FC2*A1*A2

SUPERV = /FC0*FC1*FC2*AS+FC2*/FC1*FC0*AS  ; DECODE SUPERVISOR MODE

HALT = RESET
                                          ; RESET IS INVERTED

HALT.TRST = RESET                         ; FAKE OPEN COLLECTOR TO ALLOW
                                          ; FOR SOFTWARE RESET DRIVING
                                          ; THE RESET LINE
```

```
PALASM XPLOT, V2.12 I - M.M.I. INTERNAL RELEASE (2-JUL-1985)
(C) - COPYRIGHT MONOLITHIC MEMORIES INC., 1984,1985

Title    : 68008 INTERRUPT GLUE          Author  : N.D.WAITES

Pattern  : INT.DAT                       Company : RICE UNIVERSITY

Revision : 1.0b (C) JULY 87 INTERRU      Date    : JUNE 13 87


PAL16L8
INT_GLUE

               11 1111 1111 2222 2222 2233
     0123 4567 8901 2345 6789 0123 4567 8901

  0 ---- ---- ---- ---- ---- ---- ---- ----
  1 ---- ---- ---- ---- ---- ---- -X-- ----
  2 ---- ---- ---X ---- ---- ---- ---- ----
  3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  4 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  5 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  6 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  7 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

  8 ---- ---- ---- ---- ---- ---- ---- ----
  9 ---- ---- ---- ---- ---- ---- X--- -X--
 10 ---- ---- ---X ---- ---- ---- ---- ----
 11 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 12 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 13 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 14 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 15 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 16 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 17 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 18 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 19 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 20 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 21 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 22 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 23 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 24 ---- ---- ---- ---- ---- ---- ---- ----
 25 ---- -X-- X--- X--- -X-- ---- ---- ----
 26 ---- X--- -X-- X--- -X-- ---- ---- ----
 27 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 28 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 29 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 30 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 32 ---- ---- ---- ---- ---- ---- ---- ----
 33 ---- ---- ---- ---- X--- ---- ---- ----
```

```
34 ---- ---- ---- ---- ---- -X-- --X- ----
35 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
36 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
37 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

40 ---- ---- ---- ---- ---- ---- ---- ----
41 X--X X--- X--- X--- -X-- ---- ---- ----
42 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
43 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
44 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
45 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
46 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
47 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

48 ---- ---- ---- ---- ---- ---- ---- ----
49 -XX- X--- X--- X--- ---- ---- ---- ----
50 X-X- X--- X--- X--- ---- ---- ---- ----
51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

56 ---- ---- ---- ---- ---- ---- ---- --X-
57 ---- ---- ---- ---- ---- ---- ---- --X-
58 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

TOTAL FUSES BLOWN:   574
```

Appendix B: Program Examples

This appendix contains two simple programs which demonstrate various features of the system and two others which can be used as diagnostic aids. The following programs plus all the source to the monitor, the MC68000 assembler and 'Emulate' program are available on CLEO, in sub-directories below /ul/hd/src/426. The four sub-directories are named asm, em, mon and examples.

Program 1.

The first program simply displays four identical digits on the LED displays, counting up every quarter of a second. The display should read '0000', '1111', etc. The LED display driver IC is driven by the VIA in this example. The following connections must be made for this program to work correctly :-

From		To	
BC-25	7212-AD0	BD 9	X-PA4
BC-24	7212-AD1	BD 10	X-PA5
BC-23	7212-CS1	BD 11	X-PA6
BD-25	7212-D0	BD 5	X-PA0
BD-24	7212-D1	BD 6	X-PA1
BD-23	7212-D2	BD 7	X-PA2
BD-22	7212-D3	BD 8	X-PA3

A jumper block marked LED is available from A145 (see Hubert Daugherty). This can be plugged straight into the breadboard connector for diagnostic testing.

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                                 CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------------  ---------

           org     $10000
   xouta:  equ     $40002
                                                                40002
   xdda:   equ     $40006
                                                                40006
   go:     bsr     messg
                                                                10000 6100
                                                                10002 fffe
           move.b #$ff,xdda ; set X-A to output mode
                                                                10004 13fc
                                                                10006 00ff
                                                                10008 0004
                                                                1000a 0006
           clr.b  xouta  ; write zero
                                                                1000c 4239
                                                                1000e 0004
                                                                10010 0002
   sd:     clr.w  D2     ; numeric counter
                                                                10012 4242
           move.w #3,D3        ; address counter
                                                                10014 363c
                                                                10016 0003
   start:  bsr    writled
                                                                10018 6100
                                                                1001a ffe6
           bsr    delay
                                                                1001c 6100
                                                                1001e ffe2
           addi.w #$1111,D2
                                                                10020 0642
                                                                10022 1111
           bcs    sd
                                                                10024 6500
                                                                10026 ffec
           bra    start
                                                                10028 6000
                                                                1002a ffee
   writled: move.wD2,D5
                                                                1002c 3a02
   ledl:
           move.w D5,D4
                                                                1002e 3805
           andi.w #$f,D4
                                                                10030 0244
                                                                10032 000f
           move.b D3,D1
                                                                10034 1203
           asl.b  #4,D1
```

```
                                                            10036 e901
               or.b   D4,D1
                                                            10038 8204
               andi   #$bf,D1
                                                            1003a 0241
                                                            1003c 00bf
               move.b D1,xouta
                                                            1003e 13c1

Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                              CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------------  ---------

                                                            10040 0004
                                                            10042 0002
               ori.b  #$40,D1
                                                            10044 0001
                                                            10046 0040
               move.b D1,xouta
                                                            10048 13c1
                                                            1004a 0004
                                                            1004c 0002
               asr.w  #4,D5
                                                            1004e e845
               dbf    D3,led1
                                                            10050 51cb
                                                            10052 ffdc
               move.w #3,D3
                                                            10054 363c
                                                            10056 0003
               rts
                                                            10058 4e75

    delay:     move   #$2fff,D7
                                                            1005a 3e3c
                                                            1005c 2fff
    delay1:    ori.b  #$80,xouta
                                                            1005e 0039
                                                            10060 0080
                                                            10062 0004
                                                            10064 0002
               andi.b #$7f,xouta
                                                            10066 0239
                                                            10068 007f
                                                            1006a 0004
                                                            1006c 0002
               dbf    D7,delay1
                                                            1006e 51cf
                                                            10070 ffee
               rts
                                                            10072 4e75
    messg:     lea    mess,A0
                                                            10074 41f9
```

```
                                                              10076 0000
                                                              10078 0000
    36:          writs
                                                              1007a 4e48
    37:          rts
                                                              1007c 4e75
    38: mess:    ds " LED diagnostic "
                                                              1007e 20
                                                              1007f 4c
                                                              10080 45
                                                              10081 44
                                                              10082 20
                                                              10083 64
                                                              10084 69
                                                              10085 61
                                                              10086 67
                                                              10087 6e
```



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

                                                              10088 6f
                                                              10089 73
                                                              1008a 74
                                                              1008b 69
                                                              1008c 63
                                                              1008d 20
    39:          ds "check\n\r\0"
                                                              1008e 63
                                                              1008f 68
                                                              10090 65
                                                              10091 63
                                                              10092 6b
                                                              10093 0a
                                                              10094 0d
                                                              10095 00
    40:          end    go
```

Program 2.

This program demonstrates how the assembler can be used to write relocatable code. The memory test program contains no absolute addressing. It can be loaded anywhere in memory and then executed. The program also uses the TRAP instructions to fetch numbers from the user. When the program is executed it first requests the starting addressing of the memory test. It then prompts for the number of bytes to be tested. The program runs and reports the status of the memory check.

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                              CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
-------------------------------------------------------------  ---------

    1:          org $0
    2: amess:   ds "Input start address ?\0"
                                                             0000 49
                                                             0001 6e
                                                             0002 70
                                                             0003 75
                                                             0004 74
                                                             0005 20
                                                             0006 73
                                                             0007 74
                                                             0008 61
                                                             0009 72
                                                             000a 74
                                                             000b 20
                                                             000c 61
                                                             000d 64
                                                             000e 64
                                                             000f 72
                                                             0010 65
                                                             0011 73
                                                             0012 73
                                                             0013 20
                                                             0014 3f
                                                             0015 00
    3: mess:    ds "Memory error at \0"
                                                             0016 4d
                                                             0017 65
                                                             0018 6d
                                                             0019 6f
                                                             001a 72
```

```
                                                        001b 79
                                                        001c 20
                                                        001d 65
                                                        001e 72
                                                        001f 72
                                                        0020 6f
                                                        0021 72
                                                        0022 20
                                                        0023 61
                                                        0024 74
                                                        0025 20
                                                        0026 00
    4: mlen:    ds "Input length of test ?\0"
                                                        0028 49
                                                        0029 6e
                                                        002a 70
                                                        002b 75
                                                        002c 74
                                                        002d 20
                                                        002e 6c
                                                        002f 65
                                                        0030 6e
                                                        0031 67
                                                        0032 74
                                                        0033 68
```

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                           CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
-------------------------------------------------------   ---------
```

```
                                                        0034 20
                                                        0035 6f
                                                        0036 66
                                                        0037 20
                                                        0038 74
                                                        0039 65
                                                        003a 73
                                                        003b 74
                                                        003c 20
                                                        003d 3f
                                                        003e 00
    5: welc:    ds " Memory test V1.0 (NDW) "
                                                        0040 20
                                                        0041 4d
                                                        0042 65
                                                        0043 6d
                                                        0044 6f
                                                        0045 72
                                                        0046 79
                                                        0047 20
                                                        0048 74
                                                        0049 65
                                                        004a 73
```

```
                                                        004b 74
                                                        004c 20
                                                        004d 56
                                                        004e 31
                                                        004f 2e
                                                        0050 30
                                                        0051 20
                                                        0052 28
                                                        0053 4e
                                                        0054 44
                                                        0055 57
                                                        0056 29
                                                        0057 20
    6:              ds " Dec 1 87 Rice Univ."
                                                        0058 20
                                                        0059 44
                                                        005a 65
                                                        005b 63
                                                        005c 20
                                                        005d 31
                                                        005e 20
                                                        005f 38
                                                        0060 37
                                                        0061 20
                                                        0062 52
                                                        0063 69
                                                        0064 63
                                                        0065 65
                                                        0066 20
                                                        0067 55
                                                        0068 6e
                                                        0069 69
```

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                             CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
-------------------------------------------------------------  ---------
```

```
                                                        006a 76
                                                        006b 2e
    7:              ds " (Completely relocatable) "
                                                        006c 20
                                                        006d 28
                                                        006e 43
                                                        006f 6f
                                                        0070 6d
                                                        0071 70
                                                        0072 6c
                                                        0073 65
                                                        0074 74
                                                        0075 65
                                                        0076 6c
                                                        0077 79
                                                        0078 20
```

```
                                                        0079 72
                                                        007a 65
                                                        007b 6c
                                                        007c 6f
                                                        007d 63
                                                        007e 61
                                                        007f 74
                                                        0080 61
                                                        0081 62
                                                        0082 6c
                                                        0083 65
                                                        0084 29
                                                        0085 20
  8:              ds "\r\n\n\0"
                                                        0086 0d
                                                        0087 0a
                                                        0088 0a
                                                        0089 00
  9: good:    ds " No errors found \r\n\0"
                                                        008a 20
                                                        008b 4e
                                                        008c 6f
                                                        008d 20
                                                        008e 65
                                                        008f 72
                                                        0090 72
                                                        0091 6f
                                                        0092 72
                                                        0093 73
                                                        0094 20
                                                        0095 66
                                                        0096 6f
                                                        0097 75
                                                        0098 6e
                                                        0099 64
                                                        009a 20
                                                        009b 0d
                                                        009c 0a
                                                        009d 00

Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                          CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------------  ---------

 10: go1:
 11:          clr.b  D5
                                                        009e 4205
 12:          lea    welc(PC),A0
                                                        00a0 41fa
                                                        00a2 ff9e
 13:          writs
                                                        00a4 4e48
 14:      bsr getadd
```

```
                                                              00a6 6100
                                                              00a8 ff58
15:          bsr    go
                                                              00aa 6100
                                                              00ac ff54
16:          bsr    start
                                                              00ae 6100
                                                              00b0 ff50
17:          bsr    go
                                                              00b2 6100
                                                              00b4 ff4c
18:          bsr    check
                                                              00b6 6100
                                                              00b8 ff48
19:          tst.b  D5
                                                              00ba 4a05
20:          bne    lexit
                                                              00bc 6600
                                                              00be ff42
21:          lea    good(PC),A0
                                                              00c0 41fa
                                                              00c2 ffc8
22:          writs
                                                              00c4 4e48
23: lexit:   exit
                                                              00c6 4e40
24: check:
25:          bsr    getn
                                                              00c8 6100
                                                              00ca ff36
26:          cmp.b  (A4)+,D0
                                                              00cc b01c
27:          beq    tnext
                                                              00ce 6700
                                                              00d0 ff30
28:          moveq  #1,D5
                                                              00d2 7a01
29:          lea mess(PC),A0
                                                              00d4 41fa
                                                              00d6 ff40
30:          writs
                                                              00d8 4e48
31:          move.l A4,D0
                                                              00da 200c
32:          subq.l #1,D0
                                                              00dc 5380
33:          writl

Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                             CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
-------------------------------------------------------------  ---------

                                                              00de 4e46
```

```
          crlf
                                                      00e0 4e4a
tnext:    dbf     D7,check
                                                      00e2 51cf
                                                      00e4 ffe4
          rts
                                                      00e6 4e75
getadd:   lea     amess(PC),A0
                                                      00e8 41fa
                                                      00ea ff16
          writs
                                                      00ec 4e48
          getl
                                                      00ee 4e43
          movea.l    D0,A6
                                                      00f0 2c40
          crlf
                                                      00f2 4e4a
          lea     mlen(PC),A0
                                                      00f4 41fa
                                                      00f6 ff32
          writs
                                                      00f8 4e48
          getw
                                                      00fa 4e42
          subq    #1,D0
                                                      00fc 5340
          move.w D0,D6
                                                      00fe 3c00
          crlf
                                                      0100 4e4a
          crlf
                                                      0102 4e4a
          rts
                                                      0104 4e75


go: move.w         D6,D7
                                                      0106 3e06
          clr.b  D1
                                                      0108 4201
          clr.b  D2
                                                      010a 4202
          clr.b  D3
                                                      010c 4203
          movea.l    A6,A4
                                                      010e 284e
          rts
                                                      0110 4e75
start:
          bsr     getn
                                                      0112 6100
                                                      0114 feec
          move.b D0,(A4)+
```

```
                                                        0116 18c0
   61:          dbf     D7,start
                                                        0118 51cf
                                                        011a fff8
   62:          rts
                                                        011c 4e75
   63: getn:
   64:          clr.b  D0
                                                        011e 4200
   65:
   66:
   67:
   68:          add.b  D1,D0
                                                        0120 d001
   69:          add.b   D2,D1
                                                        0122 d202
   70:          addq   #1,D3
                                                        0124 5243
   71:          cmpi.b #7,D3
                                                        0126 0c03
                                                        0128 0007
   72:          bne        next
                                                        012a 6600
                                                        012c fed4
   73:          clr.b  D3
                                                        012e 4203
   74:          addq.b #1,D2
                                                        0130 5202
   75: next:    rts
                                                        0132 4e75
   76:          end go1
```

Program 3.

Program 3 is a simple diagnostic program that checks that the 74HC4066s are functioning correctly. The program reads and writes to the parallel ports verifying the data. This program also requires jumpers across the breadboard connector. All the data bits on the X port should be looped to the data bits on the Y port. This means connections should be made from :

From		To	
BC-5	Y-PA0	BD-5	X-PA0
BC-6	Y-PA1	BD-6	X-PA1
down to			
BC-12	Y-PA7	BD-12	X-PA7

and similarly for the B ports; *i.e.*, Y-PB0 to X-PB0 upto Y-PB7 to X-PB7. A jumper block which has this connections is available from A145 (see Hubert).

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                                CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
---------------------------------------------------------------  ---------

     1:          org URAM
     2: xdda:    equ     $40006
                                                               40006
     3: ydda:    equ     $50006
                                                               50006
     4: youta:   equ     $50002
                                                               50002
     5: xouta:   equ     $40002
                                                               40002
     6: yinb:    equ     $50000
                                                               50000
     7: xinb:    equ     $40000
                                                               40000
     8: yddb:    equ     $50004
                                                               50004
     9: xddb:    equ     $40004
                                                               40004
```

```
start:  lea     tmess,A0
                                                            10000 41f9
                                                            10002 0000
                                                            10004 0000
        writs
                                                            10006 4e48
        getch
                                                            10008 4e47
        clr.b   noerr
                                                            1000a 4239
                                                            1000c 0000
                                                            1000e 0000
        move.b #$ff,ydda
                                                            10010 13fc
                                                            10012 00ff
                                                            10014 0005
                                                            10016 0006
        move.b #$55,youta
                                                            10018 13fc
                                                            1001a 0055
                                                            1001c 0005
                                                            1001e 0002
        cmpi.b #$55,xouta
                                                            10020 0c39
                                                            10022 0055
                                                            10024 0004
                                                            10026 0002
        beq     t1ok
                                                            10028 6700
                                                            1002a ffd6
        addq.b #1,noerr
                                                            1002c 5239
                                                            1002e 0000
                                                            10030 0000
t1ok:   move.b #$aa,youta
                                                            10032 13fc
                                                            10034 00aa
                                                            10036 0005
                                                            10038 0002
        cmpi.b #$aa,xouta
                                                            1003a 0c39
                                                            1003c 00aa
                                                            1003e 0004
                                                            10040 0002
        beq     t2ok

Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                             CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------   ---------

                                                            10042 6700
                                                            10044 ffbc
```

```
23:             addq.b #1,noerr
                                                    10046 5239
                                                    10048 0000
                                                    1004a 0000
24:             clr.b  ydda
                                                    1004c 4239
                                                    1004e 0005
                                                    10050 0006
25: t2ok:       move.b #$ff,yddb
                                                    10052 13fc
                                                    10054 00ff
                                                    10056 0005
                                                    10058 0004
26:             move.b #$55,yinb
                                                    1005a 13fc
                                                    1005c 0055
                                                    1005e 0005
                                                    10060 0000
27:             cmpi.b #$55,xinb
                                                    10062 0c39
                                                    10064 0055
                                                    10066 0004
                                                    10068 0000
28:             beq    t3ok
                                                    1006a 6700
                                                    1006c ff94
29:             addq.b #1,noerr
                                                    1006e 5239
                                                    10070 0000
                                                    10072 0000
30: t3ok:       move.b #$aa,yinb
                                                    10074 13fc
                                                    10076 00aa
                                                    10078 0005
                                                    1007a 0000
31:             cmpi.b #$aa,xinb
                                                    1007c 0c39
                                                    1007e 00aa
                                                    10080 0004
                                                    10082 0000
32:             beq    t4ok
                                                    10084 6700
                                                    10086 ff7a
33:             addq.b #1,noerr
                                                    10088 5239
                                                    1008a 0000
                                                    1008c 0000
34: t4ok:       tst.b  noerr
                                                    1008e 4a39
                                                    10090 0000
                                                    10092 0000
35:             beq    testok
                                                    10094 6700
```

```
                                                          CODE
-----------------------------------------------------   ---------

                                                        10096 ff6a
    36:          lea    err,A0
                                                        10098 41f9
                                                        1009a 0000
                                                        1009c 0000
    37:          writs
                                                        1009e 4e48
    38:          bra    fin
                                                        100a0 6000
                                                        100a2 ff5e
    39: testok:  lea    good,A0
                                                        100a4 41f9
                                                        100a6 0000
                                                        100a8 0000
    40:          writs
                                                        100aa 4e48
    41: fin:     clr.b  yddb
                                                        100ac 4239
                                                        100ae 0005
                                                        100b0 0004
    42:          exit
                                                        100b2 4e40
    43:
    44: tmess:   ds "Insert tester with"
                                                        100b4 49
                                                        100b5 6e
                                                        100b6 73
                                                        100b7 65
                                                        100b8 72
                                                        100b9 74
                                                        100ba 20
                                                        100bb 74
                                                        100bc 65
                                                        100bd 73
                                                        100be 74
                                                        100bf 65
                                                        100c0 72
                                                        100c1 20
                                                        100c2 77
                                                        100c3 69
                                                        100c4 74
                                                        100c5 68
    45:          ds " 'PORT' at the top"
                                                        100c6 20
                                                        100c7 27
                                                        100c8 50
                                                        100c9 4f
                                                        100ca 52
                                                        100cb 54
                                                        100cc 27
```

```
                                                              100cd 20
                                                              100ce 61
                                                              100cf 74
                                                              100d0 20
                                                              100d1 74
                                                              100d2 68
```



```
                                                                CODE
-------------------------------------------------------------  ---------

                                                              100d3 65
                                                              100d4 20
                                                              100d5 74
                                                              100d6 6f
                                                              100d7 70
    46:             ds " between BC and BD"
                                                              100d8 20
                                                              100d9 62
                                                              100da 65
                                                              100db 74
                                                              100dc 77
                                                              100dd 65
                                                              100de 65
                                                              100df 6e
                                                              100e0 20
                                                              100e1 42
                                                              100e2 43
                                                              100e3 20
                                                              100e4 61
                                                              100e5 6e
                                                              100e6 64
                                                              100e7 20
                                                              100e8 42
                                                              100e9 44
    47:             ds "\r\n\nThen hit any "
                                                              100ea 0d
                                                              100eb 0a
                                                              100ec 0a
                                                              100ed 54
                                                              100ee 68
                                                              100ef 65
                                                              100f0 6e
                                                              100f1 20
                                                              100f2 68
                                                              100f3 69
                                                              100f4 74
                                                              100f5 20
                                                              100f6 61
                                                              100f7 6e
                                                              100f8 79
                                                              100f9 20
    48:             ds "key to test\r\n\0"
```

```
                                                              100fa 6b
                                                              100fb 65
                                                              100fc 79
                                                              100fd 20
                                                              100fe 74
                                                              100ff 6f
                                                              10100 20
                                                              10101 74
                                                              10102 65
                                                              10103 73
                                                              10104 74
                                                              10105 0d
                                                              10106 0a
```

```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                               CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------------  ---------

                                                              10107 00
    49: err:     ds "\r\nConnector bad !!!\r\n\0"
                                                              10108 0d
                                                              10109 0a
                                                              1010a 43
                                                              1010b 6f
                                                              1010c 6e
                                                              1010d 6e
                                                              1010e 65
                                                              1010f 63
                                                              10110 74
                                                              10111 6f
                                                              10112 72
                                                              10113 20
                                                              10114 62
                                                              10115 61
                                                              10116 64
                                                              10117 20
                                                              10118 21
                                                              10119 21
                                                              1011a 21
                                                              1011b 0d
                                                              1011c 0a
                                                              1011d 00
    50: good:    ds "\r\nConnector OK\r\n\0"
                                                              1011e 0d
                                                              1011f 0a
                                                              10120 43
```

```
                                                        10121 6f
                                                        10122 6e
                                                        10123 6e
                                                        10124 65
                                                        10125 63
                                                        10126 74
                                                        10127 6f
                                                        10128 72
                                                        10129 20
                                                        1012a 4f
                                                        1012b 4b
                                                        1012c 0d
                                                        1012d 0a
                                                        1012e 00
51: noerr:   db 1,0
                                                        10130 00
52:          end start
```

Program 4.

Program 4 demonstrates how to use the auxiliary serial port with the ROM support routines. The ROM routines allow interrupt driven buffering of data from the second parallel port. Note: the default software sets the second serial port at 9600 baud, even parity, and one stop bit. This program requires a terminal to be connected to the second serial port, enabling the user to interact with the program.

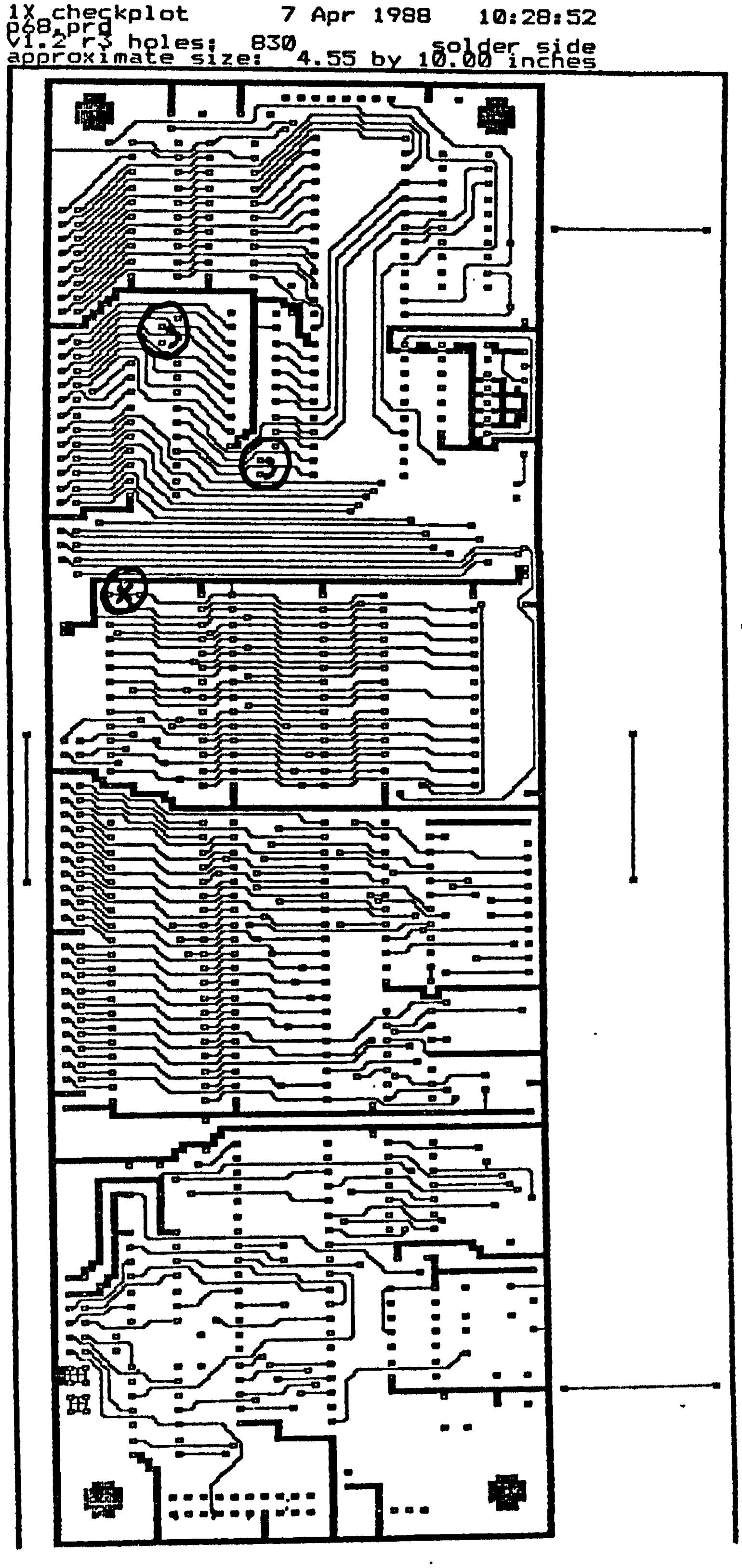
```
Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                                   CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
-------------------------------------------------------------   ---------

            org    $10000
   start:   lea    USTK,A7
                                                                10000 4ff9
                                                                10002 0001
                                                                10004 113c
            super
                                                                10006 4e4b
            move.b #$22,$3000a  ; allow both ports to interrupt
                                                                10008 13fc
                                                                1000a 0022
                                                                1000c 0003
                                                                1000e 000a
   fred:    lea    mess,A0
                                                                10010 41f9
                                                                10012 0000
                                                                10014 0000
            bsr    writm
                                                                10016 6100
                                                                10018 ffe8
   fred1:
            jsr    $5a4   ; get char from aux port
                                                                1001a 4eb8
                                                                1001c 05a4
            move.b D0,D1
                                                                1001e 1200
            writc      ; echo on users terminal
                                                                10020 4e49
            jsr    $36a   ; echo on aux terminal
                                                                10022 4eb8
                                                                10024 036a
            cmpi.b #$d,D1
                                                                10026 0c01
                                                                10028 000d
            bne    fred1
```

```
                                                            1002a 6600
                                                            1002c ffee
    14:          move.b #2,$3000a ; now only allow terminal port to
interrupt
                                                            1002e 13fc
                                                            10030 0002
                                                            10032 0003
                                                            10034 000a
    15:          andi   #$dfff,SR
                                                            10036 027c
                                                            10038 dfff
    16:          exit
                                                            1003a 4e40
    17: writm:   move.b (A0)+,D0
                                                            1003c 1018
    18:          beq    done
                                                            1003e 6700
                                                            10040 ffc0
    19:          jsr    $36a
                                                            10042 4eb8
                                                            10044 036a
    20:          bra    writm

Motorola 68000 Cross Assembler Version 5a Feb 88 Rice U.
Cross assembles 68000 on Unix,IBM PC,Amiga & Macintosh
                                                              CODE
Assembler V5.0  (C) 1988 Rice & Vanderbilt  EE & CS Department
--------------------------------------------------------------  ---------

                                                            10046 6000
                                                            10048 fff4
    21: done:    rts
                                                            1004a 4e75
    22: mess:    ds "\n\rEnter line <en"
                                                            1004c 0a
                                                            1004d 0d
                                                            1004e 45
                                                            1004f 6e
                                                            10050 74
                                                            10051 65
                                                            10052 72
                                                            10053 20
                                                            10054 6c
                                                            10055 69
                                                            10056 6e
                                                            10057 65
                                                            10058 20
                                                            10059 3c
                                                            1005a 65
                                                            1005b 6e
    23:          ds "d with CR>\0"
                                                            1005c 64
                                                            1005d 20
                                                            1005e 77
                                                            1005f 69
```

```
                                                    10060 74
                                                    10061 68
                                                    10062 20
                                                    10063 43
                                                    10064 52
                                                    10065 3e
                                                    10066 00
24:         end start
```

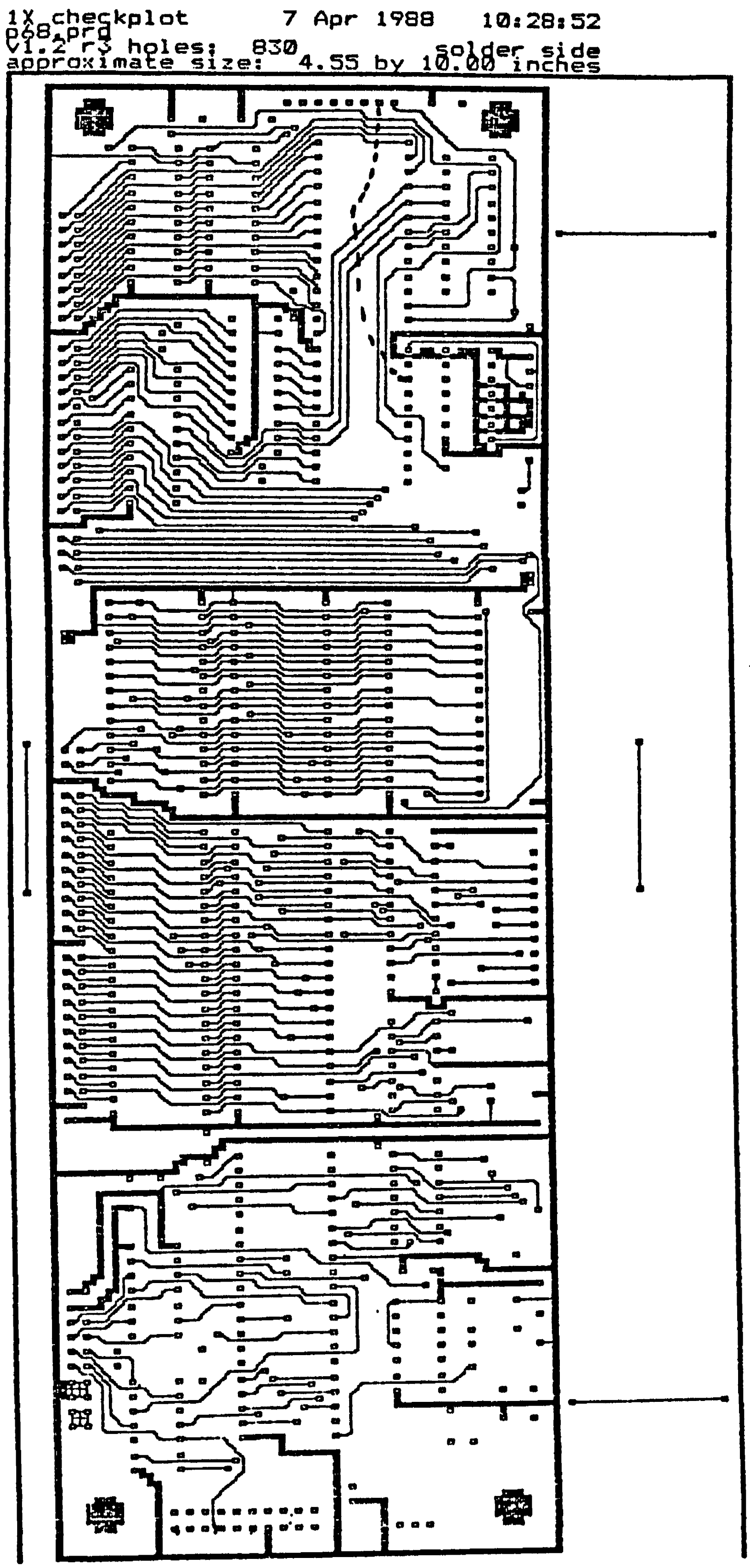
Memory Modifications

where x----x indicates a break in the trace,
and ---- represents a jumper connecting two points.



DMA Modifications

where x----x indicates a break in the trace,
and ---- represents a jumper connecting two points.



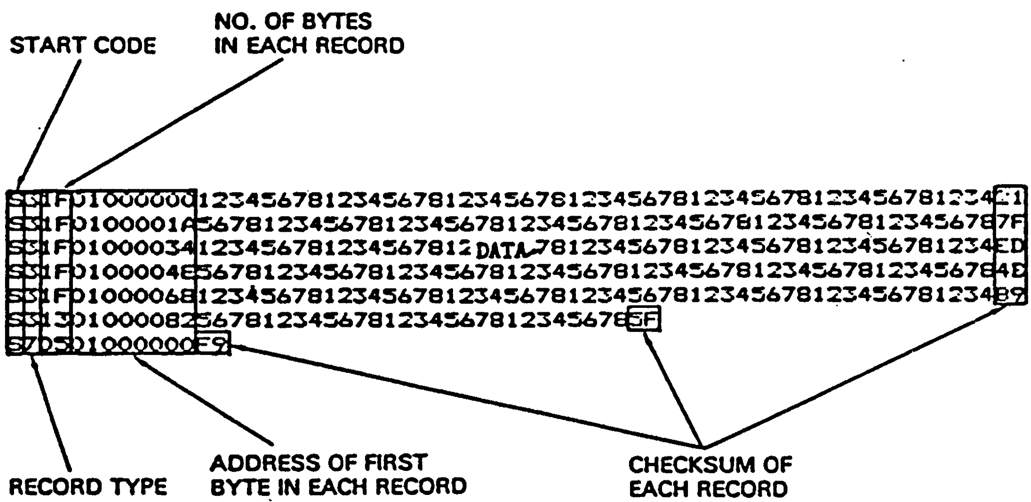
APPENDIX D: MOTOROLA S-RECORD FORMAT

The Motorola S-record format information is reproduced courtesy of Stag Microsystems.

3-Data Record (Eight Character Address)
7-End Record (Eight Character Address) } 4 BYTES

For example:

START ADDRESS:	0000
STOP ADDRESS:	008F
OFFSET:	01000000



3-Data Record (Eight Character Address)
7-End Record (Eight Character Address) } 4 BYTES

MOTOROLA S-RECORD

The MOTOROLA S-RECORD is identical to its standard version when displayed, up to the point that the data's address goes beyond FFFF and thus requires a 5th digit, e.g.: 10000. To compensate for this addition an extra byte is added to the address giving 010000.

When this occurs the record type changes:

The data record changes from 1 to 2
and the end record changes from 9 to 8.

Similarly when the data address goes beyond FFFFFF a 7th digit is required and likewise a byte is added giving the address 8 characters: 01000000.

When this occurs:

The data record changes from 2 to 3
and the end record changes from 8 to 7.

The MOTOROLA S-RECORD consists of:

(i) a start code, i.e.: S

(ii) the record types, i.e.: 1—Data Record (Four Character address)
9—End Record (Four Character address)

2—Data Record (Six character address)
8—End Record (Six character address)

3—Data record (Eight character address)
7—End Record (Eight character address)

(iii) The sum of the number of bytes in an individual record, e.g.: 1D

(iv) the address of the first byte of data in an individual record, e.g.: 0000, 010000, 01000000

(v) data in bytes, e.g.: 12 34 56 78

(vi) checksum of an individual record: 24

CALCULATION OF THE MOTOROLA S-RECORD CHECKSUM

```
S11D00001234567812345678123456781234567812345678123456781234 24
S104001A568B
S9030000FC
```

EXAMPLE: THE SECOND "DATA RECORD" OF THE ABOVE FORMAT.

(i)	this is:	S1 04 00 1A 56 8B
(ii)	the start code, the record type and the checksum are removed:	S1 8B
(iii)	four bytes remain:	04 00 1A 56
(iv)	these are added together:	04 + 00 + 1A + 56 = 74
(v)	the total '74' is converted into Binary:	7 4 0111 0100
(vi)	the Binary figure is reversed. This is known as a complement*:	8 B 1000 1011
(vii)	8B corresponds to the checksum as above:	S1 04 00 1A 56 (8B)

When addition of information occurs in longer records the checksum may consist of more than one byte. When this occurs the least significant byte is always selected to undergo the above calculation.

*When no additional figures are added to this calculation it is called a one's complement.

Appendix E

PA Connector (Microprocessor Board)

Pin	Description	Pin	Description
1	A0	26	BG
2	A1	27	D7
3	A2	28	EXSEL6800
4	A3	29	D6
5	A4	30	BERR
6	A5	31	D5
7	A6	32	AS
8	A7	33	D4
9	A8	34	DS
10	A9	35	D3
11	A10	36	RW
12	A11	37	D2
13	A12	38	DTACK
14	A13	39	D1
15	A14	40	BGACK
16	A15	41	D0
17	GND	42	BR
18	GND	43	GND
19	A16	44	GND
20	A17	45	VMA
21	A18	46	RES
22	A19	47	IRQ6522
23	FC0	48	E
24	FC1	49	VPEN
25	FC2	50	CLK

PB Connector (Microprocessor Board)

Pin	Description	Pin	Description
1	OP3	26	Y-PA7
2	OP2	27	GND
3	IP5	28	GND
4	IP2	29	X-PB0
5	GND	30	Y-PB0
6	GND	31	X-PB1
7	X-CA1	32	Y-PB1
8	Y-CA1	33	X-PB2
9	X-CA2	34	Y-PB2
10	Y-CA2	35	X-PB3
11	X-PA0	36	Y-PB3
12	Y-PA0	37	X-PB4
13	X-PA1	38	Y-PB4
14	Y-PA1	39	X-PB5
15	X-PA2	40	Y-PB5
16	Y-PA2	41	X-PB6
17	X-PA3	42	Y-PB6
18	Y-PA3	43	X-PB7
19	X-PA4	44	Y-PB7
20	Y-PA4	45	X-CB1
21	X-PA5	46	Y-CB1
22	Y-PA5	47	X-CB2
23	X-PA6	48	Y-CB2
24	Y-PA6	49	GND
25	X-PA7	50	GND

PC Connector (Microprocessor board)

Pin	Description	Pin	Description
1	+12	11	CTSA
2	+12	12	RXB
3	-12	13	GND
4	-12	14	GND
5	RTSB	15	GND
6	RTSA	16	GND
7	TXA	17	+5
8	TXB	18	+5
9	CTSB	19	+5
10	RXA	20	+5

Connectors (Buffer Board)

BA		BB	
Pin No.	Description	Pin No.	Description
1	RESET*	1	A0
2	R/W	2	A1
3	E	3	A2
4	AS*	4	A3
5	OP2	5	A4
6	CLK	6	A5
7	DS*	7	A6
8	VMA	8	A7
9	EXSEL6800*	9	A8
10	GND	10	A9
11	GND	11	A10
12	D0	12	A11
13	D1	13	A12
14	D2	14	A13
15	D3	15	A14
16	D4	16	A15
17	D5	17	A16
18	D6	18	A17

BA

Pin No.	Description
19	D7
20	BR*
21	BG*
22	IRQ*
23	FC0
24	FC1
25	FC2

BB

Pin No.	Description
19	A18
20	A19
21	GND
22	GND
23	DTACKU*
24	IP2
25	OP3

* REPRESENTS A 'LOW' ASSERTED SIGNAL

Appendix F: 'Emulate' Message System

'Emulate' communication protocol.

The nine basic primitives which are used to communicate with the MC68008 are listed below. These primitives are implemented in the module called 'monitor.c'. The modular structure of 'Emulate' allows other systems to be interfaced to the program by the adaption of the basic communication primitives.

Getmem

The getmem primitive requests data from the CPU to be sent to the 'Emulate' program. The 'Emulator' program send an address followed by a byte count. The syntax is:

oXXXXXXXX YY<cr>

where XXXXXXXX is a 32-bit address sent as an 8-digit ASCII hexadecimal number. The byte count is a single byte hex number sent as a two-digit ASCII number. *e.g.*,

o00010002 0A<cr> will request the next ten bytes of memory starting from address $10002 to be sent to 'Emulate'. The bytes are also received as ASCII hexadecimal digits. If the MC68008 fails to access the desired memory location, due to a bus error, the monitor sends a control-H to the 'Emulate' program with an ASCII text message following that describes the error.

Putmem

The putmem primitive sends data from the 'Emulate' program to the MC68008's memory. The syntax is as follows:

iXXXXXXXX<cr>[YY][YY]...<cr>.

The starting address is sent similarly to the getmem primitive. Each byte of data to be sent is formatted into a two-digit ASCII hexadecimal number. The number of data bytes sent is variable. The end of the command is indicated by sending a <cr>. If the MC68008 was unable to write into memory, a control-H is sent to 'Emulate', followed by the error message.

Getstat

The getstat primitive requests the MC68008 to send the present contents of its registers to the 'Emulate' program. The command is issued by sending a 'w'. The MC68008 then responds by sending the registers as ASCII hexadecimal digits. (the exact order in which registers are sent can be found in 'sbc.h').

Setstat

The getstat primitive requests the MC68008 to load its registers from the data sent from the 'Emulate' program. 'Emulate' sends a 'q' followed by 140 ASCII digits which represent the values to be placed in the CPU's registers. A <cr> is sent to indicate the end of the command.

Setbreak

The setbreak primitive allows breakpoints to be added into memory. The setbreak command simply sends an address to the monitor program. The syntax is:

b+XXXXXXXX

The breakpoint is inserted at the ASCII address to which XXXXXXXX refers. The monitor programs responds by sending a message back indicating that the breakpoint was added. The message has the form:

Breakpoint added at XXXXXXXX<cr>

If the breakpoint cannot be inserted due to an address error, the monitor sends the same error code as mentioned above (*i.e.*, control-H, followed by an ASCII message).

Dorun

The Dorun primitive executes a program from a given address. The syntax is:

g XXXXXXXX

Once program execution begins, the 'Emulator' waits for one of two messages to be returned. If the program runs and then terminates correctly, a control-B is sent to 'Emulate'. If however, the program causes an exception to be generated, the normal error code sequence is sent.

Dostep

The dostep primitive instructs the MC68008 to execute one instruction from the present value in the Program Counter. The syntax for this command is:

s+g<cr>s-

After this message is sent, the MC68008 executes the instruction. If the MC68008 returns a message of the form 'Program Counter=XXXXXXXX, then the step command was successful. If the step command fails the MC68008 sends an error message.

Reset Sbc

The Reset_Sbc primitive attempts to Reset the MC68008. First a control-A is sent to the MC68008. This message instructs the monitor program to stop sending data over the serial line. This is necessary as user programs can interact via the Sun console, and 'Emulate' needs to distinguish the messages sent by the monitor. Once data is no longer being sent by the monitor, 'Emulate' discards all input data that has been buffered by UNIX. 'Emulate' then sends a control-C to the MC68008. If the control-C was received correctly (*i.e.*, the DUART has not been 'trashed'), the monitor responds by sending a 'Stopped<cr>' message to 'Emulate'. If the control-C character was not received properly, 'Emulate' displays a message on the console asking for the NMI button to be activated. When the NMI button is released the monitor sends the 'Stopped<cr>' message to 'Emulate'. Once 'Emulate' receives this message, it returns to the main menu.

Rundiag

The rundiag primitive is used to check the parallel ports on the MC68008 system. The letter 'x' commands the MC68008 board to execute the diagnostic software. If the system passes all the diagnostic routines, the MC68008 sends a control-G to the 'Emulate' program. If any errors occur, the MC68008 board sends a control-H followed by an ASCII text message describing the error. The error message is terminated by a control-G.