REPORT OF WISC CPU/32 CHIP IMPLEMENTATION EFFORT

SEPTEMBER 2, 1987

Phil Koopman Jr.
WISC Technologies
5551 Beacon St.
Pittsburgh, PA 15217

EXECUTIVE SUMMARY

This is a report of the summer 1987 effort to transfer a discrete board implementation of the WISC CPU/32 processor to a semi-custom CMOS chip implementation. This effort produced a two-chip set which simulates correctly at a 10 MHz clock rate to produce a 5 MOPS 32-bit stack-oriented processor. It is quite likely that an optimized second-pass version of the chip will operate in the 15 MHz to 20 MHz range.

The most critical problem encountered in the implementation effort was the unavailability of asynchronous RAM for use in the stacks. This caused up to a 25 ns increase in the critical paths of the chip set implementation.

This report, when combined with the previously delivered WISC CPU/32 Preliminary Documentation manual, provides written documentation for the chip set: hardware implementation and theory of operation, critical path analysis, performance evaluation, software operation, firmware operation, system testing, prototype test board schematics, and recommendations for future enhancements.

CONTENTS

1. INTRODUCTION

1.1 Purpose

The Harris/WISC CPU/32 project is aimed at ultimately reducing the multi-board WISC/CPU32 discrete component implementation to a chip or chip set. This document provides amplifying information for the WISC CPU/32 implementation delivered to Harris Semiconductor during July and August 1987.

1.2 Scope

This document concerns itself with the design and implementation of the Harris/WISC CPU/32 chip. It provides amplifying information and changes to the WISC CPU/32 Preliminary Documentation (which describes the discrete component CPU/32 implementation) to build a complete picture of the CPU/32 chip design.

Areas covered include: hardware implementation, theory of hardware operation, critical path analysis, performance evaluation, software operation, firmware operation, test vectors, prototype test boards, recommendations for future enhancements, and problems encountered in the design process.

This document is written for an engineer who has already read the WISC CPU/32 Preliminary Documentation and who already has an orientation to the purposes and methods of the Harris/WISC chip project.

1.3 Previously Delivered Information

Information previously delivered to Harris Semiconductor includes: the WISC CPU/32 Preliminary Documentation (2 copies) with software and firmware, Collected WISC Papers, schematics for the chip, and test vectors with associated software. Also, courtesy copies of LIB-FORTH with associated documentation have been previously delivered.

2. HARDWARE IMPLEMENTATION

Converting the design of the WISC CPU/32 discrete component board set ("the Boards") to a chip set ("the Chips") involves partitioning the Board functions into areas that should be on the Chips, partitioning the logic between the Chips to satisfy chip area and pinout constraints, making changes to the design to accommodate partitioning and fabrication technology requirements, then generating test vectors to ensure correct functionality. This section discusses all these steps except test vectors, which are discussed in Section 6.

2.1 Portion of Design Included On-chip

Due to the severe pinout problems associated with two off-chip 32-bit stacks and an off-chip 30-bit microcode memory, the WISC Chips use on-chip stack and microcode memory. All processing logic is also included on-chip. Since the size of the stack memories and microcode memory is very large, the final Chips will have most of their area devoted to memory cells.

As a trade-off between functionality and memory sizes, both stack memories are 512 elements by 32-bits and the microcode memory is 2048 elements by 32-bits, giving a total of 256 possible op-codes.

The logic not included on the Chips includes the: host interface, program memory interface, program memory, and oscillator synthesis.

2.2 Chip Partitioning for Two-Chip Design

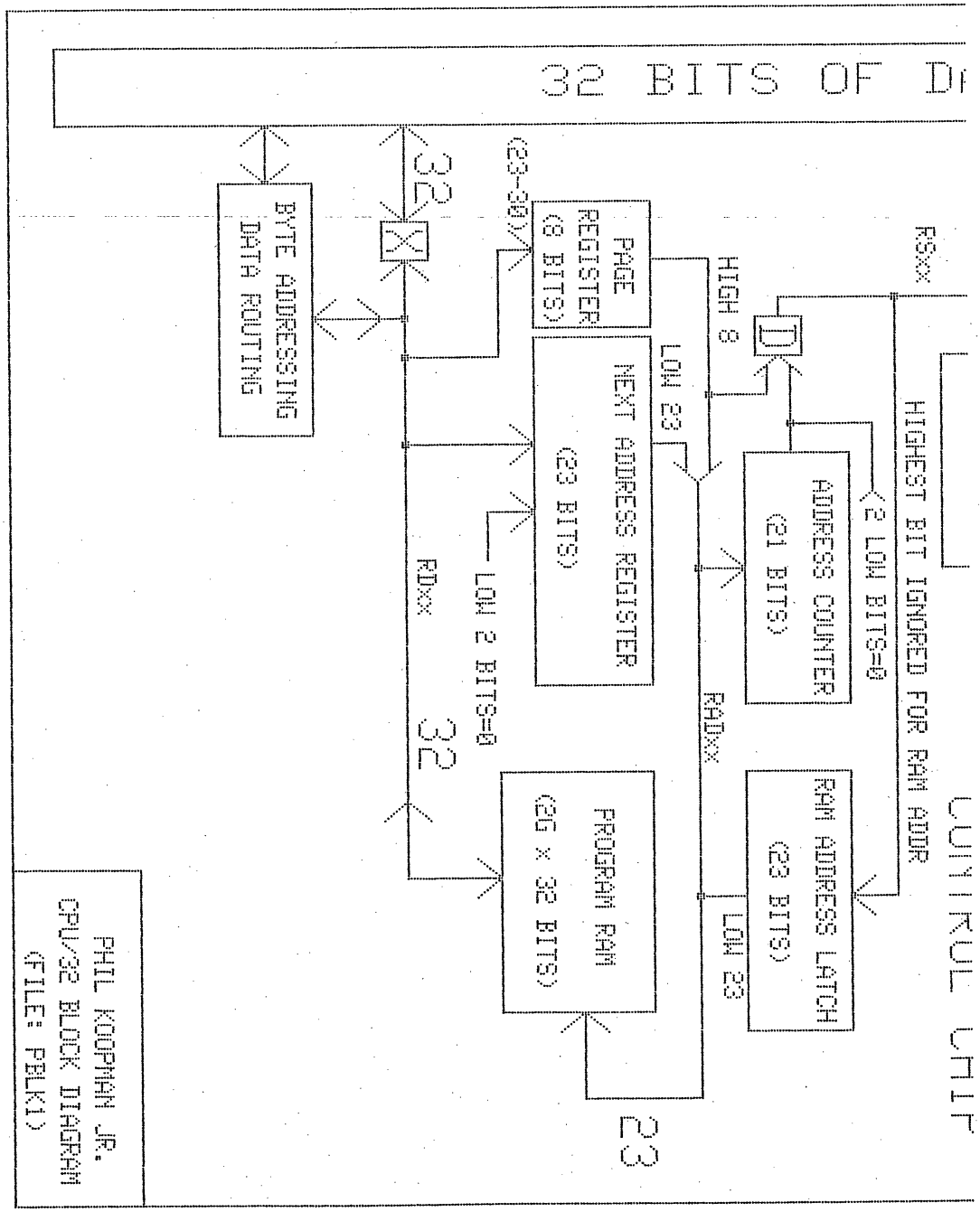
Since initial estimates for combined stack and microcode memory sizes indicated that a single chip implementation could be too large, a two-chip implementation was adopted. As is commonly the case, the cleanest way to divide the implementation was into a data path chip and a control path chip. As it turned out, this division put one of the stacks on each chip, and split up the microcode memory (12 bits on the data chip, 18 bits on the Control chip), thus giving a good partitioning in this memory-cell intensive design.

Figures 1 and 2 show the partitioning between the two chips at the block diagram level.

Appendix A describes the pinouts for the "lumped system", that is, the external interface of the Data Chip/Control Chip pair to the outside world. If the Data Chip and Control Chip were integrated into a single package, these are the only pins that would be required.

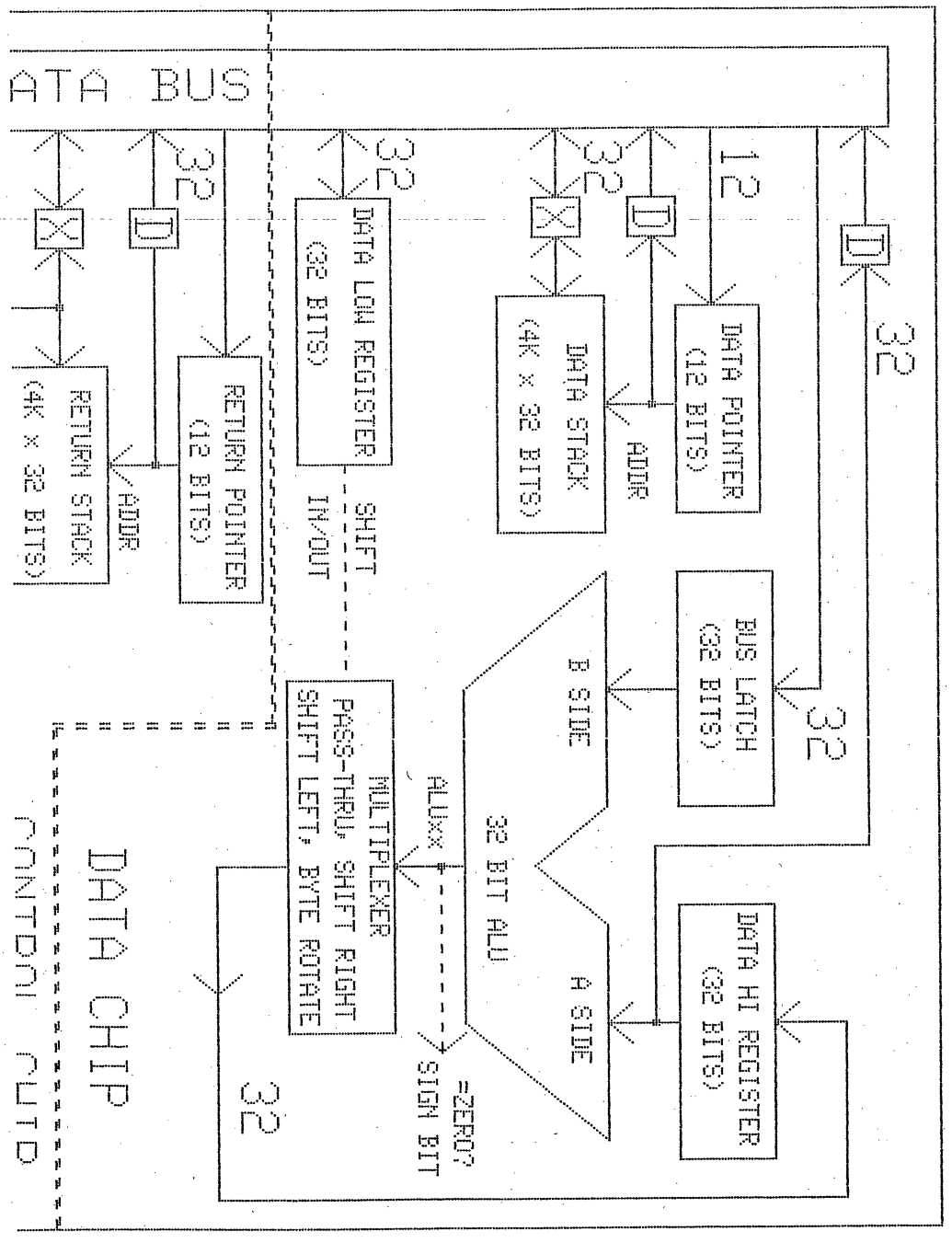
FILE:PBLK1 DATE:09-02-1987

GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 1



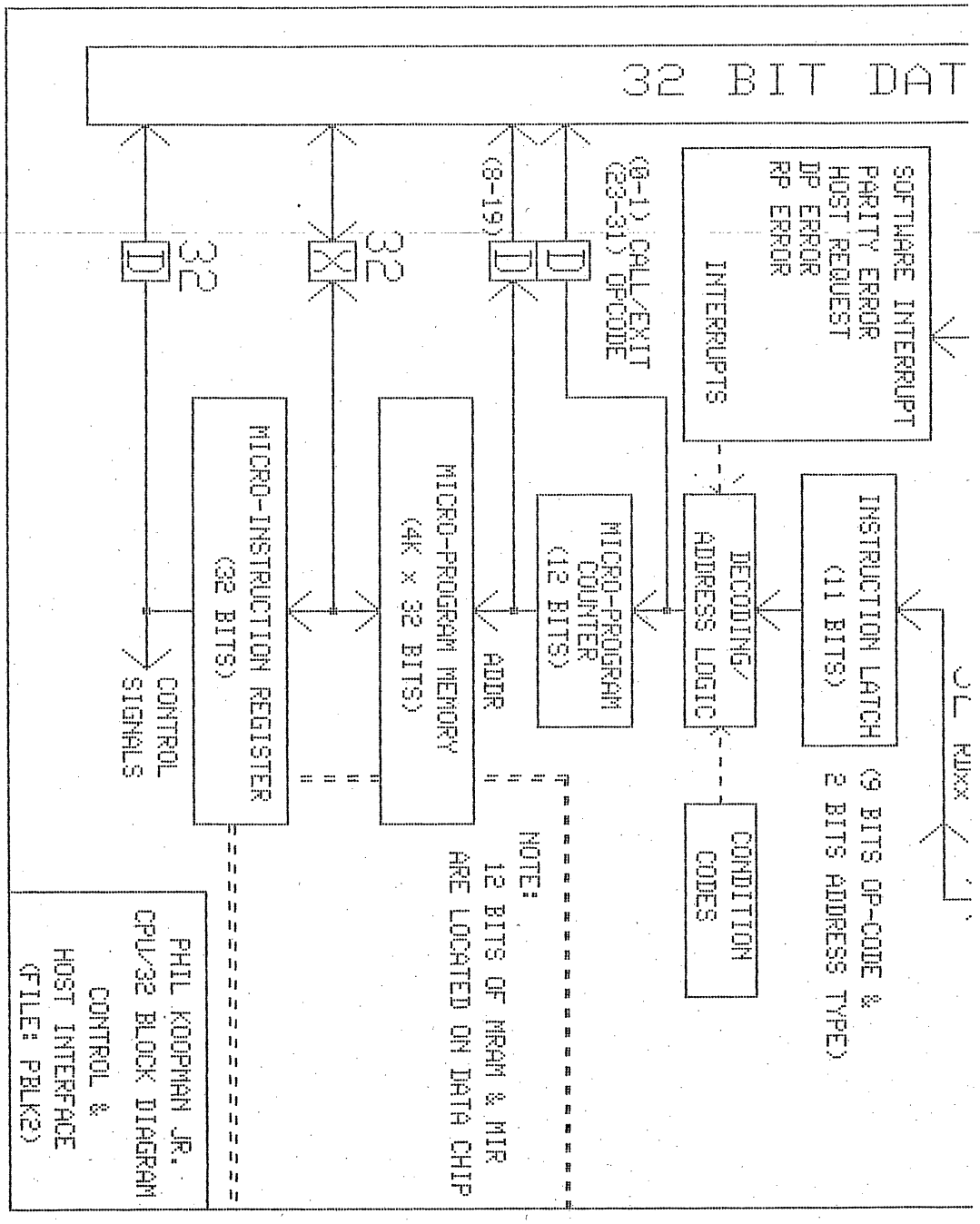
FILE:PBLK1 DATE:09-02-1987

GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 2



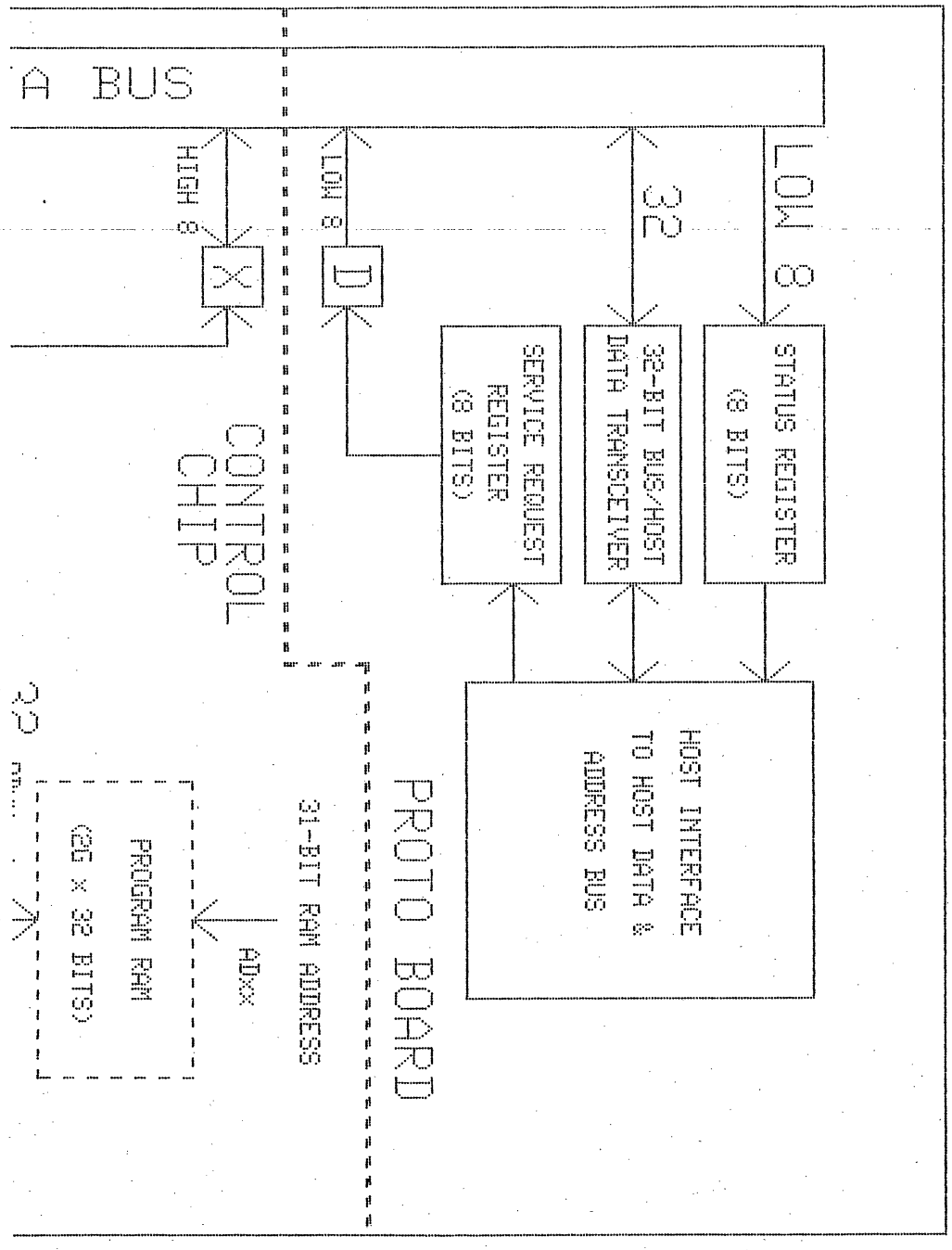
FILE:PBLK2 DATE:09-02-1987

GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 1



FILE:PBLK2 DATE:09-02-1987

GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 2



2.2.1 Data Chip

The Data Chip contains the ALU with its associated multiplexer, the Data High Register, the Bus Latch into the B-side of the ALU, the Data Stack and Data Stack Pointer, the Data Low Register, and twelve bits of the Micro-program Memory/Micro-Instruction Register structure (bits 8-9,12-13,16-23). These twelve micro-program bits were chosen for placement on the Data Chip because they directly control Data Chip resources (e.g. ALU function).

The data chip may be thought of as "the part that got off-loaded from the main chip", since the Data Chip does not interface with the outside world.

Appendix B describes the pinout for the Data Chip.

2.2.2 Control Chip

The Control Chip contains everything in the system that didn't get factored off onto the Data Chip. This includes the Return Stack, the memory addressing logic, the micro-instruction addressing logic, and almost all of the control circuitry. The Control Chip holds eighteen bits of the Micro-program Memory/Micro-Instruction Register structure (bits 0-7,10-11,24-31). Bits 14-15 of each micro-instruction are unused, and therefore omitted from Micro-program Memory.

The Control Chip may be thought of as the "main" chip, since it is the chip that controls all global actions and interfaces to the outside world. Appendix C describes the pinout for the Control Chip.

2.3 Differences Between Discrete and Chip Implementation

There are five types of differences between the Chip implementation and the Board implementation described in the WISC CPU/32 Preliminary Documentation. These differences are due to Engineering Changes to the Boards (which will appear in subsequent versions of the Board documentation), changes to eliminate floating buses in the Board design, changes due to the partitioning of the design into two chips, changes due to the use of synchronous stack and microcode memories, and changes required due the the high speed of operation in the CMOS chip (i.e. race elimination.)

2.3.1 Engineering Changes to Discrete Design

Two Engineering Changes have been made to the Board design in order to obtain correct operation of interrupts. The first change is changing IC38 pin 10 on figure 59 from being connect to the signal INTR to simply being connected to IC38 pin 9. This change in effect makes NDEC2 and ND2CK the same signal. On the Chip implementation all occurrences of ND2CK are changed to NDEC2, and the OR gate is simply eliminated.

The second change is changing the CLR pins on the EXIT and CALL flip-flops to be pulled high. This means that on figure 65,

IC57 pin 13 and IC58 pin 13 are disconnected from NINTR and instead connected to PULLG, which is a pull-up resistor.

Both these changes are reflected in the new versions of figures 59 and 65 included in Appendix D. These changes accomplish the changes in functions to interrupt processing described on the new page 51 of the Board documentation given in Appendix D.

2.3.2 Changes to Eliminate Floating Busses

Since floating tri-state busses are not tolerable on a CMOS IC, each bus on the Board design that had one or more bits that were allowed to float is changed to be driven on every clock cycle. This is accomplished by adding drivers for undriven bits, by adding a default bus driving source when the bus is not in use, or by adding pull-down resistors.

2.3.2.1 Return Stack

The Return Stack bus (RS<0:31>) connecting the Return Stack, Address Counter, and Address Latch was allowed to float when none of these resources was in use in the Board design. In the Chip design, the signal NRSOE on the figure Return Stack Control Logic (F.60) is defined to let the Return Stack drive the bus when the bus would otherwise be idle.

2.3.2.2 RAM Data

The RAM Data bus (RD<0:31>) floats whenever a RAM access is not being performed in the Board implementation. Although changing logic to allow the Chip to drive this bus during unused cycles is possible, software access to non-existent memory locations would allow the bus to float. Therefore, pull- down resistors are installed in the pads for the RAM Data bus. These pad drivers are shown in the Chip schematics figure entitled "BUS PAD DRIVERS with PULL-DOWN".

2.3.2.3 Micro-Program Memory

The Micro-Program Memory bus is allowed to float when in Slave mode in the Board implementation. In the Chip implementation, the signal NMROE enables the Micro-Program Memory outputs to drive the bus when it would otherwise have been idle. The logic to accomplish this is shown in the schematics figure entitled "MRAM/MIR CONTROL LOGIC -- F.24".

2.3.2.4 System Data Bus

The system Data Bus (BUS<0:31>) can float either due to being partially driven by system resources that are less than 32 bits wide (such as the Data Stack Pointer,) can float during the Divide and Multiply operations, and can float if no external source drives the bus during certain operations. On the Board implementation, this problem is handled by using pulldown-resistors on all 32 Data Bus lines. While this is convenient and

space-effective on a discrete board implementation, it is not acceptable for a high-speed CMOS implementation where the unused bits must be zeroed for correct program operation.

2.3.2.4.1 Partially Driven Bus Lines

In order to correct the problem of system resources less than 32 bits wide not driving the Data Bus properly, all system Data Bus sources less than 32 bits wide are expanded to 32 bits using tri-state bus drivers that drive zeros onto the data bus. These bus drivers are found in schematics: "pulldwns" for SOURCE=RAM-BYTE, SOURCE=FLAGS, SOURCE=RP, SOURCE=MPC and "DATA STACK POINTER -- F.25" for SOURCE=DP.

2.3.2.4.2 Divide and Multiply Operations

In order to correct the problem of the bus floating during multiply and divide micro-operations, the DHI register is driven to the bus during SOURCE=DHI as well as MULTIPLY and DIVIDE micro-operations. The 3-input AND gate that accomplishes this generates the signal used to enable the SN74244 devices on the schematic "Data High Register". This change does modify the functionality of the Chip compared to the Board in that during divide and multiply operations, the Board's Data Bus shows all zeros, whereas the Chip's Data Bus shows the contents of the DHI register. Because of this, the Data Bus values are "don't cares" when testing a chip and using the MULTIPLY or DIVIDE micro-operations.

2.3.2.4.3 Unused/External Bus Sources

In order to correct the problem of bus floating for Data Bus source number 15, the implemented instruction set does not use the SOURCE=FPU bit pattern. This precaution should be sufficient for normal operation. However, to guard against a floating data bus for illegal or improper micro-instructions which could conceivably be generated due to programmer errors or an unfortunate power-on micro-instruction bit pattern, the unused source decoder SN74138 output pin (Y7 on the lower left '138 of the schematic "BUS SOURCE & DESTINATION DECODERS -- F.23") should be AND'ed with the '138 output that currently drives the signal NSPC on that schematic to make the new NSPC signal. Making this change will designate the PC host to drive the Data Bus for both source number 0 and source number 15, leaving no undefined bus source situations.

2.3.3 Changes Due to Two-Chip Partitioning

Other than the obvious changes of adding pad drivers for inputs and outputs, the two-chip partitioning scheme requires changing logic to correctly orchestrate the flow of bits on the Data Bus, and requires replication and changing of the system clock logic to avoid clock skew and reduce the number of pins required on each chip.

2.3.3.1 Bus Direction Control

Since the System Data Bus may be driven from the Data Chip, the Control Chip, or the Host, special logic at each chip must be used to control the direction of the pad drivers. Since some bus sources have different bits of the bus driven from different components of the system, groups of pads within each chip may be driven in different directions during the same clock cycle.

Also, since the RAM Data Bus has pad drivers associated with it, logic to control the direction of the pad drivers is included in the Control Chip.

2.3.3.1.1 Data Chip System Bus Driving

The Data Chip's System Bus pads are driven by the gates shown in the schematic "WISC DATA_CHIP". These gates drive all 32 bits of the Data Bus as chip outputs for micro-operations SOURCE=DLO, SOURCE=DS, SOURCE=DP, SOURCE=DHI, DIVIDE, and MULTIPLY.

For the micro-operation SOURCE=MRAM and the MIR source to BUS input pin function (signal NSMIR,) bits 8-9, 12-13, and 16-23 are driven off the chip while all other bits are driven onto the chip. The bits driven off the chip correspond to the micro-instruction bits residing on the Data Chip.

For the MIR loading operation (signal NDMIR), all pins are configured to inputs regardless of the current micro-operation. If this were not done, any micro-operation specifying outputs could block the new micro-instruction from being loaded. (Optimization note: the use of the NDMIR signal to control pin directions appears redundant, since the schematic "BUS SOURCE & DESTINATION DECODERS -- F.23" shows that all bus sources are disabled during the NDMIR operation, making these pins default to inputs.)

For all other micro-operations, all 32 bits of the system Data Bus are configured as inputs.

2.3.3.1.2 Control Chip System Bus Driving

The Control Chip's System Bus pads are driven by the gates shown in the schematic "WISC CONTROL CHIP". These gates drive all 32 bits of the Data Bus as chip inputs for micro-operations SOURCE=DLO, SOURCE=DS, SOURCE=DP, SOURCE=DHI, DIVIDE, and MULTIPLY and for the default bus source designated by the NSPC control line (the default bus source is from the host PC's interface.)

For the micro-operation SOURCE=MRAM and the MIR source to BUS input pin function (signal NSMIR,) bits 8-9, 12-13, and 16-23 are configured as inputs to avoid a bus crash with these bits from the Data Chip. They are driven onto the Control Chip's BUS but are not used. All other bus bits are configured as outputs, since they correspond to the micro-instruction bits residing on the Data Chip. In this implementation the two unused micro-

instruction bits (14-15) are driven to ground whenever the MRAM of MIR are read.

For the micro-operation SOURCE=FLAGS, bits 0-7 of the Data Bus are configured as inputs so that the host Service Request Register may be driven onto BUS from outside the chips. Bits 8-31 are configured as outputs to drive the interrupt status flags and unused (forced to zero) bits onto BUS.

For the MIR loading operation (signal NDMIR), all pins are configured to inputs regardless of the current micro-operation. If this were not done, any micro-operation specifying outputs could block the new micro-instruction from being loaded.

For all other micro-operations, all 32 bits of the system Data Bus are configured as outputs.

2.3.3.1.3 Host System Bus Driving

The external Host Interface is required to drive all 32 bits of the system Data Bus when the NSPC control signal is active. NSPC is generated by the default source micro-operation (designated by the lack of a SOURCE= or the use of SOURCE=HOST in a micro-instruction.) This function is used in Slave Mode so that the X! operation can be used to drive BUS when no internal bus source is specified. In master mode, NSPC is used to signal the host to drive BUS in order to avoid letting BUS float.

The external Host Interface is required to drive bits 0-7 of the system Data Bus when the NSFLG control signal is active. These 8 bits are driven from the host's Service Request Register. The Service Request Register is an off-chip 8-bit register used to communicate the type of service required when the host generates an interrupt to the chip.

2.3.3.1.4 RAM Address Bus Driving

The RAM Address Bus is driven off the data chip for the micro-operations: DEST=RAM, DEST=RAM-BYTE, and DEST=PAGE. The RAM Address Bus is also driven off-chip for the micro-operation DEST=DECODE, except when a SOURCE=RAM is also used, to allow for the important capability to perform a "manual" decode of an instruction from RAM as in the micro-instruction "3 :: SOURCE=RAM DEST=DECODE ;;" .

2.3.3.2 Clock Logic

In the Board implementation, the system clock was generated on one board, and copies were buffered on each circuit board to generate local copies of the clock. In the Chip implementation, this method would cause a clock skew on the Data Chip, or would lead to the use of delay inverters in the Control Chip. To avoid this problem, copies of the clock generating logic are placed on both the Data Chip (schematic "DATA CLOCK CONDITIONING") and the Control Chip (schematic "CLOCK CONDITIONING".)

Instead of transmitting copies of FASTC and XCLK from a central clock generator, the Data Chip and the Control Chip each

use separate copies of the clock conditioning logic to synthesize CLOCK and FASTC from the NCYCL, DVOSC, and NMAST inputs.

As an additional change, since CLOCK and FASTC are not available off-chip, the "WISC Control Chip" schematic shows three OR gates to condition the signals NDRB, NDRW, and NRAM with FASTC to generate RAM control signals synchronized with the system clock and to eliminate spurious signals caused by decoding glitches from the bus control demultiplexers. On the Board implementation, the system clock was distributed to the memory boards to accomplish this synchronization.

2.3.4 Changes Due to the RAM Compiler

In the Board implementation, asynchronous 4K x 32-bit memories are used for both stacks and the Micro-Program Memory. Since Harris' RAM compiler allows the use of RAMs with an individual output enable line instead of a combined CS/OE pin, extra logic was generated to create the output enable function (which will eventually improve system performance).

Unfortunately, the RAM compiler also requires the use of synchronous RAM. This required the synthesis of a chip enable signal, all will significantly hurt performance as discussed in Section 9.

2.3.4.1 Data Stack

The changes in the Data Stack control signals are very straightforward. The Data Stack memory's chip enable is tied to the system clock. This means that the high part of the clock cycle is used for precharge and the low part is used to read/write data. See Section 3 for a discussion of the high/low portions of the clock cycle.

The Data Stack output enable is activated whenever a SOURCE=DS micro-operation is used.

2.3.4.2 Return Stack

The Return Stack memory's chip enable is also tied to the system clock. As in the Data Stack, this means that the high part of the clock cycle is used for precharge and the low part is used to read/write data.

The Return Stack output enable is activated whenever the Return Stack is read and whenever the RS bus would otherwise be idle. This means that the Return Stack output enable is active: when a SOURCE=RS micro-operation is used; during the first clock cycle of a subroutine return; any time the following signals are all inactive: NDADC, NDADL, NSADC, NDRS and NCAL2. (Note: NCAL2 is active during the DEC2 cycle of a subroutine call.)

2.3.4.3 Micro-Program Memory

The changes in the Micro-Program Memory control signals are very straightforward. The MRAM's chip enable is tied to the system clock. This means that the high part of the clock cycle

is used for precharge and the low part is used to read/write data.

The MRAM output enable always activated, except when a DEST=MRAM micro-operation is used or a write to the MIR is performed (using signal NDMIR).

2.3.5 Changes Due High Speed Operation

Surprisingly few changes were required to compensate for timing differences between relatively slow ALS and LS logic used in the Board implementation and the faster CMOS process used by Harris.

2.3.5.1 MRAM Transceiver Vs. MIR-Clock Race

The one required change to the design due to the logic speed difference resolves a race condition between the signals MIRCK and NMRXE on schematic "MRAM/MIR CONTROL LOGIC -- F.24". The signal MIRCK is used to clock the value on the internal MRAM data bus into the Micro-Instruction Register. In Slave Mode, this data bus is driven by SN74245 transceivers from the system Data Bus (see schematic "Cmicro_ram".)

The enable signal for those '245 transceivers is derived from the signal NDMIR in the Board implementation. But, since NDMIR is also used to generate MIRCK in Slave Mode (see schematic "CLOCK CONDITIONING",) a race develops between the signal trying to turn off the data transceiver and the signal trying to clock that output of the transceivers into the MIR. In order to resolve this race, a latch was added to the "MRAM/MIR CONTROL LOGIC - F.24" schematic to force NMRXE to remain active until after MIRCK has triggered the MIR clock.

This race was not detected on the Board implementation, since the difference between two equal-depth paths of ALS SSI logic gates is much less than the time it takes a tri-state device to disable. (In other words, the designer missed the race, but got lucky. No system failures, instabilities, or manufacturability problems have resulted, since tri-state turn off times combined with voltage bleed-off times for a floating bus are VERY long compared to a gate delay.)

2.3.5.2 Changes in the Use of FASTC

The signal FASTC (Fast-Clock) was used in the Board implementation to ensure that the rising edges of RAM writes and SN74373 transparent latch clocks occurred before the rising system clock edge disrupted the values on the system Data Bus. This was required because some static RAM devices and all '373 devices have a 5 to 10 ns data hold requirement after the rising clock edge.

Since the RAM Compiler generated RAMs do not have a hold requirement, FASTC is no longer required for stack and Microcode Memory RAMs (although the schematic "MRAM/MIR CONTROL LOGIC - F.24" still appears to use FASTC).

As the SN74373 devices used on the Chips have a data hold requirement, FASTC is still used for these devices.

Since some chips used for Program Memory may have a hold requirement (or may be subject to propagation delays due to control circuitry), FASTC instead of CLOCK is used to condition the signals NRAM, NDRB and NDRW just before the output pins for these signals on the control chip (schematic "WISC CONTROL CHIP").

3. THEORY OF HARDWARE OPERATION

This discussion of the theory of the hardware operation is intended only to supplement the WISC CPU/32 Preliminary Documentation. This section discusses clock control timing, amplifies upon the information about macro- instruction processing, and discusses details of interrupt processing.

3.1 Slave Mode

Whenever the system input NMAST is high, the system is in Slave Mode. When idling in Slave Mode, DVOSC must be high and NCYCL must be high. Of course, NDMA, NSMIR, and NDMIR must also be inactive=high. When idling in Slave Mode, the system clock (CLOCK -- for the purposes of Section 3 there is no difference between CLOCK and FASTC) idles high. Since CLOCK is not cycling, no micro-instructions are being executed, and the system is halted.

3.1.1 Loading a Micro-Instruction

In order to accomplish anything in Slave Mode, the MIR must first be loaded with a valid micro-instruction. This is accomplished by having the host interface place a 32-bit value on the Data Bus while driving the signal NDMIR low. This same 32-bit value can be read back to the host via the Data Bus by driving the NSMIR signal low. Since only 30 bits of the 32-bit MIR field are actually used by the Chip, bits 14-15 of the read back value will be forced to zero. This differs from the way the Board works, since the Board stores but does not use all 32 bits of the MIR value.

NSMIR and NDMIR must be held low for a period at least as long as the low part of the clock cycle in Master Mode. The rising edge of NDMIR causes the signal MIRCK to clock the data value into the MIR.

Since the host must be able to read and write the MIR at will, the MIR can not itself hold a SOURCE or DEST field that specifies the MIR as a bus source or destination. This would cause a chicken-and-the-egg problem of not being able to write the MIR unless the MIR already had a DEST=MIR micro-operation in it. Instead, the NDMIR and NSMIR signals are used to over-ride the bus source and destination fields in the MIR to read and write the MIR upon demand in Slave Mode.

Since the MIR provides access to all system resources, a global reset pin is not required for the Chips. All initialization can be performed by an appropriate sequence of micro-instruction loads and executions.

3.1.2 Executing a Micro-Instruction

Once the MIR is loaded with a valid value using the micro-assembler (e.g. ">> SOURCE=DS ALU=A+B DEST=DHI ;SET",) the micro-instruction may be executed by single-stepping the system

clock. The clock is cycled by driving the signal NCYCL low and then driving it high. NCYCL must be held low for a period at least as long as the low part of the clock cycle in Master Mode. The falling edge of NCYCL must be delayed beyond the rising edge of NDMIR, NSMIR, or the previous NCYCL rising edge at least as long as the high period of CLOCK in Master Mode.

The main difference between Slave Mode operation and Master Mode operation is that the signal MIRCK is not driven by CLOCK in Slave Mode. This means that the MIR remains unchanged no matter how many times NCYCL is cycled.

3.1.2.1 Single Stepping the Clock

While NCYCL is low, the data bus is driven from whatever bus source was specified in the micro-instruction. If no bus source was specified, then the low=active value of signal NSPC indicates that the host interface must drive BUS<0:31>. If NSPC is high, or no Data Bus destination is specified, then the NCYCL merely executes a single-step of the clock.

3.1.2.2 X@ Operations

If a bus source (and optionally, a bus destination) are specified, then the host may read the Data Bus while cycling the clock (using the X@ micro-assembler command.) The data for X@ may be clocked into host holding registers as NCYCL is released, or may be examined 8 or 16 bits at a time by holding NCYCL low for longer than the minimum clock low time, then examining the Data Bus values as desired. In any event, NCYCL should be driven high when the X@ operation is completed.

Since the X@ operation cycles the clock, a single micro-instruction may be repeated for operations such as popping successive data stack elements without reloading the MIR between operations.

3.1.2.3 X! Operations

If a bus source is not specified, then the host is allowed to drive BUS<0:31> with values using the X! micro-assembler operation. While NCYCL is driven low as in the above operations, the host drives BUS. As NCYCL is driven back high, whatever bus destinations were specified are loaded with the BUS contents on the rising edge of CLOCK (which is driven by the rising edge of NCYCL.)

The data driven onto the bus must be valid in time to make it through the ALU and into the DHI register (and also set the zero condition code.) This is the same time specified in Section 4.1.2 on ALU critical path timing.

If the host has less than a 32-bit data path, then the host interface must build a 32-bit word out of smaller pieces, then perform a single High-Low-High pulse of NCYCL with the complete 32-bit data word driven onto BUS. In practice, NCYCL may be held low while building the 32-bit word, as long as all 32 bits meet

the setup time before driving NCYCL high.

Since the X! operation cycles the clock, a single micro-instruction may be repeated for operations such as pushing successive data stack elements without re-loading the MIR between operations.

3.1.3 DMA Transfers

The DMA Transfer Mode is a special case of the Slave Mode. In order to transfer a block of sequential Program Memory words to or from the host, first the Address Counter is loaded with the address of the first word to transfer. Then, the MIR is loaded with the micro-instruction: ">> DEST=RAM INC[ADC] ;SET" for a DMA write-to-Program-Memory operation, or ">> SOURCE=RAM INC[ADC] ;SET" for a read-from-Program-Memory operation.

After the MIR has been properly set, the NDMA line is driven low and held low for the duration of the DMA transfer. This action causes the Address Latch to become transparent regardless of the CLOCK value, in effect driving the RAM Address pins (RAD<0:22>) directly from the Address Counter. After waiting a few nanoseconds after NDMA goes low for the address value to get to the RAD pins, the DMA transfer may begin.

As far as the Control Chip is concerned, the DMA transfer is no different than a sequence of X@'s or X!'s, with the exception that the RAM Address pins are driven from the Address Counter. NCYCL performs a High-Low-High transition for each 32-bit transfer word. NCYCL must remain low long enough to read or write program memory (depending on memory implementation, this low period may be up to twice as long as the required low period at full speed, but most hosts are slower than this anyway.)

The DMA mode does not necessarily imply that DMA is being performed to or from the host PC's memory. While DMA'ing out of the host PC's memory and into the Chip's Program Memory is obviously the fastest way to do a transfer, the host may also do a series of X@'s or X!'s if the DMA circuitry to the host's bus is not implemented.

3.2 Master Mode

Master Mode is the "normal" mode of Chip operation. Whenever NMAST is active=low, the Chip is placed in Master Mode. In this mode, DVOSC supplies a one-third low, two-thirds high duty cycle oscillator, while NDMA, NSMIR, NDMIR, and NCYCL must all remain high.

In Master Mode, CLOCK and FASTC are one-third high/two-thirds low, and MIRCK follows CLOCK with a delay of a few nanoseconds to ensure that system Data Bus values are successfully clocked in to registers or memory before the next micro-instruction is executed. RALCK is essentially an inverted copy of FASTC, since the SN74373's used to implement the RAM Address Latch allow data to pass when the input clock is high (in this case, during the low period of FASTC).

3.2.1 Executing a Micro-Instruction

Each micro-instruction in Master Mode is executed during a single high-low period of CLOCK. All micro-instructions begin at the beginning of the one-third high period, and end at the rising edge of the next high period.

3.2.1.1 High Clock Period

The one-third high period of the system CLOCK is the setup phase of the micro-cycle. Just after the rising edge of CLOCK the signal MIRCK rises, clocking the micro-instruction to be executed in the current clock cycle into the MIR register.

When the new MIR value is loaded, the SN74138 decoders shown in "BUS SOURCE & DESTINATION DECODERS -- F.23" decode the source and destinations for the system Data Bus. Since this decoding process can produce spurious values on the '138 outputs, all clocking signals derived from the destination selectors (NDDP .. NDMRA) are conditioned with CLOCK to mask selections during the high clock period.

Also during the high clock period, the stack and micro-program RAMs have a high chip enable input, allowing them to precharge before use.

3.2.1.2 Low Clock Period

The two-thirds low period of the system CLOCK is the execution phase of the micro-cycle. During this phase, the current contents of the MPC, the two JMP=xxx bits from the MIR, and the condition code multiplexer output are all used to address the next micro-instruction in MRAM.

Also during this phase, the Data Bus source is allowed to complete its "turn-on" to drive BUS, while the selected destination is driven from BUS. In all cases, write/load signals to registers and memories are derived from CLOCK. This means that all clocks and transparency enables (for transparent latches) are disabled during the high phase of CLOCK and enabled during the low phase of CLOCK. At the rising edge of CLOCK at the end of the low period, registers and latches grab the Data Bus value for retention into the next clock cycle.

The slight delay between the rising edge of CLOCK, which denotes the end of the current clock cycle, and the rising edge of MIRCK, which clocks in the next MIR value, ensures that no control inputs will change until after the rising edge of CLOCK has clocked in data, incremented counters, etc.

3.2.2 Executing a Macro-Instruction

The actions for executing a Macro-Instruction are discussed from a register-transfer viewpoint in the WISC CPU/32 Preliminary Documentation. Since the hardware actions during an interrupt have been modified slightly by an engineering change, pages 34 through 35 and page 51 of the manual have been modified and are

included in Appendix D.

The preliminary documentation discusses macro-instruction operation from a register transfer point of view. The following sections discuss macro-instruction operation from a functional point of view.

Since each each macro-instruction executed by the Chip contains both a 9-bit opcode and a 23-bit memory address (with the bottom two bits zero), each macro-instruction contains the address of the next instruction to be executed. This makes every macro-instruction (at this point in our discussion) an unconditional jump.

Since the Chip uses a word-aligned memory organization, all instructions must begin on full-word boundaries. Thus, the bottom two bits of the 23-bit address are always zero when addressing a 32-bit full-word in memory. Conveniently, this allows the lowest 2 bits of a macro-instruction to be used as control bits to specify an unconditional jump, subroutine call, or subroutine exit (with jump and call addresses having an implicit low order 2 bits of zero.)

3.2.2.1 Unconditional Jump

Whenever the bottom two bits of a macro-instruction are "00", the Chip executes an unconditional jump. Unconditional jump processing is very straightforward. At the end of the DEC0 cycle, the pending instruction (whose 32 bits are contained in the Instruction Latch and Next Address Register) begins the sequence leading to its execution. At the rising edge of MIRCK at the end of the DEC0 cycle, the 9 bits of the Instruction Latch are clocked into the Micro-Program Counter to begin addressing the first micro-instruction of the new opcode.

During the DEC1 cycle of the unconditional jump (which corresponds to the last micro-instruction of the macro-instruction currently executing, that is, the END micro-instruction) the MPC together with an JMP=000 micro-operation, implicitly invoked by END, address the first micro-instruction for the new opcode in the MRAM. This new micro-instruction will be available for use in the MIR during the next clock cycle.

Also during the DEC1 cycle, the contents of the Next Address Register are driven out onto the RAM Address Bus. This begins the fetching sequence for the next macro-instruction. Note that even while the next macro-instruction is being prepared for execution via the MPC to MRAM to MIR pipeline, the macro-instruction subsequent to that is being fetched from Program Memory via the Program Memory to Next Address Register pipeline.

During the DEC2 cycle, the first micro-instruction of the macro-instruction beginning execution is loaded into the MIR (at the very beginning of DEC2 on the rising MIRCK edge) and executed. Also, the Next Address Register continues to drive the RAD pins, and the outputs of Program Memory on the RD bus are clocked into the Next Address Register and Instruction Latch

(that is, the pending instruction position for the next macro-instruction) at the end of the clock cycle.

When macro-instructions only require two micro-instructions for execution, the DEC2 cycle of one macro-instruction overlaps the DEC0 cycle of the next macro-instruction. Since the DEC0 instruction actually does little more than prepare the system for the macro-instruction decoding sequence, there is no conflict between these two operations. The only exception is that the Instruction Latch, which is being loaded by the DEC2 cycle, must be transferred with its new contents directly to the MPC during the DEC0 cycle. To accomplish this, the IL is implemented as a transparent latch instead of a register to allow flow-through operation during the low clock cycle period.

3.2.2.2 Subroutine Call

Subroutine calls, denoted by a "10" bit pattern in the low order two bits of a macro-instruction, are very similar to unconditional branches. The only difference between calls and jumps is that the address after the address of the calling macro-instruction in the calling subroutine must be saved for an eventual subroutine exit. This saving is accomplished via the Address Counter to Return Stack data path.

During all DEC2 cycles (including unconditional jumps) the Address Counter is clocked with the value on the RAM Address Bus. This saves the address from which the pending instruction has been fetched. During the DEC1 cycle of that pending instruction's execution initiation, the Address Counter is incremented by 4. If the pending instruction happens to be a subroutine call, the Return Pointer is also decremented during the DEC1 cycle.

During the DEC2 cycle of the pending call instruction's execution initiation, the Address Counter contains the value of the return address, and the Return Stack contains an allocated word for saving the return address. So, during the DEC2 cycle of a call operation, the Address Counter is written to the Return Stack. Call operations may occur repetitively, since the new Address Counter value clocked in during the DEC2 cycle destroys the old Address Counter value only after it has a chance to be written to the Return Stack during the DEC2 cycle.

It is very important that all microcode which does a DEST=DECODE also saves the memory address from which the instruction was fetched in the Address Counter, in case the macro-instruction placed in the pending instruction registers happens to be a subroutine call. This is typically accomplished by specifying a DEST=ADDRESS-COUNTER instead of a DEST=ADDRESS-LATCH when setting up the memory address prior to a "SOURCE=RAM DEST=DECODE" operation.

3.2.2.3 Subroutine Exit

Subroutine exits, denoted by a "01" bit pattern in the low

order two bits of a macro-instruction, differ from unconditional branches in that the address for the next macro-instruction is obtained from the Return Stack instead of the Next Address Register. The address field of subroutine exit macro-instructions is unused.

Since a subroutine call places its return address on the top of the Return Stack, the subroutine exit function routes the top element of the Return stack through the Address Latch to fetch the next macro-instruction in the calling routine. During the DEC1 cycle of a subroutine exit, the Return Stack drives the RS bus, and the Address Latch is made transparent. The Address Latch outputs are also enabled, allowing the return address on the Return Stack instead of the Next Address Register to drive the RAM Address Bus. At the end of the DEC1 cycle the Address Latch control signal goes high, latching the return address for use in the DEC2 cycle.

During the DEC2 cycle of a subroutine exit, the Address Latch is still used to drive the RAD bus instead of the Next Address Register. Also, the Return Pointer is incremented, popping the return address from the Return Stack. As might be expected, the Address Counter is clocked with the contents of the RAD bus (which points to the macro-instruction being fetched) for use in the case of a subsequent subroutine call, and the Next Address Register and Instruction Latch are loaded with the pending instruction being fetched from Program Memory.

3.2.3 Interrupts

Interrupts on the CPU/32 are synchronized with the macro-instruction decoding cycle. This greatly simplifies the interrupt processing logic by eliminating the need for restartable microcode and reducing the state of the machine that must be saved when processing interrupts. Since interrupts only occur between opcode executions, only the DHI, DP, RP, and ADC registers need be saved for restarting after an interrupt. Of course the Data and Return Stacks must not be corrupted during interrupt processing, but this is an easy constraint to satisfy for small interrupt service routines. For larger interrupt servicing requirements such as true unsynchronized task swapping, the Data and Return Stacks must be offloaded into Program Memory to provide a fresh environment for the new task.

3.2.3.1 Interrupt Causes

Interrupts may be caused by stack pointer overflows/underflows, a host service request (in which the host loads the service request register with an 8-bit value), an external interrupt source designated NPRTY (contemplated for use with memory parity errors), and software interrupt requests caused by loading non-zero values into the interrupt flag register. Interrupts may be masked by loading a 1 bit into the highest bit of the interrupt flag register.

3.2.3.2 Interrupt Synchronization

If an interrupt, particularly a stack overflow/underflow interrupt, is generated by the last two clock cycles of a macro-instruction's execution, it will not be processed until after the next macro-instruction, since such interrupts miss the window during the beginning of the DEC0 cycle (described in the next subsection) of the macro-instruction interpretation sequence. In practice, this means that if a stack underflow occurs on a two-clock cycle primitive such as "DROP" in the sequence "DROP DUP SWAP", the "DUP" macro-instruction will execute, and an interrupt will be processed instead of the "SWAP" macro-instruction. Since the Data and Return Stacks provide a buffer area above and below the active stack area, this causes no problems as long as no macro-instruction attempts to push or pop more than 32 of the 512 stack elements.

3.2.3.3 Interrupt Servicing

When an interrupt sets a bit in the flag register, no interrupt is actually observed by the system until the next DECODE/END sequence. When the next DEC0 cycle occurs, the logical OR of all the interrupt bits is clocked into a flip-flop to generate the INTR signal. This INTR signal generates an interrupt condition during the DEC1 and DEC2 phases of the macro-instruction interpretation sequence.

During an interrupt, the input address to the MRAM is forced to a value of opcode page 1, regardless of MPC contents. The normal unconditional jump, subroutine call, or subroutine exit actions continue during the DEC1 and DEC2 cycles. During the DEC2 cycle, the first micro-instruction of the interrupt servicing opcode (which must be opcode 1) executes. During the DEC2 cycle, the interrupt servicing microcode must read the value from the Address Counter and placed in the DLO register. This step is vital, as it captures the incremented value of the address used to fetch the macro-instruction preempted by the interrupt. This fetched value, when 4 is subtracted, provides the restart memory address for the RTI instruction when interrupt processing is completed.

The interrupt servicing microcode word must also capture the values of the CALL and EXIT bits by reading the MPC value to undo any Return Pointer manipulations done by the call or exit hardware. Opcode 1 must also read the interrupt flags and save the value, then mask interrupts by setting the highest interrupt flag bit to avoid an infinite progression of successive interrupts.

3.2.3.4 Restarting After an Interrupt

Restarting after an interrupt is accomplished by reloading the interrupt flag register with an unmasked value (taking care to examine the register first for interrupts that have occurred

while processing the current interrupt), restoring the RP, DP, and DHI values, placing the restart address in the Address Latch and Address Counter, and then performing a macro-instruction decode on the macro-instruction addressed by the restart address.

4. CRITICAL PATH ANALYSIS & PERFORMANCE EVALUATION

The intent of this section is not to simply provide a final result for maximum system speed. Rather, it is intended to provide information about how to determine this speed after changes and enhancements have been made to the Chip implementation.

Timing results discussed in this section refer to non-optimized implementation simulation results obtained in August 1987, and should be used as a rough guide to system performance for the first-pass chip design. Some of the timings given are reconstructions and good estimates of simulation performance. After all optimizations have been completed, these measurements should be repeated to obtain the anticipated maximum speed of the final implementation.

TABLE 1
Simulator Results for Critical Paths Within Subsystems

Time (ns)	Path
23	Data Bus source select (Data Chip)
26	Data Bus destination select (Data Chip)
33	Data Bus through ALU before CLOCK goes high
27	DS to Data Bus after CLOCK goes low
27	RS to Data Bus after CLOCK goes low
38	RS to RAM Address Pins after CLOCK goes low (exit function)
13	Data Bus to RAM Address Pins after CLOCK goes low
11	Data Bus to RAM Data outputs
11	RAM Data input to Data Bus
12	RAM Data input to MPC
7	Inter-chip Data Bus delay
18	MRAM address valid at Data Chip after CLOCK goes low

4.1 Subsystem Timing Analysis

Since the CPU/32 is organized as a collection of subsystems connected by a system-wide bus, the best way to analyze system performance is by looking at the times required to get values from each subsystem onto the data bus, and the time required to get a values from the data bus into each subsystem. Breaking the analysis into source and destination halves eliminates the combinatorial explosion associated with exhaustively examining source/destination pairs. Since bus sources and destinations are driven by the source/destination decoding logic, it further makes sense to treat this logic as a separate entity common to all transfer paths.

As a result of this analysis, the following sections

describe the critical path circuitry for bus source/destination decoding, data bus sources, data bus destinations, Program Memory addressing, and micro-instruction addressing and fetching. The measurement methods given are obviously intended for use with a simulator.

4.1.1 Bus Source/Destination Decoding

Since all Data Bus transfers are ultimately controlled by the bus source and bus destination fields of each micro-instruction (microcode bits 0-7), the source/destination decoding logic forms a critical path for all micro-instructions.

Since bus sources can not drive the bus until they have a valid source select signal, the critical path controlling bus sources is: MIRCK clocking bits 0-3 of the MIR (see schematic "Cmicro_ram"); bits 0-3 of the MIR providing the SRC<0:3> inputs to the '138 source decoders ("BUS SOURCE & DESTINATION DECODERS -- F.23"); and these same '138 decoders providing control signals NSPC .. NSMRA. Of course, for the Data Chip, these control signals must also go through two pads to become available. Therefore, source control signals should be measured at the '138 outputs for the Control Chip and at the input side of the pads for the Data Chip.

Table 1 contains the simulated results for bus source selection and other values. Critical path measurements for bus sources should measure the time between MIRCK and when the bus source receives a signal to drive the bus.

Since bus destinations must have their clock inputs masked until there is a valid destination select signal, the critical path controlling bus destinations is: MIRCK clocking bits 4-7 of the MIR (see schematic "Cmicro_ram"); bits 4-7 of the MIR providing the DST<0:3> inputs to the '138 source decoders ("BUS SOURCE & DESTINATION DECODERS -- F.23"); and these same '138 decoders providing control signals NDDP .. NDMRA.

The critical path measurements shown in Table 1 for bus destinations measure the time between MIRCK and when the subsystem control logic receives a valid signal designating it as a bus destination. Since this signal must be further conditioned with the clock and other signals, the minimum length of the high portion of the clock cycle due to destination decoding is the time from the rising CLOCK (which drives the rising MIRCK) to the time the conditioned bus destination reaches the masking gate with CLOCK or FASTC. In the case of the simulated hardware, this path extended to the NRSWE signal on "RETURN STACK CONTROL LOGIC -- F.60" for the Control Chip and the LATEN signal on "SHIFT INPUT CONDITIONING" for the Data Chip. The number given in Table 1 represents the minimum time between the rising value of CLOCK and the falling value of FASTC based on the Data Chip path.

4.1.2 ALU

Since data from the Data Bus is routed through the ALU

before being clocked into the DHI register, the ALU data path constitutes the only critical path for bus destination functions during the low clock cycle period. The Data Bus is not driven into the ALU until the low-going edge of FASTC to prevent false triggering of the Bus Latch connected to the ALU B side. Thus, the timing requirements for an ALU function involving the B side are: the low-going edge of FASTC drives the high-going value of LATEN ("SHIFT INPUT CONDITIONING"); the high value of LATEN enables BUS data to flow through the '373 latches making up the Bus Latch ("ALU Data Latch"); the data flows into the B side of the '181/'182 ALU complex ("ALU BYTE (0:7) -- Figure 42"); presuming that an arithmetic function involving a carry-in propagated all the way to bit 31 of the ALU is involved, the critical path takes a circuitous route through the 181/182 carry-lookahead logic to enter the CIN bit of the highest order '181, producing bits ALU<28:31>.

At this point, the critical path splits. The actual critical path depends on the final implementation. One critical path is: bits ALU<28:31> are used to compute the signal NALU0 ("ALU ZERO DETECT"); which is then transmitted to the Control Chip and must meet setup time for the Condition Code Register '374 ("CONDITION CODE -- F.63"). This turned out to be the critical path shown in Table 1, but only by a few nanoseconds.

The second possible critical path is from ALU<28:31> through the ALU multiplexer ("MUX_SHIFTER"); and then to meet setup time for the Data High Register data inputs ("Data High Register").

The measurement for the ALU critical path may be thought of as the time between when the Data Bus becomes valid from being driven by the bus source and when the DHI register is clocked with the output of the ALU and/or the condition code register is clocked with the output of the ALU zero detection logic.

4.1.3 Data Stack

The Data Stack becomes a critical path when it is a bus source. The critical path for SOURCE=DS is: the falling edge of CLOCK which drives the Data Stack RAM chip enable line ("DATA STACK (0:31) -- SHEET 26_27_28_29"); and the data path from the Data Stack RAM through the '245 transceivers on that same sheet.

The critical path timing shown in Table 1 for the Data Stack is the time at which the Data Stack data is valid on BUS<0:31>. Special care needs to be taken when measuring stack timing with the simulator, since the simulator assumes essentially zero stack access time if the DP has not changed value since the previous cycle. The results in Table 1 were obtained by adding in the 25 ns RAM access time to simulated results.

If an asynchronous RAM were used for the Data Stack, the critical path might be controlled by the signal NSDS controlling the output enable of the stack RAM or the access time from a change in DP through the DS, instead of the stack RAM chip enable signal.

4.1.4 Return Stack

The Return Stack becomes a critical path when it is a bus source and when it is used for a subroutine exit function. The critical path for SOURCE=RS is: the falling edge of CLOCK which drives the Return Stack RAM chip enable line ("RETURN STACK (0:31)"); and the data path from the Return Stack RAM through the '245 transceivers on that same sheet.

The critical path timing shown in Table 1 for the Return Stack source is the time at which the Return Stack data is valid on BUS<0:31>. Special care needs to be taken when measuring stack timing with the simulator, since the simulator assumes essentially zero stack access time if the RP has not changed value since the previous cycle. The results in Table 1 were obtained by adding in the 25 ns RAM access time to simulated results.

If an asynchronous RAM were used for the Return Stack, the critical path might be controlled by the signal NRSOE controlling the output enable of the stack RAM or the access time from a change in RP through the RS, instead of the stack RAM chip enable signal.

4.1.5 Memory Addressing

The worst case delay path for memory addressing comes when performing a subroutine exit function. The critical path for a subroutine exit is: the falling edge of CLOCK which drives the Return Stack RAM chip enable line ("RETURN STACK (0:31)"); Return Stack access time, which is subject to the same measurement problems as the SOURCE=RS path discussed in the previous subsection; RS<0:22> flowing through the RAM Address Latch SN74373's to RAD<0:22> ("RAM ADDRESS LATCH -- F.55"); and the pad drivers to drive RAD off-chip.

The time for this path shown in Table 1 indicates the time between the falling CLOCK edge and the time RAD outputs are valid for successive subroutine return operations. This provides information about where in the clock cycle memory addressing for a two-cycle memory fetch begins.

A second situation that occurs in memory addressing is sending the contents of the data bus through the RAM Address Latch to the RAD pins. The path for this is identical to the path for the Return Stack to RAD path, except that RS<0:22> are driven from the Return Stack transceivers instead of the Return Stack RAM.

4.1.6 Data Bus to Program Memory

The Data Bus to Program Memory path is exercised when doing a DEST=RAM operation or DEST=RAM-BYTE operation. This path is: data valid on the Data Bus; data flowing from BUS to RD via the transceivers in the 32-bit structure in "byteaddr_DB" (see "RAM DATA TO BUS (8-15)"); and the pad drivers for the RD pins on the

control chip.

This time is measured between the Data Bus becoming valid and the RD pins becoming valid on the Control Chip package.

4.1.7 Program Memory to Data Bus

The Program Memory to Data Bus path is exercised when doing a SOURCE=RAM or SOURCE=RAM-BYTE operation. This path is: data valid on the RD pins of the control chip; data flowing from RD to BUS via the transceivers in the 32-bit structure in "byteaddr_DB" (see "RAM DATA TO BUS (8-15)".

This time is measured between the RD Bus becoming valid and the Data Bus being driven from the transceivers. This measurement may be a little tricky, because the source decoders may block the signals in the transceivers. This can be overcome by executing two SOURCE=RAM micro-instructions in a row using different RAM Data inputs.

4.1.8 Program Memory to MPC

The Program Memory to MPC path is exercised when doing an instruction decode. This path is: data valid on the RD pins of the control chip; data flowing from RD through the Instruction Latch 373's ("INSTRUCTION REGISTER -- F.61"); and to the inputs of the '161 MPC on the same schematic.

This time is measured between the RD Bus becoming valid and the MPC data inputs becoming valid.

4.1.9 Inter-chip Data Bus Delay

The inter-chip data bus delay is the delay introduced when transmitting a Data Bus signal from one chip to the other. This may be measured by examining the time difference between a bus signal being asserted on the internal Data Busses of each chip.

4.1.10 MRAM address valid at Data Chip

The MRAM address becomes valid for fetching the next micro-instruction after the MIR has been clocked and the conditional branch address has been generated. The path is: CLOCK goes high, causing MIR to be clocked by MIRCK ("Cmicro_ram"); MIR outputs drive the COND<0:2> inputs to the condition code multiplexer '151 ("CONDITION CODE -- F.63"); the condition code multiplexer drives MAD0; and MAD0 is transmitted from the Control Chip to the Data Chip.

This time is measured between the rising edge of CLOCK and the time MAD0 becomes valid internal to the Data Chip.

4.2 Critical Paths

Once the critical path delays within the important subsystems of the Chip implementation are identified, system-wide critical paths may be found by taking the slowest combinations of slow subsystems. While these times may be computed using the values in Table 1, the following subsections discuss how these

times may be directly measured in a simulation.

TABLE 2
Simulator Results for Maximum Operating Speed

Minimum Clock High Time	Path
26	Bus destination select at Data Chip
18	Micro-instruction address generation

Minimum Clock Low Time	Path
60	Data Stack Through ALU
67	Return Stack Through ALU
36	Return Stack to Data Stack
45	Data Stack to Program Memory (plus memory write)

Minimum time *	Path
31	Data Hi to Address Latch/ Program Memory to DLO register (plus memory read)
50	Subroutine exit/ Macro-instruction fetching (plus memory read)

* - The minimum time given for RAM operations is measured from the falling edge of the addressing clock cycle to the rising edge at the end of the following RAM read/write cycle, and is exclusive of the time required for the RAM to perform the read.

4.2.1 Data Stack Through ALU

The Data Stack through the ALU critical path is measured with a micro-instruction of the form: "SOURCE=DS ALU=A+B+1 DEST=DHI". The time delay between the rising edge of CLOCK to initiate this micro-instruction and the time when the NALU0 bit is valid at the inputs to the condition code register sets a minimum clock cycle length. The DHI register should contain a value which is non-zero in the highest four bits, and the DS register added to the DHI value should be selected to give a result of zero. An arithmetic test is important to provide for the slowest mode of ALU operation.

The minimum clock period (which reflects the maximum operating speed) for this path and the paths discussed in the subsequent sub-sections is shown in Table 2. In order to get an accurate measurement, an INC[DP] or DEC[DP] should occur on the micro-instruction before the SOURCE=DS micro-operation, otherwise the 25 ns RAM access time must be added to the simulated critical

path time.

4.2.2 Return Stack Through ALU

The Return Stack through the ALU critical path is measured with a micro-instruction of the form: "SOURCE=RS ALU=A+B+1 DEST=DHI". The time delay between the rising edge of CLOCK to initiate this micro-instruction and the time when the NALU0 bit is valid at the inputs to the condition code register sets a minimum clock cycle length. The DHI register should contain a value which is non-zero in the highest four bits, and the RS register added to the DHI value should be selected to give a result of zero. An arithmetic test is important to provide for the slowest mode of ALU operation. As with the Data Stack, the 25 ns RAM access time must be added to the critical path time if the RP did not change just before execution of the micro-instruction.

The Return Stack through ALU critical path is slower than the Data Stack critical path since the bus value must flow from the Control Chip to the Data Chip.

4.2.3 Return Stack to Data Stack

The Return Stack to Data Stack critical path is measured using a micro-instruction of the form "SOURCE=RS DEST=DS". This critical path measures the longest path (exclusive of ALU paths) involving inter-chip data transfer. The measurement is made by observing the time difference between the start of the clock cycle at the rising edge of CLOCK and the time when data is valid at the Data Stack RAM inputs (plus any set-up time if required.)

4.2.4 Data Stack to Program Memory

The Data Stack to Program Memory critical path is measured using a micro-instruction of the form "SOURCE=DS DEST=RAM". This critical path measures the longest path for writing to Program Memory, since it involves an inter-chip data transfer. This measurement is made by measuring the time difference between the start of the clock cycle and the time the Ram Data bus contains valid data from the Data Stack. For the purposes of computing required RAM chip speed, some reasonable time must be added for the data to go through buffers and reach the RAM chips used to implement Program Memory.

In addition, since memory accessing is a two-cycle operation, all reads or writes from/to Program Memory must ensure that the memory has had a sufficient addressing time. The addressing time for RAM read/writes is the time to transfer DHI to the RAD pins via the Address Latch plus the stable address time required by the RAM. This time starts at the falling edge of CLOCK during the RAM Address Latch write cycle.

4.2.5 Program Memory to Data Low Register

Since a memory read takes two clock cycles, a Program Memory

to DLO register transfer is accomplished in two micro-instructions: "SOURCE=DHI DEST=ADDRESS-LATCH" followed by "SOURCE=RAM DEST=DLO". To compute the maximum clock speed for this data path, begin with the time required for the Data High register to flow through the Address Latch (which becomes transparent on the falling edge of CLOCK), and drive the RAM Address Pins. Then add to this the time required for RAM Data inputs to reach the DLO register. This value gives a basis for determining what RAM response speed is required to support a given clock frequency.

In general, this measurement does not limit the operating speed of the Chip implementation of the CPU/32; it simply specifies the speed of memory required to make the system work at full speed. Note that the memory speed specified includes time spent on data buffers, decoders, and other devices, so actual RAM chip speed must be faster than the specified RAM response times.

4.2.6 Micro-Instruction Fetching

Micro-instruction fetching is limited by the time required to generate the address for MRAM for the next micro-instruction. In the current design, this time must be less than the clock high time. This restriction could easily be removed by stretching the MRAM chip enable high time as required.

This measurement is made by taking the time difference between the rising edge of CLOCK and the time that MAD0 becomes valid internal to the Data Chip.

4.2.7 Macro-Instruction Fetching

Macro-Instruction Fetching is a special case of Program Memory reading. In the slowest scenario, a subroutine exit is being performed, meaning that the RS must supply the value to the RAM Address Latch. On the other hand, the value read from RAM need only get to the inputs of the MPC before the rising edge of clock, so the inter-chip communication delay seen in the normal RAM reading scenario in the previous subsection is eliminated. The measurement in Table 2 indicates the sum of the addressing time and the time required to go from the RAM Data bus to the MPC.

4.3 Recommendations for Illegal Operations

Since some operations have rather slow maximum speeds but are not very useful, some micro-operation combinations should be disallowed for full-speed operation of the Chip set. This allows running the Chip implementation at the fastest useful speed for maximum throughput.

The recommended illegal micro-operation combinations are:

1) SOURCE=DS DEST=ADDRESS-LATCH (followed on the next cycle by a RAM read.)

2) SOURCE=RS DEST=ADDRESS-LATCH (followed on the next cycle by a RAM read.)

3) SOURCE=RAM DEST=xxx (where xxx is anything except DECODE or DLO.)

4.4 Performance Estimates

The minimum high clock cycle time is 26 ns, controlled by the bus destination selection logic.

Given that the illegal operations in the previous section are not used, the critical path for minimum clock low time is determined by the time to perform a SOURCE=RS ALU=xxx DEST=DHI operation, where "xxx" is an arithmetic operation and the next clock cycle uses the zero branching capability. This gives a minimum clock low time of 67 ns.

In order to meet these minimums, the fastest clock speed allowable is 10 MHZ (100 ns clock) with a 33%/67% high/low duty cycle. This of course gives no margin, but it also does not account for optimizations which will no doubt be made in the implementation before producing silicon.

Program Memory speed requirements at 10 MHz are determined by adding a clock high time to the subroutine exit time and subtracting from 200 ns (two clock periods.) This gives a total memory speed requirement of 117 ns, which might be achievable with 100 ns static memory in a well designed system with a small number of memory banks. 80 ns memory is probably more realistic for larger static memory systems. Of course, dynamic memory may be used by adding a wait-state to memory accesses.

The throughput of the system at 10 MHz is an average of 5 million stack macro-operations per second (5 MOPS), with subroutine calls, exits, and unconditional jumps for free; and combined stack primitives such as SWAP_DROP executing as a single macro-operation.

Execution speed with 120ns 32k x 8 static CMOS memory (which is common on the market) will probably be in the neighborhood of 8 MHz.

5. DESCRIPTION OF SOFTWARE AND FIRMWARE

This section amplifies the description and listings of programs supplied in the WISC CPU/32 Preliminary Documentation.

5.1 MVP-FORTH/32

The MVP-FORTH/32 kernel supplied with the CPU/32 is a 32-bit version of standard MVP-FORTH. For the purposes of this document, MVP-FORTH is the language documented in All About FORTH: An Annotated Glossary, 2nd ed., by Glen Haydon.

5.1.1 Similarities to MVP-FORTH

Wherever possible, MVP-FORTH/32 is functionally identical to MVP-FORTH with the exception that all stack elements are 32-bits wide. This means that words that expect a 16-bit integer or 16-bit address in MVP-FORTH expect a 32-bit integer or 32-bit address in MVP-FORTH/32. All MVP-FORTH words included in MVP-FORTH/32 expect the same number, order, and type of parameters as in the original MVP-FORTH. Double precision integers which take two 16-bit cells in MVP-FORTH take two 32-bit cells in MVP-FORTH/32.

For the purposes of this documentation, identical functionality means that the same input parameters produce the same output parameters on both MVP-FORTH and MVP-FORTH/32. Thus, a word such as -FIND is functionally identical, even though the dictionary structures of the two Forths are slightly different in that the LFA of the header in MVP-FORTH/32 comes before the NFA instead of after the name text.

All words that function identically in the two Forth implementations have the same names. All words that do not function identically do not have the same name. Although two words function identically, this does not necessarily mean that they are implemented identically.

5.1.2 Differences from MVP-FORTH

Most differences between MVP-FORTH and MVP-FORTH/32 have to do with words that are added or omitted. Words that are left out of MVP-FORTH/32 include some of the disk-I/O words, since the host PC performs the actual Disk-I/O. Also missing are extended PC addressing words such as C, and port addressing words such as P@. In other words, the functions that are left out are the functions that are hardware-specific to the host PC.

The additions to MVP-FORTH/32 are mostly low-level words that have little obvious use to the user. These words tend to be microcoded speed-ups of inner loops and low-level specialized utility words. Also, MVP-FORTH/32 has special words added to implement a compiler that compresses opcodes, subroutine calls/exits, and unconditional jumps as well as words to deal with interrupts and stack overflow/underflow paging.

5.1.3 LIB-FORTH as a Base for MVP-FORTH/32

All software delivered to Harris Semiconductor has been delivered for use with the LIB-FORTH version of MVP-FORTH. LIB-FORTH is a source-code compatible, speed-optimized version of MVP-FORTH for use under a PC-DOS environment. Among other features, LIB-FORTH supports an assembler, editor, and math package that reside outside the base 64k Forth dictionary, as well as DOS file support for screens files. LIB-FORTH is an unsupported public domain product that was used because it did the job required. It is not part of the supported WISC software suite.

5.2 Microcoded Functions

In general, all functions that are coded in assembly language in the MVP-FORTH kernel are coded in microcode in MVP-FORTH/32. Some functions have been modified, while others have been added. Functions in MVP-FORTH not listed here are not available in MVP-FORTH/32.

5.2.1 Functions Identical to MVP-FORTH

The following is a list of functions that operate identically to MVP-FORTH functions of the same name, with the exception of changing 32-bit to 16-bit integers:

! + +! - -1 0 0< 0= 0BRANCH 1 1+ 1- 2* 2/ <
<+LOOP> <DO> <LOOP> = > >R ?DUP @ ABS AND C! C@ D!
D+ D= D@ DDROP DDUP DNEGATE DOCON DOVAR DOVER DR> DROP
DSWAP DUP I I' J LEAVE LIT NEGATE NOT OR OVER R> R@
ROT S->D SWAP TOGGLE U* U/MOD

The following functions are documented in the MVP-FORTH Math Package, and are implemented identically to their MVP-FORTH versions, with the exception of double precision integers being a pair of 32-bit numbers instead of a pair of 16-bit numbers:

ADC ASR C+! D>R DLSL DLSR DROT LSLN LSR LSRN RLC RRC

5.2.2 Functions Modified from MVP-FORTH

No MVP-FORTH/32 microcoded primitives differ in functions from their MVP-FORTH counter-part words.

5.2.3 New Functions

Several microcode functions have been added that have no counterparts in MVP-FORTH. These words may be divided into three groups: those that are useful only as factors of high-level MVP-FORTH/32 system words, those that are combinations of normal MVP-FORTH words, and those new words that stand by themselves.

Words that are used only as factors for MVP-FORTH/32 system words are documented in the source screens. These words are:

%DP!% %DP@% %RP!% %RP@% <$=STEP> <<ABORT">> <<CM-STEP>
<CM-STEP> <COUNT-DOWN> <ENCLA> <ENCLB> <PICK> <ROLL>
<UDNORM> LOAD-DS LOAD-RS STORE-DS STORE-RS

Words that are combinations of normal MVP-FORTH words are self-documenting. They accomplish the same actions as the pair of words would if executed separately. These words are:
0=_NOT 3_PICK 4* 4_PICK @+ DUP_0< R>_DROP SWAP_!
SWAP_DROP

The final group of new microcoded words are those words which perform useful new functions. These words are:

-ROT - "Backwards" stack rotation. Equivalent to the sequence: ROT ROT.

<INTERRUPT> - Opcode 1, performs interrupt processing as described in Section 3.2.3.3.

BYTEROLL - Performs a byte rotate function of the top stack element using the ROLL[ALU] micro-operation.

HALT - Stops the CPU/32 and returns control to the host PC by placing a 1 in the status register.

NOP - Opcode 0, performs a no-operation.

RTI - Return from interrupt. Performs the actions described for returning from an interrupt in Section 3.2.3.4.

SYSCALL - Performs a "system service call" function by writing the input value into the status register. Each value from 2-255 requests a different service from the PC host.

WFILL - Word fill. Operates like the MVP-FORTH word FILL, only does the filling 32 bits at a time. Useful for initializing blocks of memory.

5.3 High Level Functions

In general, all functions that are coded in high level in the MVP-FORTH kernel are coded in high level in MVP-FORTH/32. Some functions have been modified, while others have been added. Functions in MVP-FORTH not listed here are not available in MVP-FORTH/32.

5.3.1 Functions Identical to MVP-FORTH

The following is a list of functions that operate identically to MVP-FORTH
functions of the same name, with the exception of changing 32-bit to 16-bit integers:

```
#  #>  #BUFF  #S  '  '-FIND  '?TERMINAL  'ABORT  'BLOCK  'CR
'EMIT  'EXPECT  'INTERPRET  'KEY  'LOAD  'NUMBER  'PAGE  'R/W  'S
'STREAM  'T&SCALC  'VOCABULARY  'WORD  (  *  */  */MOD  +-  +BUF
+LOOP  ,  -FIND  -TEXT  -TRAILING  .  ."  .LINE  .R  .S  .SL  .SR
.SS  /  /LOOP  /MOD  0>  2  2!  2+  2-  2@  2CONSTANT  2DROP
2DUP  2OVER  2SWAP  2VARIABLE  :  ;  <#  <-FIND>  <.">
<?TERMINAL>  <ABORT">  <ABORT>  <BLOCK>  <CMOVE  <CR>  <DOES>
<EMIT>  <EXPECT>  <FIND>  <INTERPRET>  <KEY>  <LINE>  <LOAD>
<NUMBER>  <PAGE>  <QUIT-ADDR>  <R/W>  <WHERE-ADDR>  <WORD>
>BINARY  >IN  >TYPE  ?  ?COMP  ?CSP  ?LOADING  ?PAIRS  ?STACK
?STREAM  ?TERMINAL  ABORT  ABORT"  AGAIN  ALLOT  BASE  BEGIN  BL
```

BLANK BLK BLOCK BUF-SIZE BUFFER BUMP-BUFF BYE C, C/L CFA CLEAR CMOVE COMPILE CONSTANT CONTEXT CONVERT COPY COUNT CR CREATE CSP CURRENT D+- D, D- D. D.R D0= D< D> DABS DCONSTANT DECIMAL DEFINITIONS DEPTH DIGIT DLITERAL DMAX DMIN DO DOES> DOUSE DOVOC DP DPL DU< DUMP DVARIABLE ELSE EMIT EMPTY-BUFFERS ENCLOSE EPRINT ERASE EXECUTE EXIT EXPECT FENCE FILL FIND FIRST FLD FLUSH FORGET FORTH H HERE HEX HLD HOLD ID. IF IMMEDIATE INDEX INTERPRET KEY LATEST LFA LIMIT LIST LITERAL LOAD LOOP M* M*/ M+ M/ M/MOD MAX MIN MOD NFA NUMBER OCTAL OFFSET OUT PAD PAGE PAUSE PFA PICK PP PREV QUERY QUIT R# R/W R0 REPEAT ROLL RP! RP@ S0 SAVE-BUFFERS SCR SIGN SMUDGE SP! SP0 SP@ SPACE SPACES STATE TEXT THEN THRU TIB TRAVERSE TYPE U. U.R U< UNTIL UP UPDATE USE USER VARIABLE VLIST VOC-LINK VOCABULARY WARNING WHERE WHILE WIDTH

5.3.2 Functions Modified from MVP-FORTH

COLD is the only MVP-FORTH/32 microcoded primitive that differs in function from its MVP-FORTH counter-part. COLD works almost the same as the MVP-FORTH COLD word, but differs in that it does not restore the dictionary pointer and user variables. This is to allow the user to switch between MVP-FORTH and MVP-FORTH/32 (which executes COLD every time it is restarted) for screen editing without erasing the MVP-FORTH/32 dictionary.

5.3.3 New Functions

Several high level functions have been added that have no counterparts in MVP-FORTH. These words may be divided into three groups: those that are useful only as factors of high-level MVP-FORTH/32 system words, those that are combinations of normal MVP-FORTH words, and those new words that stand by themselves.

Words that are used only as factors for MVP-FORTH/32 system words are documented in the source screens. These words are:
'ISERVICE <$MATCH> DEFAULT-JUMP DO-EXECUTE DS-ADJUST DS-AREA DS-LIMIT DS-PTR EXEC-ADDR INIT-DP INIT-RP INTERRUPT-DECODE IS-A-CALL ISERVICE OPT-STATUS RS-ADJUST RS-AREA RS-LIMIT RS-PTR

The following words are self-explanatory:
4 4+ 4-

The final group of new words are those words which perform useful new functions. These words are:

CALL, - Compiles a subroutine call. Used instead of the normal "," to allow the optimizing compiler to work properly. The input is the program field address of the subroutine.

COUNT-DOWN - This is a new kind of count-down loop. Used as a BEGIN ... COUNT-DOWN structure, this word decrements the top of the data stack and branches back to BEGIN if the value is not -1. If the value is -1, the loop is terminated and the count

value is dropped from the data stack. COUNT-DOWN only takes three clock cycles to loop, four to fall through.

DON'T-DISTURB - Signals the optimizing compiler not to attempt further optimization on the just-compiled dictionary cell.

DS-SIZE - Constant specifying the number of bytes in the Data Stack overflow save area.

EXIT, - Compiles a subroutine exit. Used to allow the optimizing compiler to work properly.

INTERRUPT-SERV - Interrupt service routine. This routine pages the Data Stack and Return Stack into or out of Program Memory if a stack overflow or underflow occurs. Other interrupts cause program termination with a message describing the type of interrupt.

OPCODE, - Compiles an opcode. The input is the opcode desired (placed in the highest 9 bits of a 32-bit word.) This is used instead of the normal "," to allow the optimizing compiler to work properly.

OPTIMIZE? - Optimization flag. This variable, when non-zero, signals the compiler to perform opcode/subroutine call/unconditional branch/subroutine exit compression.

POISON - A word that sets a header bit, much like IMMEDIATE. This header bit, when set, signals INTERPRET to abort if the word is executed in interactive mode. This prevents crashes caused by trying to execute dangerous words like 0BRANCH, >R, and LIT from the keyboard.

RS-SIZE - Constant specifying the number of bytes in the Return Stack overflow save area.

SPECIAL - A word that sets a header bit, much like IMMEDIATE. This header bit, when set, signals the optimizing compiler that the word in question is to be compiled as a stand-alone instruction. This prevents opcodes like R> from being combined with subroutine calls, which would lead to improper program execution.

WDUMP - This works just like DUMP, except memory is displayed in 32-bit chunks. This is very useful for examining compiled high level word definitions in Program Memory.

5.4 Additional High Level Functions

Some additional high level Forth functions have been added to MVP-FORTH and/or MVP-FORTH/32. These functions are added for convenience, and are not to be considered fully supported at this time. These functions should be optimized and fully supported in a final release off the software for general distribution.

5.4.1 Math Package

The MVP-FORTH Math Package as documented in Volume 3 of the MVP-FORTH Series: Forth Floating Point and Extended Precision Integer Math, by Phil Koopman, is partially supported by MVP-FORTH/32. Supported words function identically to the words

described in the book, except floating point numbers fit in a single 32-bit stack element, temporary floating point numbers fit in two 32-bit stack elements, and double precision integers are two 32-bit stack elements. Most omissions involve lack of support for quad precision integers, since 128-bit integers are bigger than most users need.

The included math support words are:

2 1/X 10 2** 2*PI <?MODE> <ACOS> <ACOT> <ACSC> <ASEC> <ASIN> <ATAN2> <ATAN> <COS> <COT> <CSC> <F.> <FINTERPRET> <FNUMBER> <P->R> <R->P> <SEC> <SIN> <TAN> <TNUMBER> ?MODE ACOS ACOT ACSC ASEC ASIN ASRN ATAN ATAN2 CHK0 COS COT CSC D* D*/ D*/MOD D+! D->S D/ D/MOD D0< D0> D? DADC DAND DASR DASRN DEG->RAD DINP# DLSLN DLSRN DM* DM/ DM/MOD DMOD DMODE DOR DPICK DR@ DRLC DROLL DRRC DU* DU/MOD DU> DXOR E** EXP F* F** F+ F+! F+- F- F->ME F->N F->T F. F.A F.AR F.E F.ER F.R F.X F.XR F/ F0< F0= F0> F2* F2/ F< F= F> F? FABS FACTORIAL FCONVERT FINP# FMAX FMIN FMODE FNEGATE FRAC FSGN FTERM INT LN LOG LOG2 LOGB LSL N->F N->T P->R PI PI/2 PI/4 Q! Q+ Q+! Q+- Q- Q>R Q@ QABS QADC QAND QASR QDROP QDUP QLSL QLSR QNEGATE QOR QOVER QR> QR@ QROT QSWAP QXOR R->P RAD->DEG REM ROOT SEC SEPARATE2 SGN SIGDIG SIN SQRT T* T+ T+! T+- T- T->F T->N T. T/ T0= T2** TABS TAN TATAN TCONVERT TCOS TEMP-ADDR TEMP-CARRY TERM TFRAC TINP# TLOG2 TLOGB

5.4.2 Screen Editor

A full-screen editor is included with LIB-FORTH. This is an unsupported public domain screen editor. Documentation is included in the LIB-FORTH package.

5.4.3 DOS File Interface

A DOS file interface is included with LIB-FORTH. This is an unsupported public domain file interface for PC-DOS. Documentation is included in the LIB-FORTH package.

6. TEST VECTORS

Three sets of test vectors were supplied with the schematics for the Chip implementation of the CPU/32. Each set tests the system under different conditions. Together, the sets test all possible data paths and control conditions for the CPU/32.

6.1 Main Slave Mode Test Vectors: HARRIS.BIN

Almost all the control signals and data paths can be tested using single-stepped microcode in Slave Mode. The file HARRIS.BIN contains information to perform approximately 2800 single-stepped micro-instructions to perform testing of the entire CPU/32 system. These tests are a more thorough adaptation of the single-step tests used to test the Board implementation.

6.1.1 Test Vector Generating Program

Since the desired single step tests closely resembled the normal tests performed on the Board implementation with existing software, and since the hand calculation of 2800 test vectors is rather tedious, the microcode assembler was modified to incorporate a special simulating and test vector generating functions. The input to the simulator program is standard single-stepped microcode on Forth screens. The output of the simulator is the HARRIS.BIN test vector text file, as well as a hexadecimal version of the test vectors in the file HARRIS.HEX.

The test vector generating/simulating program runs on a PC-AT with an installed CPU/32 Board set. It uses a combination of actual CPU/32 hardware and a functional simulation of the memory address logic to create test vectors.

6.1.2 Clock Cycling Information

The clock cycle for each test vector in HARRIS.BIN consists of two phases. The first phase of the clock cycle loads a micro-instruction into the MIR by pulsing the NDMIR signal low. The second phase of the clock cycle cycles CLOCK by pulsing the NCYCL signal low.

The outputs of the system are sampled just before the rising edge of the NCYCL signal.

Because of the limits of the SDA simulation software, a rigid clock cycle format must be maintained. Therefore, the MIR is loaded prior to each cycling of CLOCK, whether or not the MIR value needs to be changed. This is a very slight limitation in practice, since the CYCLE micro-assembler directive has been modified to re-load the MIR with the same value remaining from the previous clock cycle.

6.1.3 Test Vector Formats

Each of the approximately 2800 test vectors actually consists of two sets of input vectors (one for the MIR load and one for the CLOCK cycle) and one output vector.

The format for the MIR input vector and the CLOCK cycle input vector is: DVOSC, NDMA, NDSRV, NMAST, NPRTY, NSMIR, BUS<0:31>, RD<0:31>.

The format for the output vector is: NDRB, NDRW, NSINT, NRAM, NSFLG, NSPC, BUS<0:31>, RAD<0:22>, RD<0:31>.

6.2 Miscellaneous Slave Mode Test Vectors: CYCLE.BIN

A few tests can not be performed in single step mode. These tests, incorporated into CYCLE.BIN, are an MIR read-back test, a DMA transfer test, and an MRAM to MIR transfer test. The MIR read-back test actually can be accomplished in single-step mode, but not under the constraints of the clock cycle used by HARRIS.BIN

6.2.1 Clock Cycling Information

No regularly oscillating clocks are used by CYCLE.BIN. Instead, a separate control input file provides values at regular intervals to change input clock signals. These input control signals are sampled twice as fast as the input data. The output data is sampled on every second input control signal.

6.2.2 Test Vector Formats

The format for the control input file is: DVOSC, NCYCL, NDMA, NDMIR, NDSRV, NMAST, NPRTY, NSMIR.

The format for the data input file is: BUS<0:31>, RD<0:31>.

The format for the output vector is: NDRB, NDRW, NSINT, NRAM, NSFLG, NSPC, BUS<0:31>, RAD<0:22>, RD<0:31>.

6.3 Master Mode Test Vectors: RUN.BIN

While a Master Mode test run is not strictly required to prove chip functionality, a full-speed Master Mode test was added to build confidence and to provide a convenient vehicle for measuring critical path delays.

6.2.1 Microcode Memory Set-Up

RUN.BIN is a two-part test. The first part, which takes up the bulk of the test vectors, loads the MRAM with micro-instructions to be executed during the full-speed run portion. These micro-instructions are documented in the comments of the test file, and exercise all the important data paths of the chip as well as subroutine call and return logic.

6.2.2 Master Mode Clock Cycling Information

The second part of RUN.BIN is a full-speed program execution test. Since no software tools are appropriate to help build the information used in this test, the test was created to be a compact, hand-assembled program. This section of RUN.BIN is a maximum clock speed test. If the clock is cycled too quickly, indeterminate values will show up on the system bus, and/or the micro-instructions will be executed in improper order due to

improper branch on zero conditions.

A problem with RUN.BIN is that it was not designed to account for the stack RAM's providing almost instantaneous response when the stack pointer has not changed from the previous cycle. This problem means that RUN.BIN gives a maximum speed that is too optimistic. However, the critical path analysis done using the simulation results from RUN.BIN in Section 4 have been corrected for this, and are correct speed estimates. For testing production line chips, a more extensive at-speed test should be devised and captured with a logic analyzer from the pins of first pass Chip hardware.

6.2.3 Test Vector Formats

The input and output formats as well as the sampling strategy for RUN.BIN are identical to the ones used for CYCLE.BIN.

The format for the control input file is: DVOSC, NCYCL, NDMA, NDMIR, NDSRV, NMAST, NPRTY, NSMIR.

The format for the data input file is: BUS<0:31>, RD<0:31>.

The format for the output vector is: NDRB, NDRW, NSINT, NRAM, NSFLG, NSPC, BUS<0:31>, RAD<0:22>, RD<0:31>.

7. PROTOTYPE TEST BOARD

When the Chip implementation is fabricated, a prototype test board will be needed. This section provides a schematic design for an IBM AT compatible plug-in board for this purpose.

7.1 Purpose and Limitations

The prototype test board design discussed in this section is designed to be a bare minimum testing platform for fabricated chips. The design is mostly an extraction of appropriate portions of the Board implementation, and is only meant to guarantee that Harris has enough information to construct a testing platform. The prototype test board design uses only two banks of static memory and does not attempt to solve the PC address noise problem (which is not a problem on the PC-AT.)

The schematics for the prototype test board are given in Appendix E. The actual chips used for the Chip implementation do not appear on the schematics, since the pin number assignments have not been established. The pins on the CPU/32 chips should simply be connected to all signals with matching mnemonics.

It is contemplated that WISC Technologies will design and manufacture a more powerful test platform in cooperation with Harris for use with the prototype chip run.

7.2 Possible Expanded Versions

Possible enhancements which may appear on a future prototype board design include: a mix of fast static and slow dynamic memory, a cure for the PC address bus noise problem, interrupt logic to interrupt the PC Host for service requests, and semi-custom or programmable hardware to eliminate some SSI chips.

8. RECOMMENDATIONS FOR FUTURE ENHANCEMENTS

This section contains various recommendations for enhancements that should be considered for current and future generations of the CPU/32 Chip implementation.

8.1 Return and Data Stack Memory

As discussed in Section 9, the use of synchronous RAM for the Return Stack and Data Stack exacts a significant performance penalty. There are three steps that may be taken to reduce this impact: make a slight change to subroutine exit operation, provide asynchronous RAM, and change the timing of the stack accessing.

8.1.1 Change to Subroutine Exit Operation

IF the RAM compiler actually produces memory that has outputs valid essentially immediately after chip-enable goes low (when the stack pointer value has not changed from the previous cycle,) then the Program Memory speed requirements for macro-instruction fetching can be eased by 19 ns.

The change required is in schematic "RETURN STACK CONTROL LOGIC -- F.60". In this schematic, the NDEC<2> signal should be disconnected from the NOR gate instance 12 pin 12, but left connected to OR gate instance 6 pin 12. Then, the signal NDEC<1> should be connected to NOR gate instance 12 pin 12. This change increments the Return Pointer at the end of the DEC1 cycle instead of the DEC2 cycle, meaning that successive subroutine exits will find the Return Stack valid for a clock cycle before using it during the DEC1 cycle or a subroutine return. This, in connection with modest microcode changes to ensure that the Return Pointer is not changed during DECODE micro-instructions, will eliminate subroutine exits as a critical path for Program Memory access. The Program Memory critical path will then become the data read path, which is 19 ns shorter.

This change is highly recommended for immediate implementation.

8.1.2 Asynchronous RAM

Providing asynchronous RAM will speed up all Return Stack and Data Stack operations. Since the stack pointers provide valid stack addresses approximately 5-10 ns after the rising edge of clock, asynchronous RAMs will substantially reduce or eliminate the 27 ns period after the falling edge of CLOCK until the stack contents become valid on the system Data Bus. This will reduce the critical path for the RS through ALU case and DS through ALU case, increasing the potential operating speed of the Chip implementation. An operating speed of approximately 13 MHz should be possible by making this change independent of other optimizations.

Implementing this change will involve eliminating the chip

enable signal to the stack RAMs.

This change is highly recommended for implementation if and when a suitably fast asynchronous RAM design is available.

8.1.3 Change to Stack Accessing Timing

A longer-range solution to the stack timing problem using synchronous memory is to change the chip enable high time to be the third of the clock cycle before the high portion of CLOCK. This would entail generating a different clock phase that is high during the last third of the clock cycle. This clock phase would be used as the clock input to RS, RP, DS, and DP.

With this scheme, stack pointers would be loaded, incremented, or decremented at the rising edge of the new clock phase. The stack RAM chip enables would also be driven by the new clock phase. This would mean that a valid stack address would be available at the rising edge of CLOCK, and the RAM access time would also start at the rising edge of CLOCK. In this manner, stack RAM values would be available to the data bus and/or the RAD outputs one-third of a clock cycle earlier than with the current design.

In order to make this scheme work, transparent latches for RP and DP values as well as RS and DS values are needed to hold the "old" values during the last third of the clock cycle when these resources are sourced to the system Data Bus.

This change requires than any source writing to RS, DS, RP, or DP must present valid data on the Data Bus before the rising edge of the new clock phase. In practice this will probably not be a limitation except on the RAM to DS and RAM to RS paths, which are too slow to be useful even in the current implementation. This change will speed the system up even more if used in conjunction with asynchronous RAM. There are no required microcode changes. The benefits of making this changer are the capability for a substantially faster clock cycle (especially if asynchronous RAM is used with this scheme) and slower Program Memory RAM response time requirements for any given clock speed than for the current design.

The only drawback to this change is that it would increase the risk on the first-pass silicon by introducing operating mechanisms different from the Board implementation. It will also take a fair amount of engineering time and simulation time to implement and verify. For these reasons, this change is recommended for incorporation into the second version of the Chip implementation.

8.1.4 Elimination of Data Stack Transceivers

A minor change that will help the DS through ALU critical path by a few nanoseconds is the elimination of the SN74245's on the schematic "DATA STACK (0:31) - SHEET 26_27_28_29". This change is feasible if the Data Stack RAM tri-state outputs are powerful enough to drive the system Data Bus directly. Of

course, this change should not be implemented if the extra load on the Data Bus slows the bus down more than the elimination of the '245 components speeds it up.

Since this change will not significantly affect chip operating speed, this change this change is recommended for the second-pass implementation.

8.1.5 Stack Size Issues

The current stack size of 512 elements for both the Return Stack and Data Stack is somewhat arbitrary, but is a good size for the first version of the Chip implementation. Actual run-time analysis in a variety of operational environments is highly desirable before changing these values for a second-pass design. The following paragraphs contain some subjective observations about appropriate stack sizes that should be considered when considering stack size changes.

For most standard Forth programs, 512 elements is extremely large, with probably 128 to 256 elements being quite sufficient. In most Forth applications, the Data Stack tends to grow faster than the Return Stack, since most Forth words take one or more input parameters while requiring only a single return address value. Therefore, for standard Forth applications 128 elements on the Return Stack and 256 elements on the Data Stack should be quite sufficient.

For deeply recursive applications, such as expert systems in any language, or implementations of Prolog or LISP, deep stacks are very important. How deep the stack should be depends on the dynamics of stack usage, but 512 elements is probably a usable minimum size for both the Data Stack and Return Stack. The appropriate stack size in a highly recursive environment can be experimentally found by looking for the smallest stack size that provides a minimum amount of thrashing to and from the stack overflow image stored in Program Memory. The optimal stack size may be rather application dependent, and will definitely be extremely sensitive to the aggressiveness of the compiler writer in using the hardware stacks versus software stacks residing in main memory.

For conventional languages such as C, the optimum depth of the stack will depend on how successfully dynamically activated subroutine parameter lists can be maintained on the hardware stacks. If most subroutine parameters can be held in hardware stacks, then deep stacks are desirable. If the mechanics of maintaining parameters on hardware stacks are so inefficient that parameter lists are instead maintained in Program Memory, then small stacks of the size required by normal Forth applications are sufficient.

Obviously there is a trade-off to be made between stack size, execution speed, and chip area available for microcode memory or other functions.

8.2 Bus Multiplexing

The current Chip implementation uses several tri-state busses. Since multiplexers are more appropriate for a CMOS semi-custom approach, the following small buses should be converted to multiplexed data paths as soon as possible: RS bus, RAD bus, RD bus.

The system Data Bus presents a harder problem. Since it has so many sources and destinations, it presents a speed problem. At the same time, however, the Data Bus would require truly massive multiplexers to become a multiplexed selector instead of a tri-state bus. A partial solution is suggested: divide the bus resources into time-critical and non-time-critical groups. Then connect the time-critical groups to a central data bus while grouping and buffering non-time-critical groups to reduce the load on the central data bus.

The only time-critical data paths for bus destinations are: DEST=ADDRESS-LATCH and the input to the B side of the ALU through the Data Latch.

The non-time-critical paths for bus sources are: SOURCE=HOST (signalled by NSPC), SOURCE=DP, SOURCE=FLAGS, SOURCE=RP, SOURCE=ADDRESS-COUNTER, SOURCE=MPC, SOURCE=MRAM, and the MIR source signalled by NSMIR.

Optimizations to improve the speed of the data bus are recommended for the first chip version if time and resources allow. They are highly recommended for inclusion in the second chip version.

8.3 Microcode Memory

Microcode Memory takes up more than half of the Chip implementation. Therefore efficient usage and size requirement estimation is imperative for cost-effective production of CPU/32 chips.

8.3.1 Microcode Memory Size

Microcode Memory is currently 30 bits wide. The two unused bits are not included in the RAM storage cells. If a direct control bit were required for some reason future versions of the chip, MRAM width could be increased to 31 or 32 bits to provide one or two additional control signals with minimal hardware changes. Of course, this technique would increase instead of decrease the silicon area used by MRAM.

The number of words in Microcode Memory is an issue worthy of study. 2048 words giving up to 256 opcodes is certainly enough for Forth or any other single language environment. Probably a reasonably efficient second language implementation could co-inhabit the 2048 words, but there would be little if any room left for application-specific microcode.

A fairly effective way of squeezing more service out of a given amount of Microcode Memory is to scatter long, non-looping microcode definitions such as U* and U/MOD in the unused micro-

instructions of opcode pages containing short macro-instruction implementations. For example, the last six micro-instructions of U* could occupy offsets 2 through 7 in the same page as the DUP opcode, since DUP only uses offsets 0 and 1. This technique could give an estimated 10% to 20% compaction of microcode.

Another useful technique (if efficiently supported by the RAM compiler) would be to populate the first 64 to 128 opcodes with only two micro-instruction RAM words per page, the second 64 to 128 opcodes with only four micro-instruction RAM words per page, and the remaining MRAM locations with the full eight micro-instruction RAM words per page. This scheme takes advantage of the fact that approximately 25% of opcodes written to date require only two micro-instruction words and 25% of opcodes written require three or four micro-instruction RAM words.

8.3.2 RAM Vs. ROM

A more straightforward way to reduce MRAM space requirements is to ROM part of the MRAM contents. Many opcodes such as DUP, SWAP, and U* will never change, and are applicable to many language environments. These words can and should be ROM'ed in the second or subsequent chip versions.

In some dedicated application versions of the chip, the entire instruction set can be ROM'ed after development with a RAM-based version of the chip. Except for dedicated applications, however, it is important that some amount of RAM-based Microcode be left on-chip to accommodate software changes and allow for users to obtain significant speed increases by microcoding frequently used subroutines. The minimum amount of RAM that should always be left on chip is subject to debate, but probably should exceed 64 opcodes (512 words).

8.4 A Stand-alone Processor/Single Chip Version

A potentially important version of the Chip implementation is a stand-alone processor version. This version of the Chip implementation will need to either have control circuitry on-chip to perform initialization and microcode loading from non-volatile memory at power-up time, or will need to have EEPROM or other non-volatile programmable microcode memory. A modest amount of support circuitry should allow booting up from a ROM-based program memory in a stand-alone mode.

If a stand-alone single-chip processors is created, a significant number of pins may be saved by eliminating the 32-bit system Data Bus from the pinout. This will necessitate performing memory-mapped I/O to communicate with the outside world, or will require the addition of serial communications control circuitry and a serial I/O pin to the chip.

An important consideration for the second-pass version of the design is the availability of a full 32-bit RAM Address Bus. Such a bus should allow un-paged linear data memory accessing over 4 gigabytes, and paged program memory addressing in 8 or 16

megabyte page increments. This can be accomplished by bringing all 32 bits of the RAM Address bus out onto pina, and by expanding the RAM Address Latch to a full 32-bit width (and therefore using the Page Register only in conjunction with the Next Address Register.)

Another possible enhancement is the use of the currently unused micro-condition codes 5 and 6 to test external status pins or internal data bits. A particularly interesting possibility is using a condition code to test the highest bit of the DLO register for 32-bit floating point normalization.

8.5 Uniform Software Environment for FORCE/WISC

The software environment for the CPU/32 is in a well-tested but preliminary state. While the current software could be polished for use with the final product, this is probably not the best way to proceed.

Harris Semiconductor, as the potential industry leader for Forth-related business, should make an attempt to standardize its own Forth-related products. In particular, the language and support tools for FORCE should have the same "feel" as the language and support tools for the WISC product when executing a Forth environment. This will reduce the burden on Harris employees by controlling the proliferation of multiple software environments for support of the chips, and will give the impression of having a family of stack-oriented processors - both 16-bit and 32-bit.

This does not mean that Harris should necessarily develop identical compilers for both products on its own. What it does mean is that Harris should make every attempt to encourage software developers to make available consistent software for both systems. This means that screen editors for both systems should use the same keystrokes to accomplish similar actions, the supported instruction sets should both make the same assumptions about things like how division and PICK work (Forth-79 vs. Forth-83), and at least one dialect of Forth should be available in highly consistent form on both machines. Harris and WISC should work together to create a consistent software environment on both machines.

As an observation: whether Harris Semiconductor wants to become involved or not, and regardless of any Forth "standard" efforts, the software tools used on the FORCE and WISC chips will probably play a large role in setting the de facto standard for Forth software in the coming years. This will happen because engineers out in the "real world" who are not steeped in the politics and traditions of the Forth community will use whatever software tools are most convenient to get their systems employing the FORCE and WISC chips to operate. Harris should take advantage of this opportunity to make life easy for software suppliers and its own support engineers by encouraging the development of a consistent, useful software environment.

9. PROBLEMS ENCOUNTERED IN THE DESIGN PROCESS

As may be expected, several problems were encountered in transferring the CPU/32 design from a discrete implementation to a semi-custom chip implementation. The following sections discuss only the major problems that had or could have had a large impact on the effort.

9.1 Synchronous Stack Memory

The most severe problem in transferring the technology to the Chip implementation is the lack of an asynchronous RAM compiler for stack memory. The use of synchronous RAM instead of the asynchronous RAM used in the Board implementation entailed making several design changes to generate a chip enable signal for the synchronous RAM macros.

The worst problem with using synchronous RAM is that it lengthens the critical paths of the Chip implementation by up to the access time of the RAM array (25 ns.) This happens because in the Board implementation, stack memory addresses are valid a few gate delays after the rising edge of CLOCK, so the asynchronous RAM begins its access period near the beginning of the high CLOCK period. Depending on the operating frequency and RAM speed, the stack data output is available for reading at or shortly after the falling edge of CLOCK.

With synchronous RAM, the high CLOCK period must be used as the chip enable signal, so the access delay does not begin until after the falling edge of CLOCK, essentially adding the entire access delay into the critical path. As seen in earlier sections, this delay affects both the data flow critical path (RS and DS through the ALU) and the program memory access critical path (RS to RAD for subroutine exits.) While some work-arounds are suggested in section 8.1.1, the availability of an asynchronous RAM compiler would have reduced the design changes necessary (and therefore reduced risk,) and would have produced a faster component for the first pass.

9.2 Untested Library Macros

During the simulation of the Chip implementation, it was discovered that the SN74181 macro-cells did not work at all. This was not a case of a single missed test vector when the macro-cells were developed; they had clearly never been tested at all. Furthermore, when a corrected version was entered into the system, there was still an error that was only caught by an added last-minute set of test vectors.

Requests for a 100% coverage test vector set for the '181 macro were never satisfied. While it is likely that the test vectors supplied in the file HARRIS.BIN prove correct functionality for the macros used for the ALU, this can not be guaranteed without a '181 test vector set to incorporate into HARRIS.BIN. It is possible that some combination of input data

and functions will not produce correct results.

It is even more possible, and perhaps probable, that the test vectors in HARRIS.BIN will not find some manufacturing defects that may occur in the ALU area, since the current tests are aimed at proving functionality, not aimed at testing gates.

The issues to be resolved are: how can Harris be sure that there are no latent bugs in other macros in this and other designs that weren't caught by the CPU/32 test vectors. Also, how can Harris be sure that fabricated silicon has no defects, especially in the ALU's, that will only be uncovered when the user runs real data in real programs? While total 100% test coverage of the chip may not be practical (although with the CPU/32 design, it should be relatively easy), starting with one or more unproven macros and progressing to a test environment where fault coverage is computed after the fact (as opposed to set as a goal at design time) is a scary prospect.

9.3 Simulator Failing to Produce Complete Output Lists

A problem that was uncovered only after the end of the summer consulting period was the fact that the SDA software was producing an incomplete test vector output file without any noticeable warnings. This situation was apparently due to an input command that did not allocate enough disk space for some step of the simulation process. This problem was further compounded by the fact that the SILOS-CHECK automatic test vector output checking program did not flag the fact that there were fewer output vectors in the simulation run file than were expected.

These problems together led to a situation where the simulation appeared to be working correctly, but actually had problems in the second half of the output vector file. Fortunately, this problem was caught and is now being corrected by manually checking to ensure all output vectors are in the "store.out" file. Potentially, this could have lead to a non-functional chip.

The recommended corrective action is to get the CAD tools fixed so that they produce error messages on the screen when running out of disk space or when the test vector simulation file is shorter than the expected result file.

10. CONCLUSIONS

The effort to convert the discrete Board implementation of the CPU/32 to a semi-custom Chip implementation appears to be a success. The combination of hard work by Harris employees and a set of CAD tools for the implementation effort which behaved reasonably well enabled the project to go "from 0 to 32 bits in 31 days."

A major limitation of system speed is the unavailability of asynchronous RAM for use in the stacks. Even so, successful simulation runs indicate that the Chip set will function correctly at approximately 10 MHz.

Once the first set of chips is fabricated, studies should be made to determine optimum stack memory and microcode memory sizes and implementations for future versions of the chip. The development of a self-booting stand-alone version of the Chip with a partially ROM'ed microcoded instruction set is a worthy goal for the second implementation cycle.

APPENDIX A. SIGNAL DESCRIPTIONS FOR LUMPED SYSTEM

The Lumped System is a conceptual package containing both the Data and Control chips as a single entity. This would be the result of mounting both chips in the same package or integrating the logic from both chips into a single piece of silicon. For the two-chip implementation, the Lumped System is the chip set as it appears to the outside world.

The Lumped System has 101 pins plus power and ground.

INPUTS:

DVOSC	Divided oscillator input. 1 in slave mode
NCYCL	NOT-Cycle clock (single step in slave mode)
NDMA	NOT-DMA transfer mode
NDMIR	NOT-Dest MIR
NDSRV	NOT-Dest service request register
NMAST	NOT-Master mode
NPRTY	NOT-Parity error input
NSMIR	NOT-Source MIR

BIDIRECTIONALS:

BUS0:31	System data bus
RD0:31	RAM Data bus

OUTPUTS:

NDRB	NOT-Data bus dest is ram(byte)
NDRW	NOT-Data bus dest is ram(word)
NSINT	NOT-Interrupt to host (status reg has changed)
NRAM	NOT-Enable RAM to/from RD bus
NSFLG	NOT-Data bus source is flag register
NSPC	NOT-Data bus source is PC interface
RAD0:22	Ram Address 0-8Mbytes

APPENDIX B. SIGNAL DESCRIPTIONS FOR DATA CHIP

The Data Chip has 66 pins plus power and ground. All pins are connected to pins of the same name on the Control Chip, and do not go elsewhere (except the data BUS, which is also connected to the external host interface.)

INPUTS:

DVOSC	Divided oscillator input. 1 in slave mode
MAD0:10	Microcode memory address
NCYCL	NOT-Cycle clock (single step in slave mode)
NMAST	NOT-Master mode
NMRCE	NOT-MRAM chip enable
NMROE	NOT-MRAM output enable
MRXDR	MRAM xceiver direction control
NDDP	NOT-Data bus dest is DP
NDDS	NOT-Data bus dest is DS
NDIV	NOT-Division select
NDMIR	NOT-Dest MIR
NMRXE	NOT-MRAM xceiver enable
NMULT	NOT-Multiplication select
NSDHI	NOT-Data bus source is DHI
NSDLO	NOT-Data bus source is DLO
NSDP	NOT-Data bus source is DP
NSDS	NOT-Data bus source is DS
NSMIR	Not-Data bus source is MIR
NWMRA	NOT-Write MRAM

BIDIRECTIONALS:

BUS0:31	System data bus

OUTPUTS:

ALU31	Sign bit of ALU output
DLOLO	Lowest bit of DLO register
NACOT	NOT-ALU carry-out bit
NALU0	NOT-ALU output equal to 0 condition bit
NDPER	NOT-DP error (underflow/overflow)

APPENDIX C. SIGNAL DESCRIPTIONS FOR CONTROL CHIP

The Control Chip has 130 pins plus power and ground. The control chip may be thought of as the "main" chip, since it controls all interfacing with the outside world.

INPUTS:

ALU31	Sign bit of ALU output
DLOLO	Lowest bit of DLO register
DVOSC	Divided oscillator input. 1 in master mode
NACOT	NOT-ALU carry-out bit
NALU0	NOT-ALU output=0 bit
NCYCL	NOT-Cycle clock (single step in slave mode)
NDMA	NOT-DMA transfer mode
NDMIR	NOT-Dest MIR
NDPER	NOT-DP error (underflow/overflow)
NDSRV	NOT-Dest service request register
NMAST	NOT-Master mode
NPRTY	NOT-Parity error input
NSMIR	NOT-Source MIR

BIDIRECTIONALS:

BUS0:31	System data bus
RD0:31	RAM Data bus

OUTPUTS:

MAD0:10	Microcode memory address
MRXDR	MRAM xceiver direction control
NDDP	NOT-Data bus dest is DP
NDDS	NOT-Data bus dest is DS
NDIV	NOT-Division select
NDRB	NOT-Data bus dest is ram(byte)
NDRW	NOT-Data bus dest is ram(word)
NMRCE	NOT-MRAM chip enable
NMROE	NOT-MRAM output enable
NMRXE	NOT-MRAM xceiver enable
NMULT	NOT-Multiplication select
NRAM	NOT-Enable RAM to/from RD bus
NSDHI	NOT-Data bus source is DHI
NSDLO	NOT-Data bus source is DLO
NSDP	NOT-Data bus source is DP
NSDS	NOT-Data bus source is DS
NSFLG	NOT-Data bus source is flag register
NSINT	NOT-Interrupt to host (status reg has changed)
NSPC	NOT-Data bus source is PC interface
NWMRA	NOT-Write MRAM
RAD0:22	Ram Address 0-8Mbytes

APPENDIX D. CHANGES TO WISC CPU/32 DOCUMENTATION

The following pages are important changes to the WISC CPU/32 Preliminary Documentation. Each page should be directly substituted for the existing page in the document. These changes reflect hardware engineering changes made to the discrete Board implementation, and therefore to the functionality of the Chip.

The DEC0 cycle, initiated by the DECODE micro-operation, must always occur in the next-to-last micro-instruction executed within a microcoded word. During the DEC0 cycle, the interrupt flag registers are examined for pending interrupts, and the MPC is clocked with the value of the Instruction Latch (IL). If a non-masked interrupt is pending, the MPC value is forced to 1 instead of the IL value, causing an interrupt service word to start execution.

The DEC1 cycle, denoted by the END micro-assembler word, must always occur in the last micro-instruction executed within a microcoded word. During the DEC1 cycle, the ADDRESS-COUNTER is incremented to form a subroutine return address pointing to the next sequential word. If a subroutine call or unconditional branch is specified by the instruction in the IL, the NAR outputs are enabled to drive the RAM address bus. If a subroutine call is being processed, then an INC[RP] is automatically performed. If a subroutine exit is being processed, then the ADDRESS-LATCH is loaded from the RS, and the ADDRESS-LATCH outputs are used to drive the RAM address bus. The END micro-assembler word forces a JMP=000 micro-operation to ensure that the 0th offset micro-instruction is the first micro-instruction executed by the next opcode.

The DEC2 cycle occurs during execution of the first micro-instruction of the word that was held in the IL during the DEC0 cycle. During the DEC2 cycle, if an interrupt is not being processed, the ADDRESS-COUNTER is loaded from whatever value is present on the RAM address bus, and the IL and NAR are loaded with whatever value is on the program RAM data bus. All these loads occur at the end of the clock cycle. Additionally, if an unconditional branch is being processed, the NAR is used to drive the RAM address bus. If a subroutine call is being processed, the NAR is used to drive the RAM address bus and the RS is written with the contents of the ADDRESS-COUNTER. If a subroutine exit is being processed, then the contents of the ADDRESS-LATCH register are used to drive the RAM address bus and the RP is incremented.

Each microcoded instruction must be at least two clock cycles long. Since the IL is a transparent latch that contains valid data before the end of a clock cycle, the opcode may be read from RAM during the DEC2 cycle and clocked into the MPC in the same clock cycle if desired. This means that the DEC2 cycle of one instruction may occur simultaneously with the DEC0 cycle of the next instruction.

In the instruction decoding process, the <u>next</u> instruction to be executed is being loaded into NAR and IL as the first micro-instruction of the <u>current</u> instruction is being executed. Use of

the micro-operation DEST=DECODE allows changing the contents of the NAR, IL, and subroutine control bits during the middle of a microcoded word, just as if those registers had been loaded with the normal decoding sequence. The MVP-FORTH/32 word OBRANCH and other words exploit this fact.

There are some micro-operations that, while prohibited under most circumstances, are essential for efficient program execution for special cases. Be very careful when exploiting these special cases, and do not rely on a single test to ensure correct operations, since some violations of usage rules manifest themselves as relatively infrequent random failures:

1) SOURCE=ADDRESS-COUNTER may be used during the DEC2 cycle to fetch the contents about to be saved on the RS if the instruction being executed is guaranteed to be a subroutine call. This is accomplished by having a special compiling word for the microcoded word in the Forth kernel.

2) SOURCE=RS, DEST=RS, SOURCE=RP, DEST=RP may all be used during the DEC2 cycle if the instruction being executed is guaranteed to be an unconditional jump. This is accomplished by denoting the microcoded word as SPECIAL with the micro-assembler.

Micro-Instruction Processing

Each CPU/32 opcode is implemented as a series of two or more micro-instructions. Each opcode starts on the MRAM page number corresponding to the opcode number 0-511. Each opcode may take one to sixteen consecutive MRAM pages as required.

Within each page, the order of the micro-instructions is unimportant, except that the first micro-instruction on the opcode's first page must be located at offset 0. The micro-assembler assumes that instructions are to be processed in sequential order (i.e. 0,1,2,3,4,5,6,7) unless instructed otherwise with the JMP=xxx micro-instruction.

The JMP=xxx micro-instruction, where the "xxx" may be replaced with a large variety of binary bit patterns, allows non-sequential conditional branching within and between micro-program memory pages. As an example, consider the microcode sequence:

```
3 ::  ALU=A+1 DEST=DHI ;;
4 ::  ALU=A+1 DEST=DHI JMP=110 ;;
```

DECODING SEQUENCE

CYCLE	ACTION
DEC0	Clock interrupt register at falling clock edge IF interrupt, then clock MPC with value 1 ELSE clock MPC with IL value ENDIF
DEC1	Increment address counter on rising clock edge Enable next address register outputs to RADxx IF EXIT, then clock Ram Address Latch from RS and enable RAL outputs instead of Next Addr Reg ENDIF IF CALL then decrement RP ENDIF
DEC2	(Note: DEC2 from one instruction may occur simultaneously with DEC0 of the next instruction) Enable Next Addr Reg outputs to RADxx IF EXIT, then enable RAL outputs and increment RP ENDIF IF CALL, then write address counter to RS on rising clock edge ENDIF Clock next address register and instruction latch from RD bus contents. Clock address counter from address bus

INTERRUPT HANDLING ACTIONS

If an interrupt occurs, then the MPC is loaded with a reference to instruction # 1 (interrupt handler) on DECO. The interrupt handling microcode at opcode 1 is expected to save the contents of the address counter during the DEC2 cycle, which will point to the word AFTER the restart word for returning from the interrupt. Opcode 1 is also expected to save the contents of the MPC/IL (specifically, the CALL and EXIT bits) and to set the interrupt mask bit. IMPORTANT: The interrupt handling word must wait one clock cycle after setting the interrupt mask before doing its own DECODE!!!

Since CALL and EXIT functions are allowed to proceed during an interrupt, the interrupt handling high level code must un-do the return stack pointer actions caused by aborted CALLs and EXITS.

INTERRUPT FLAGS

BIT	CONTENTS
24	DATA STACK POINTER ERROR
25	RETURN STACK POINTER ERROR
26	HOST SERVICE REQUEST (HOST WROTE TO SERVICE REQ REG)
27	DYNAMIC RAM PARITY ERROR (UNIMPLEMENTED)
28	Software interrupt #2
29	Software interrupt #1
30	Software interrupt #0
31	INTERRUPT MASK (1=ENABLE 0=MASK)

FILE:ADDR7 DATE:09-02-1987

GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 1

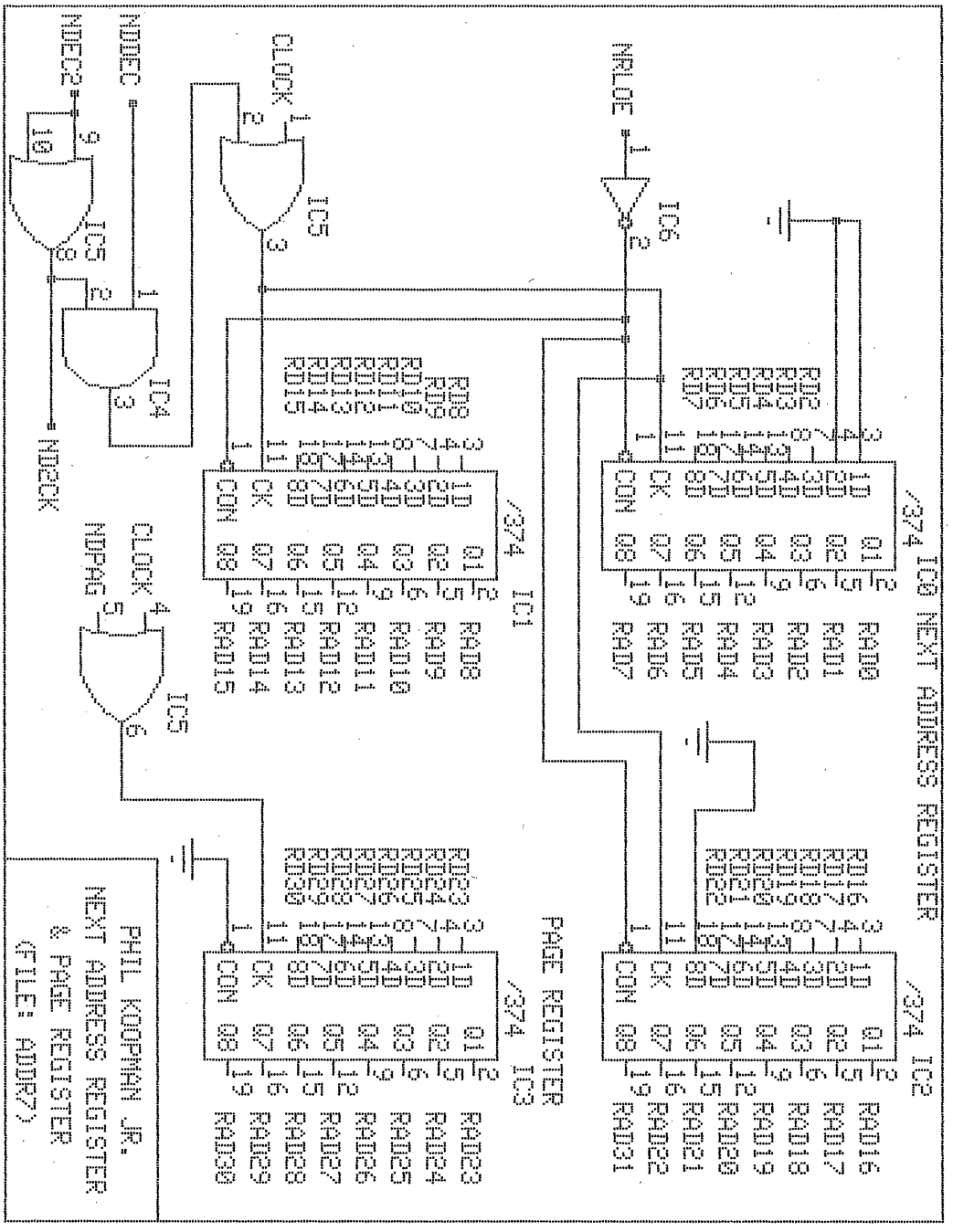
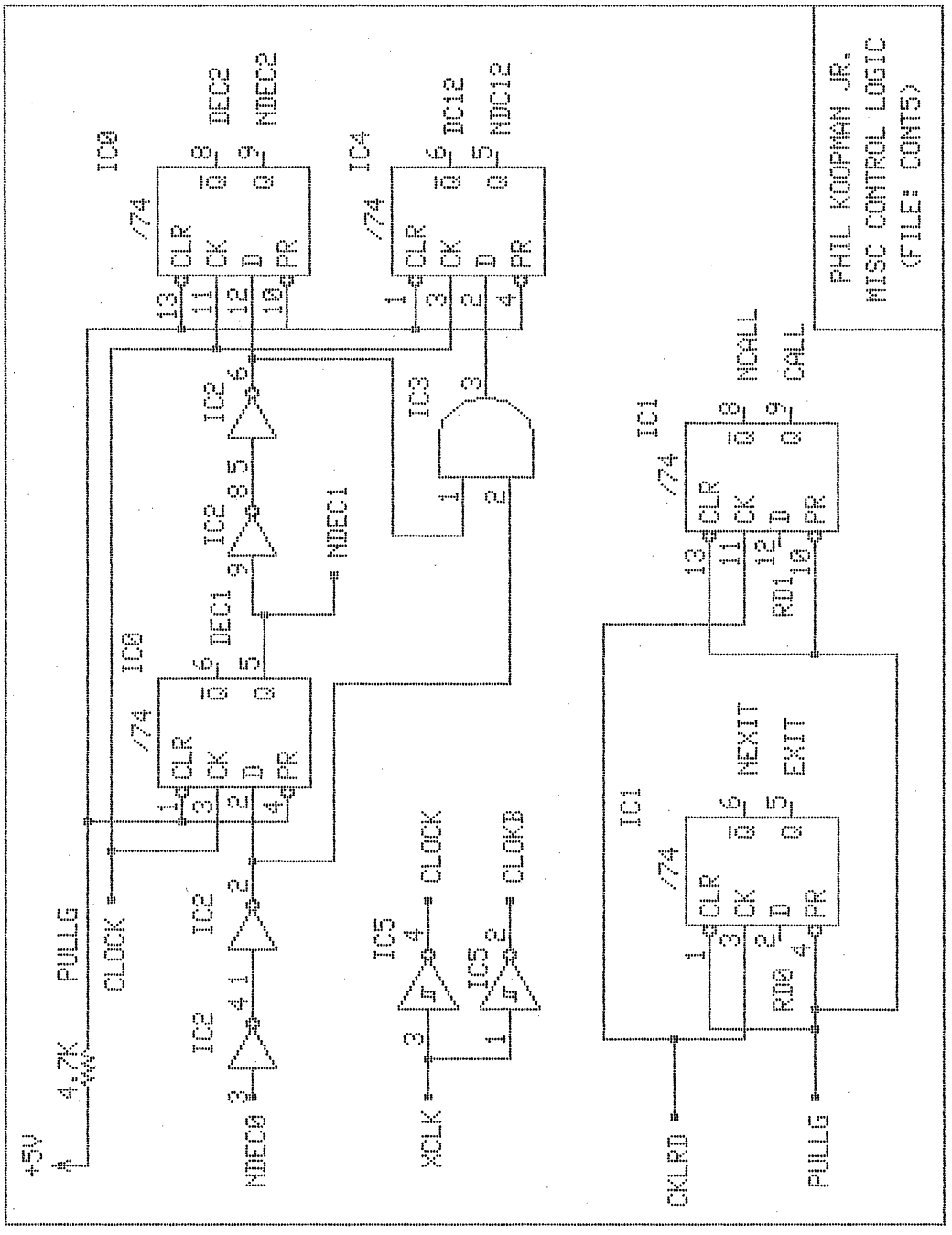


FIGURE 59

FIGURE 65

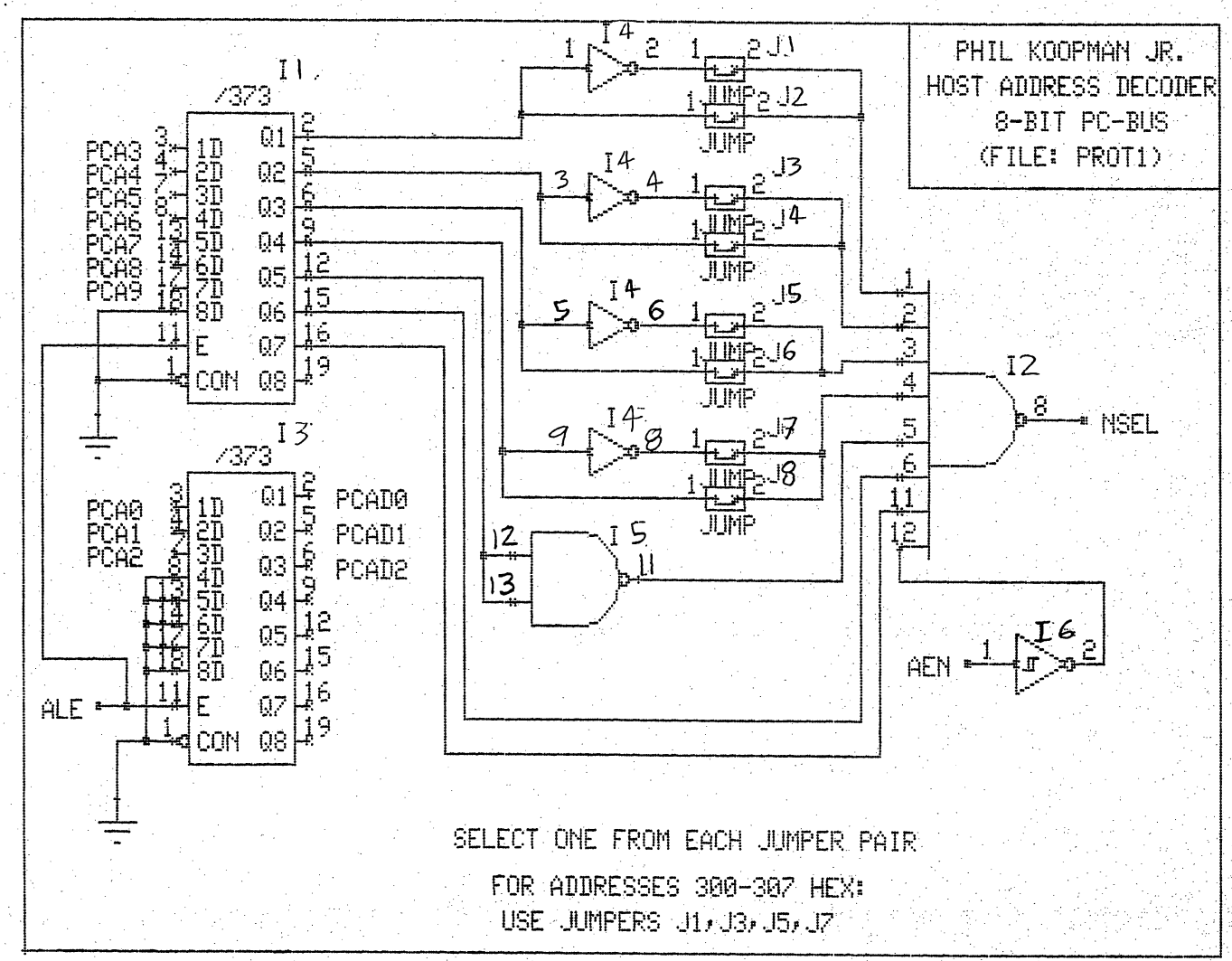


GRAPHICS SCREEN IMAGE - SCHEMATIC SHEET- H: 1 V: 1

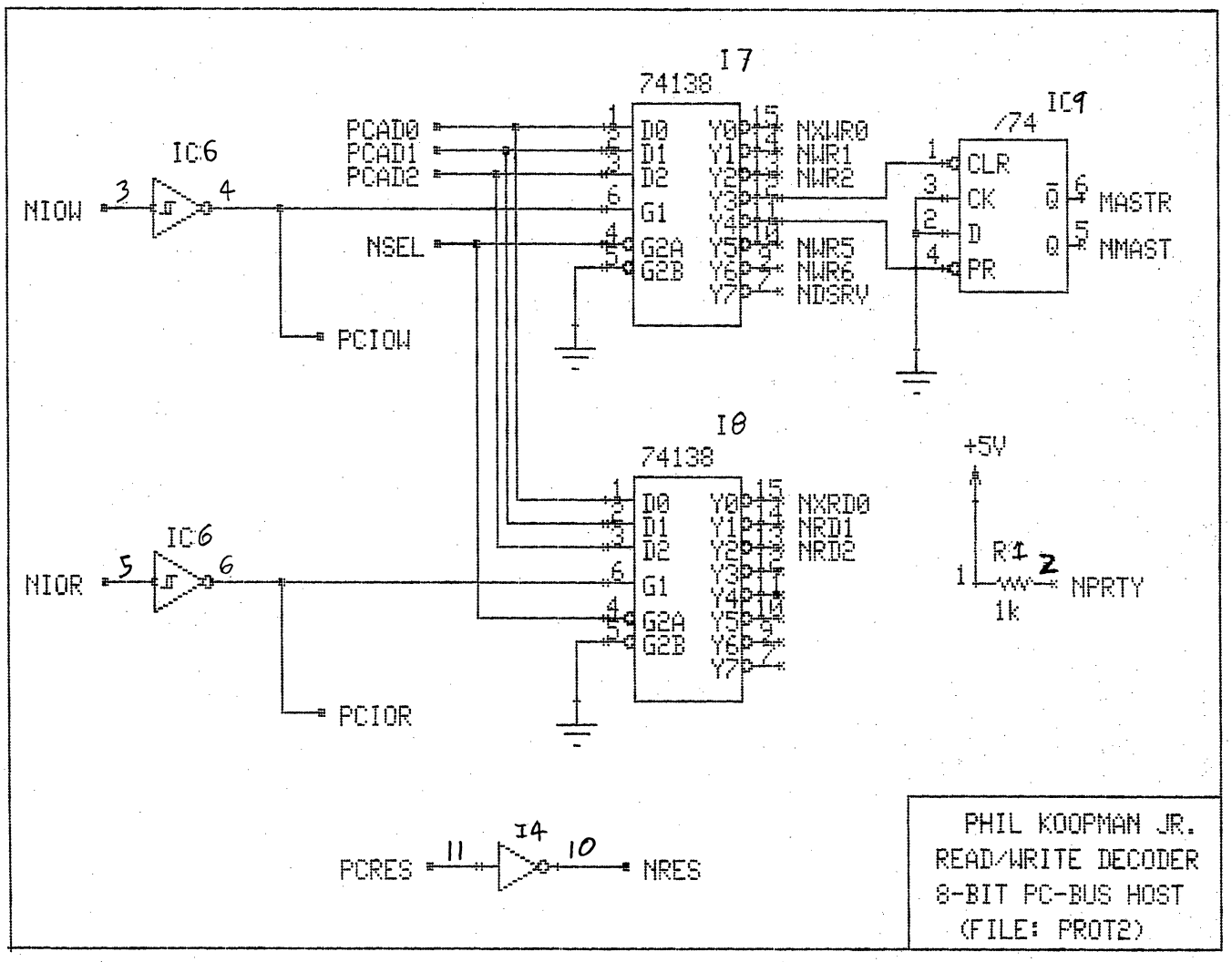
FILE:CONT5 DATE:09-02-1987

APPENDIX E. SCHEMATICS FOR A PROTOTYPE TEST BOARD

The following pages are schematics for designing a prototype test board for evaluation of the first Chip implementation. The WISC Chip set is not included in the schematics, since the pinout is unknown. The WISC chip pins should be connected to all signals with identical names in these schematics.

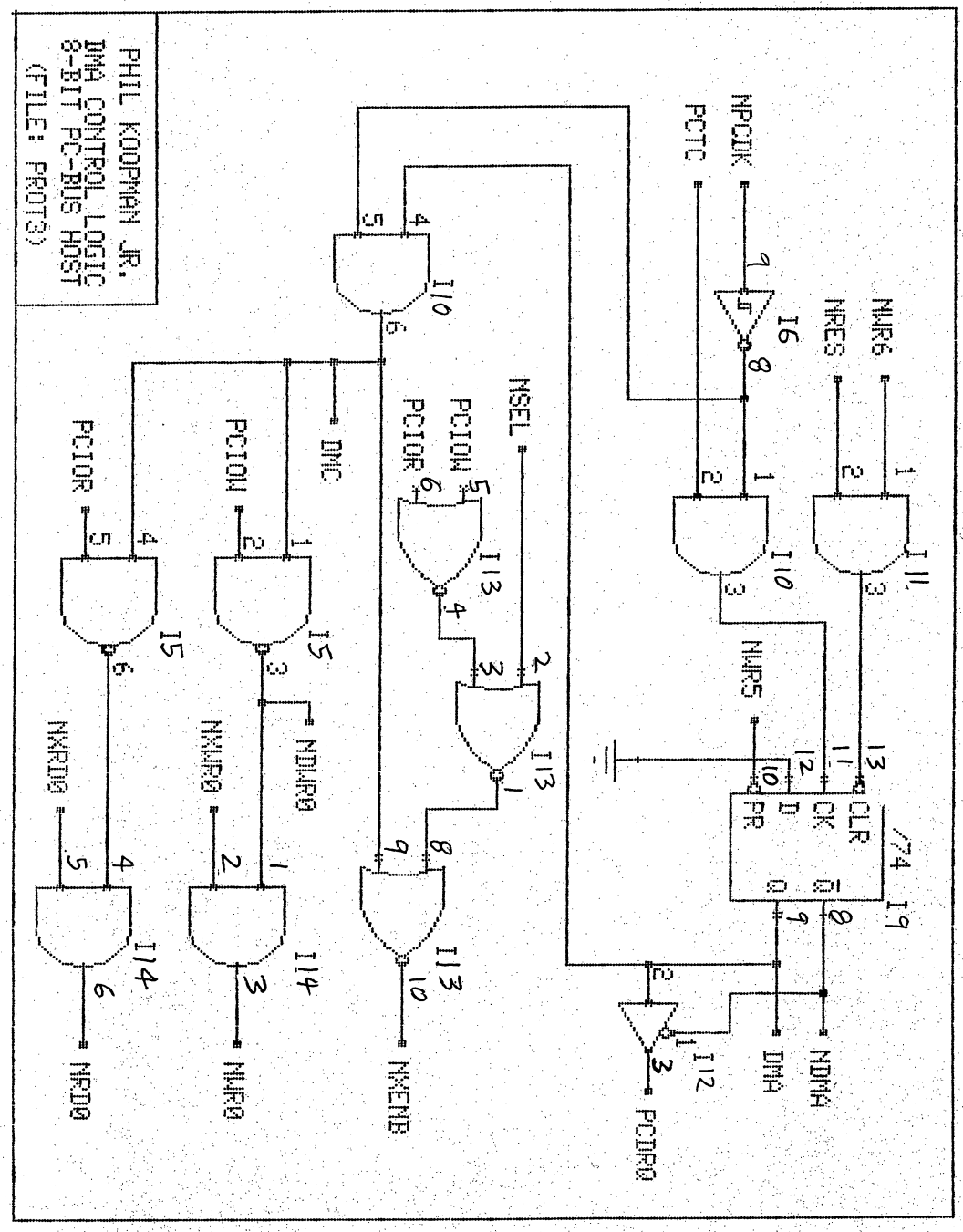


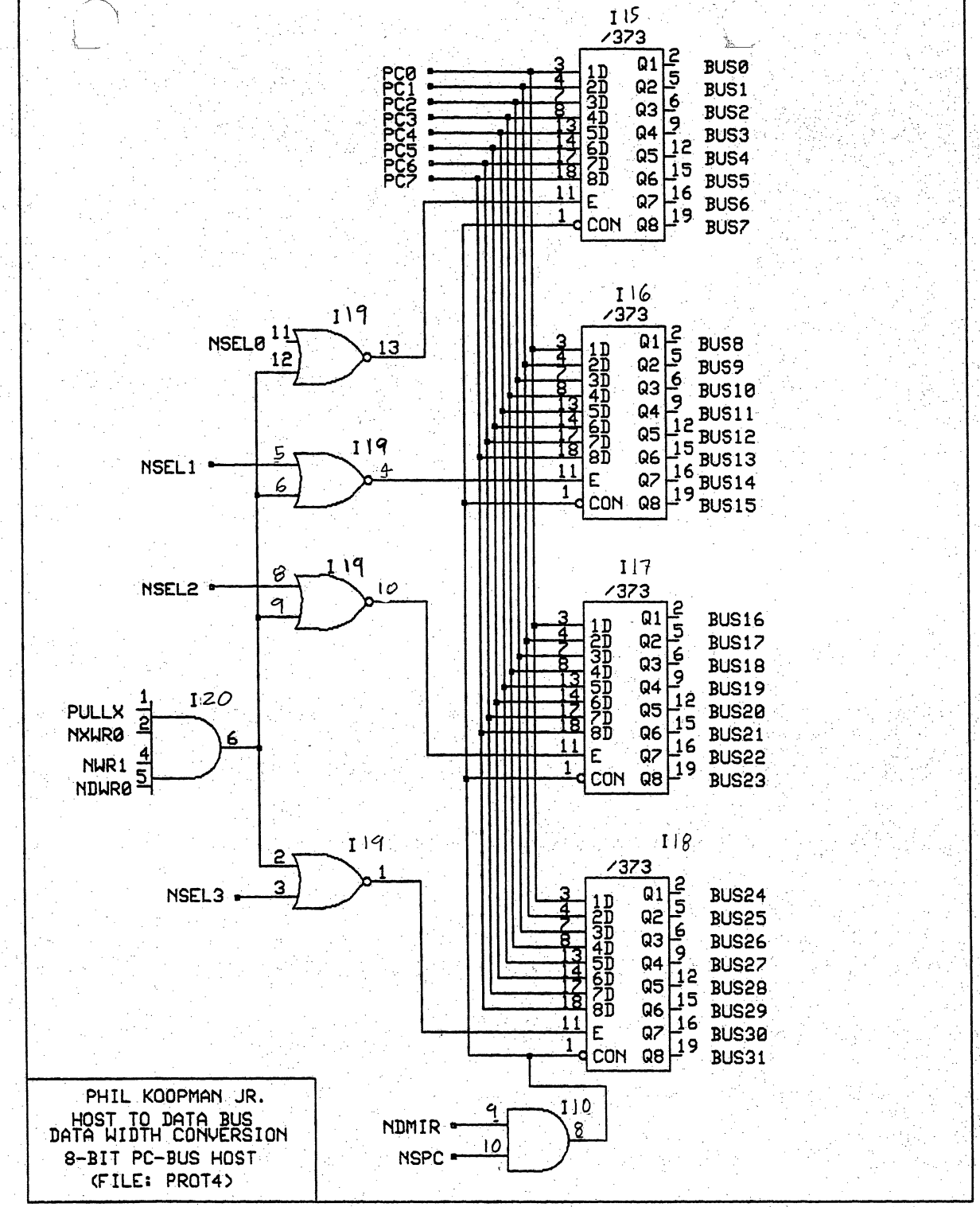
FILE: PROT1 DATE: 03-04-1988

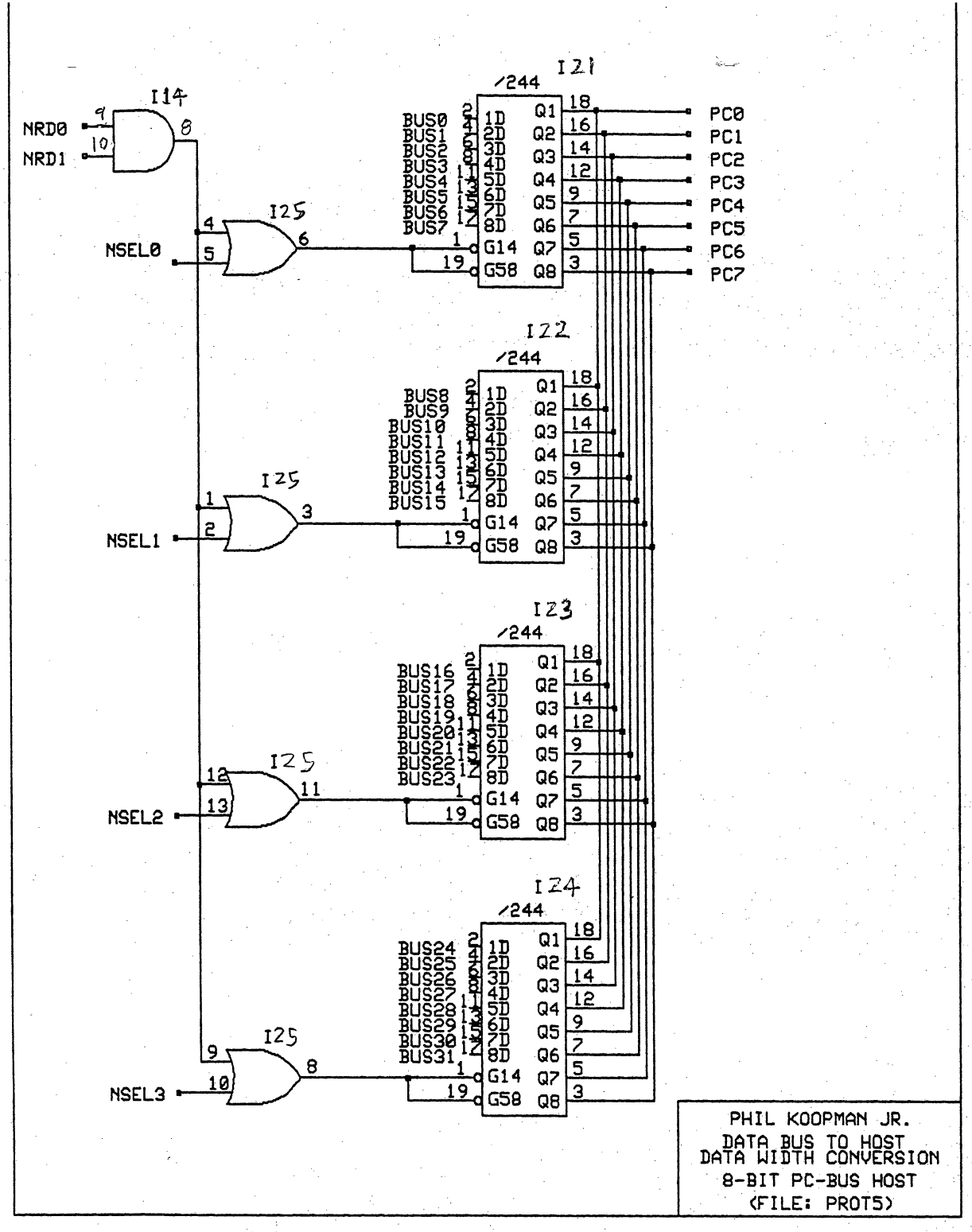


FILE: PROT2 DATE: 03-04-1988

FILE: PROT3 DATE:03-04-1988

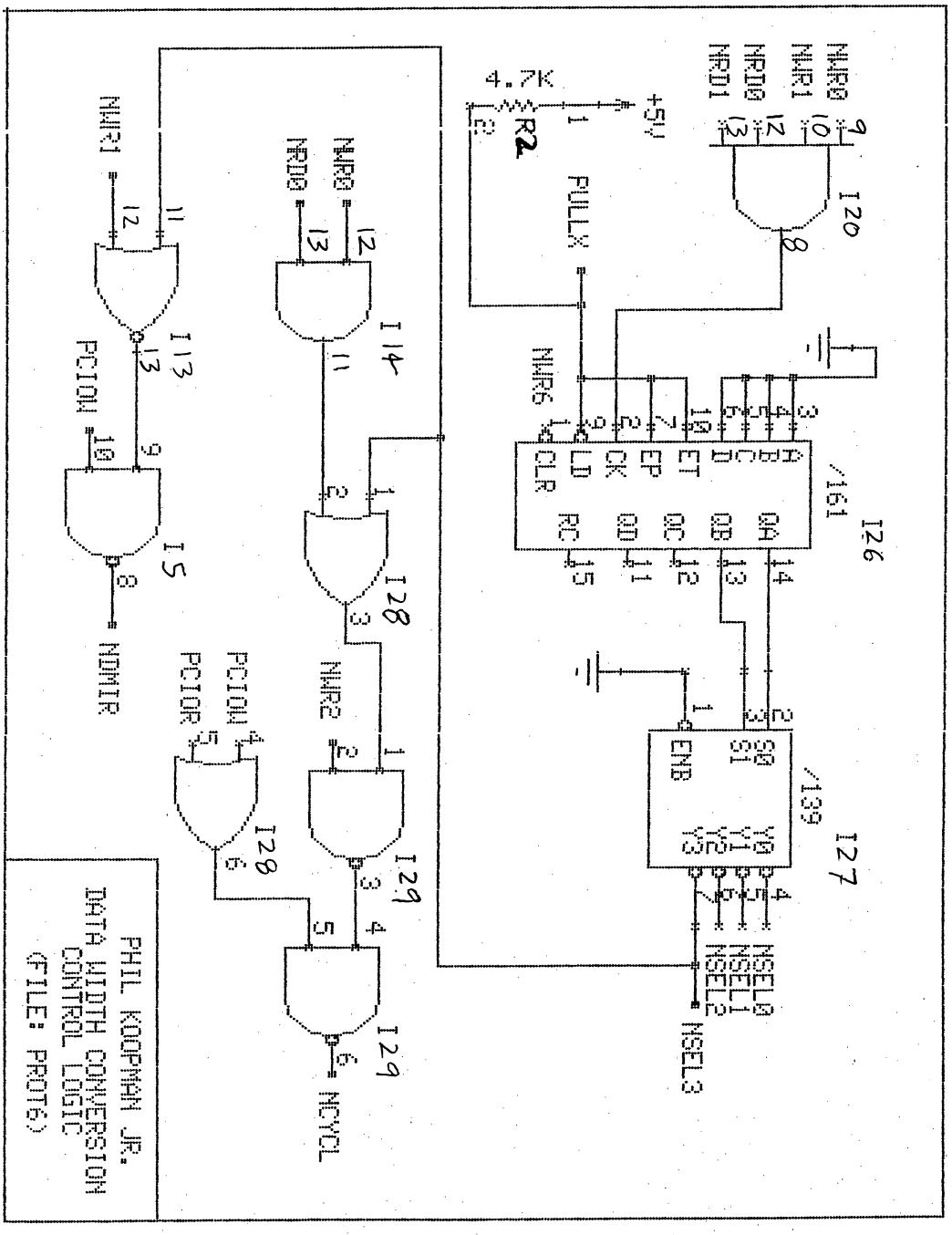



I15
/373
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
1D
2D
3D
4D
5D
6D
7D
8D
E
CON
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
BUS0
BUS1
BUS2
BUS3
BUS4
BUS5
BUS6
BUS7
I16
/373
BUS8
BUS9
BUS10
BUS11
BUS12
BUS13
BUS14
BUS15
I17
/373
BUS16
BUS17
BUS18
BUS19
BUS20
BUS21
BUS22
BUS23
I18
/373
BUS24
BUS25
BUS26
BUS27
BUS28
BUS29
BUS30
BUS31
I19
NSEL0
NSEL1
NSEL2
NSEL3
I20
PULLX
NXWR0
NWR1
NDWR0
I10
NDMIR
NSPC
PHIL KOOPMAN JR.
HOST TO DATA BUS
DATA WIDTH CONVERSION
8-BIT PC-BUS HOST
(FILE: PROT4)

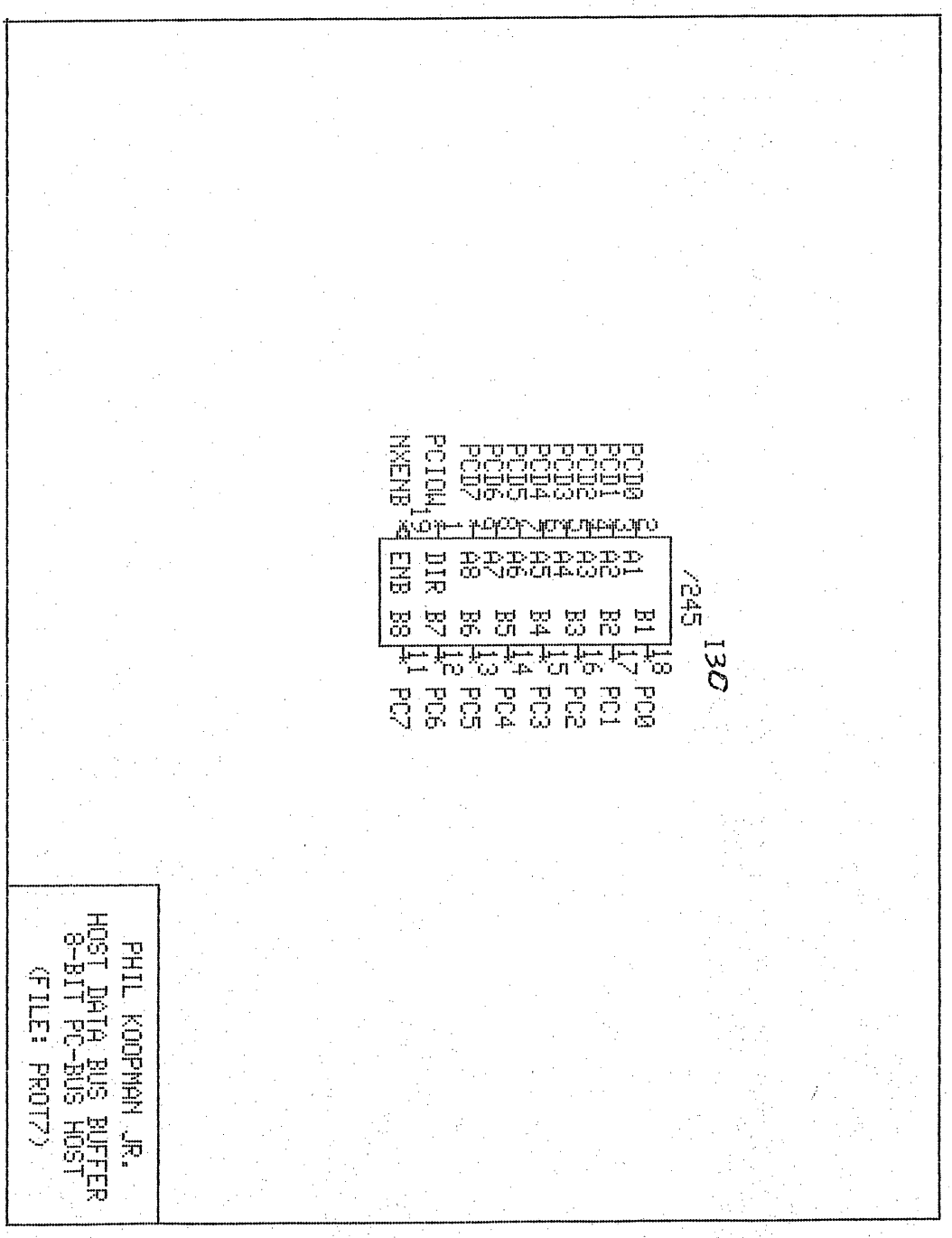


PHIL KOOPMAN JR.
DATA BUS TO HOST
DATA WIDTH CONVERSION
8-BIT PC-BUS HOST
(FILE: PROT5)

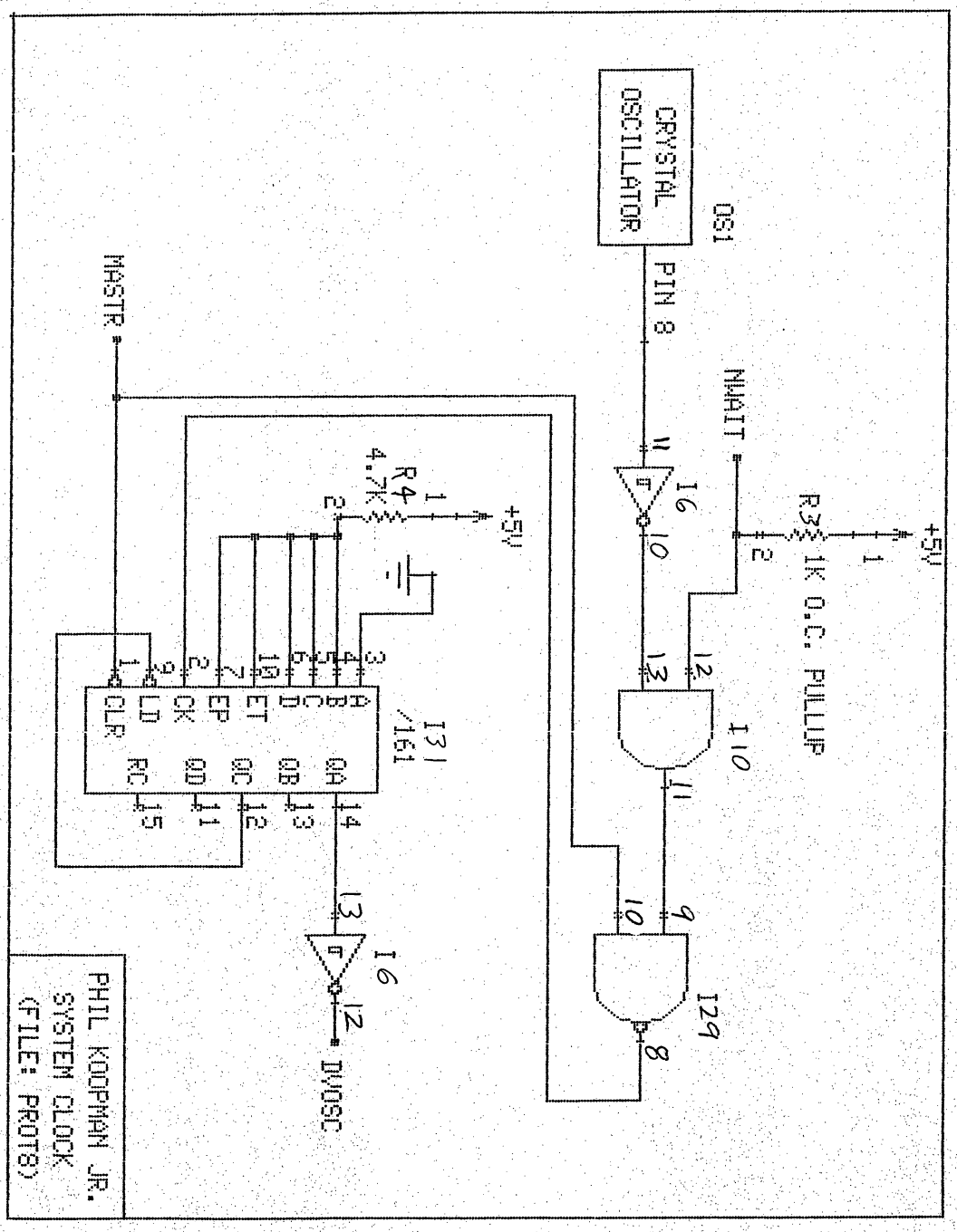
FILE: PROT6 DATE:03-04-1988



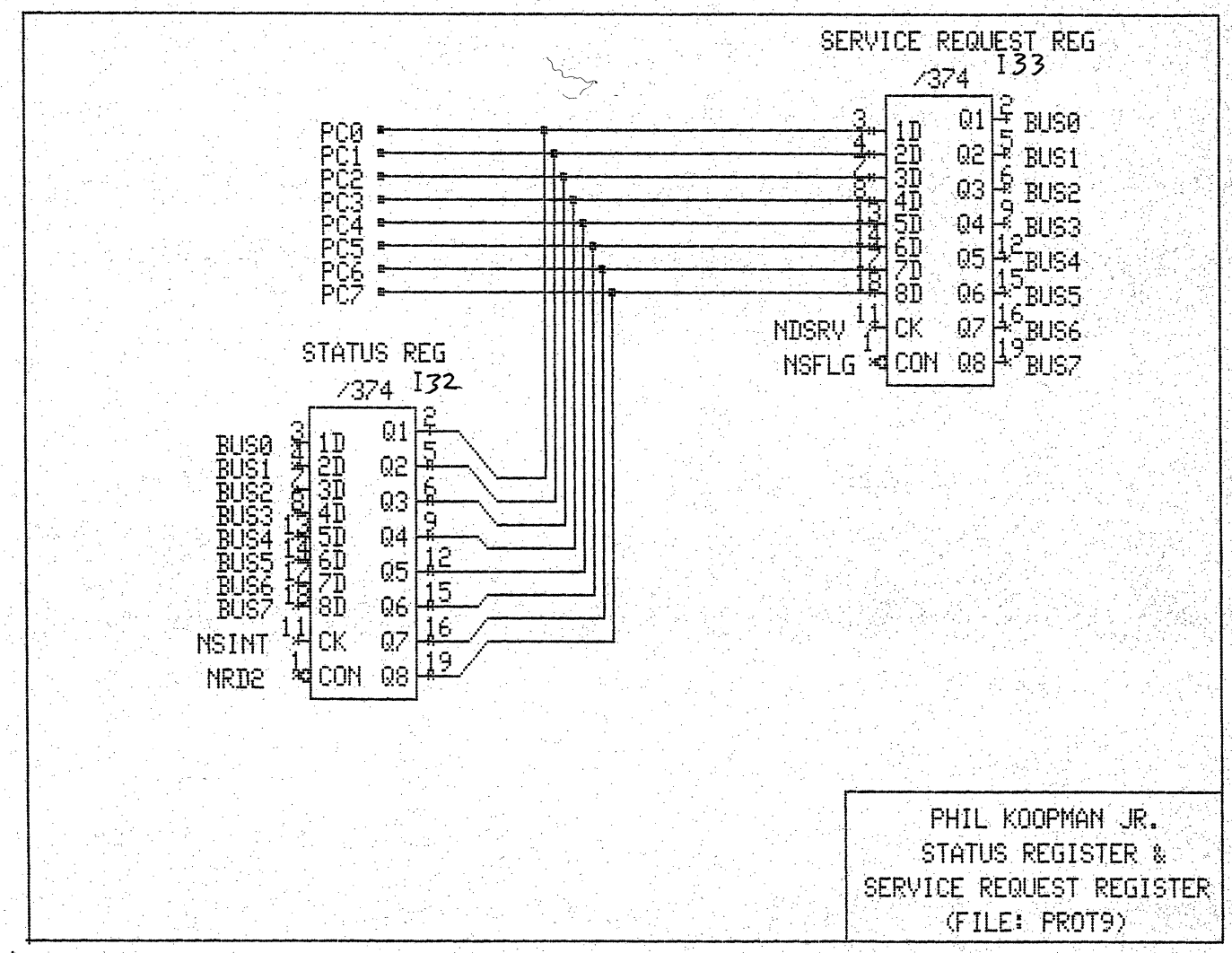
FILE: PROT7 DATE:03-04-1988



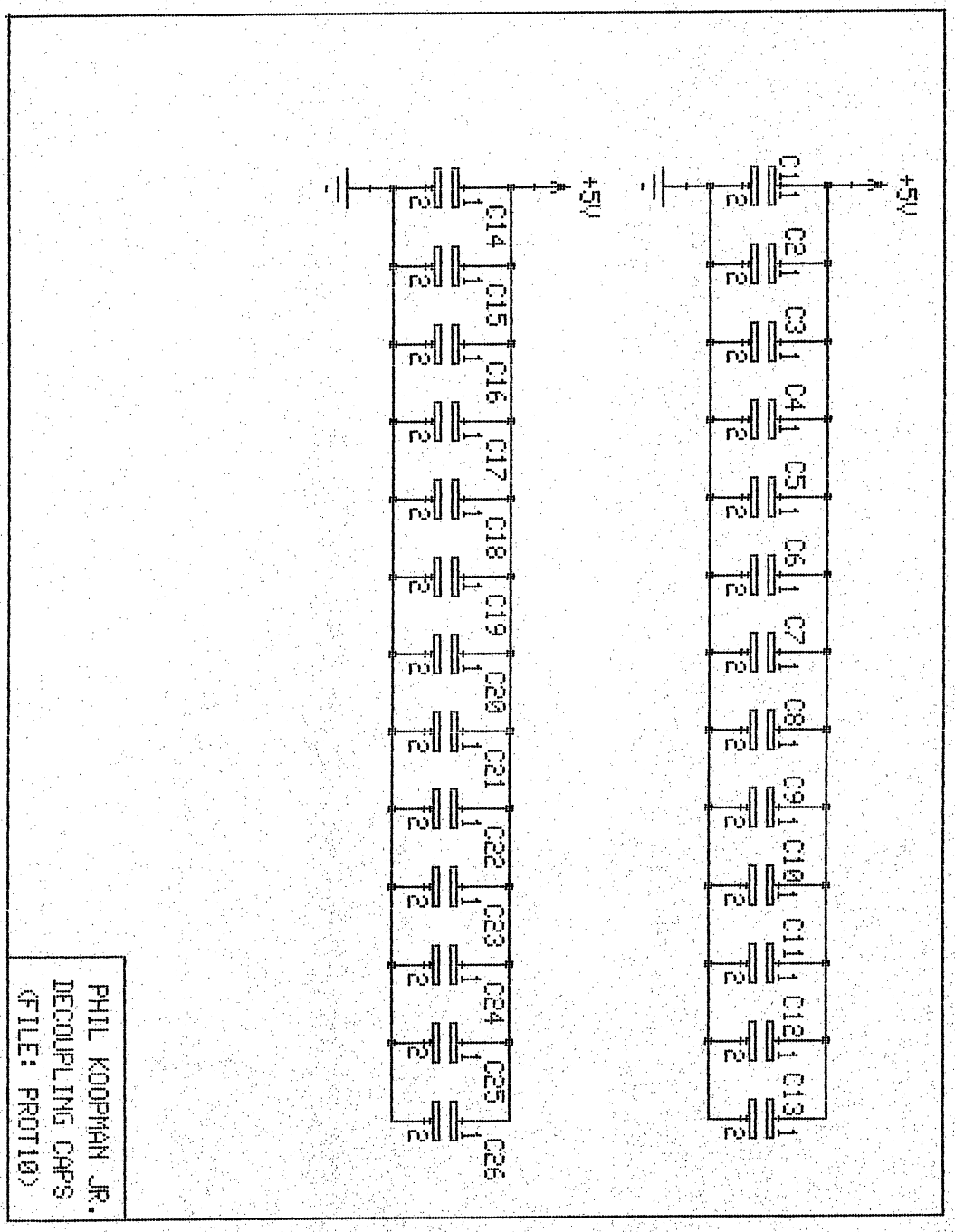
FILE: PROT8 DATE:03-04-1988



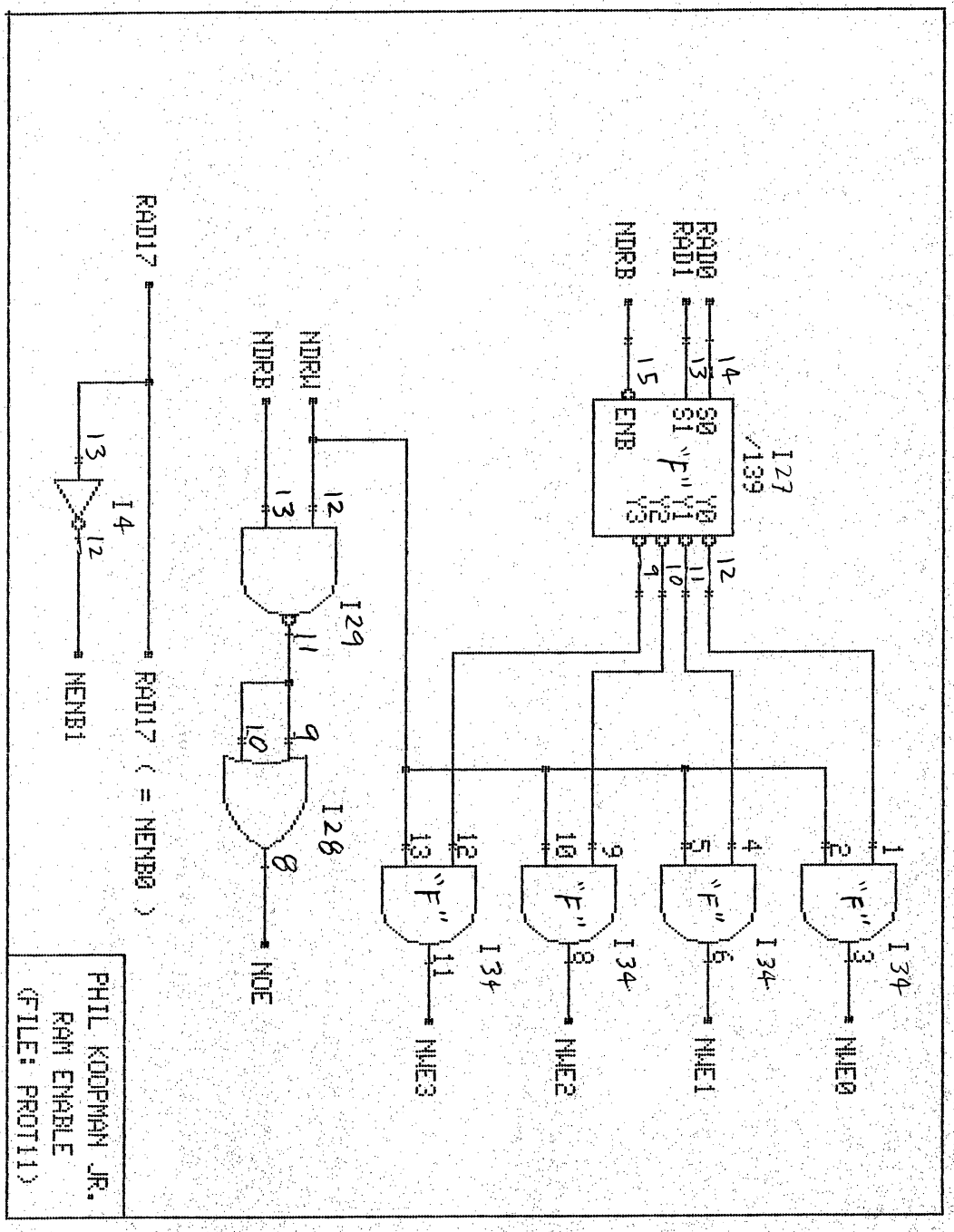
FILE: PROT9 DATE:03-04-1988



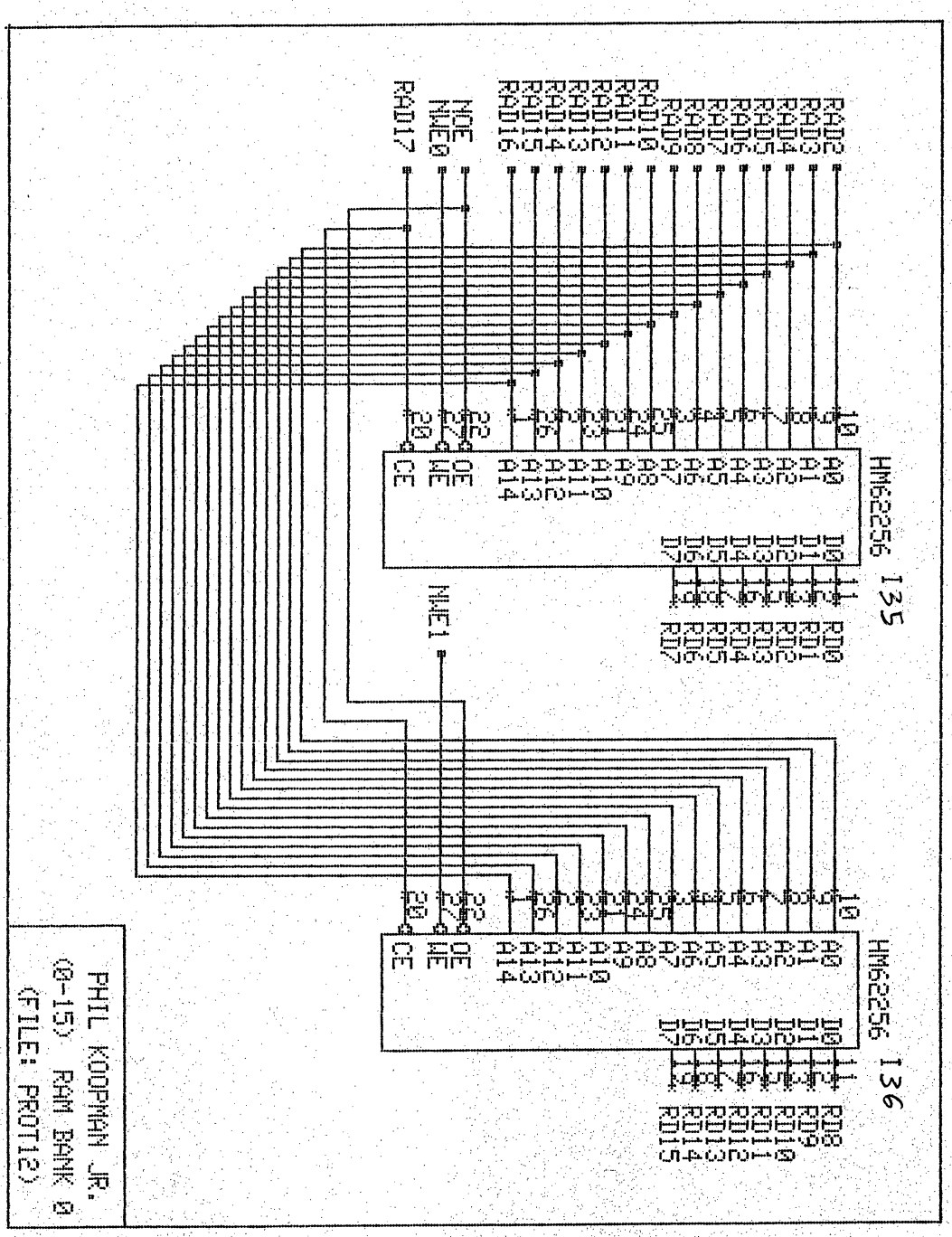
FILE: PROT10 DATE:03-04-1988



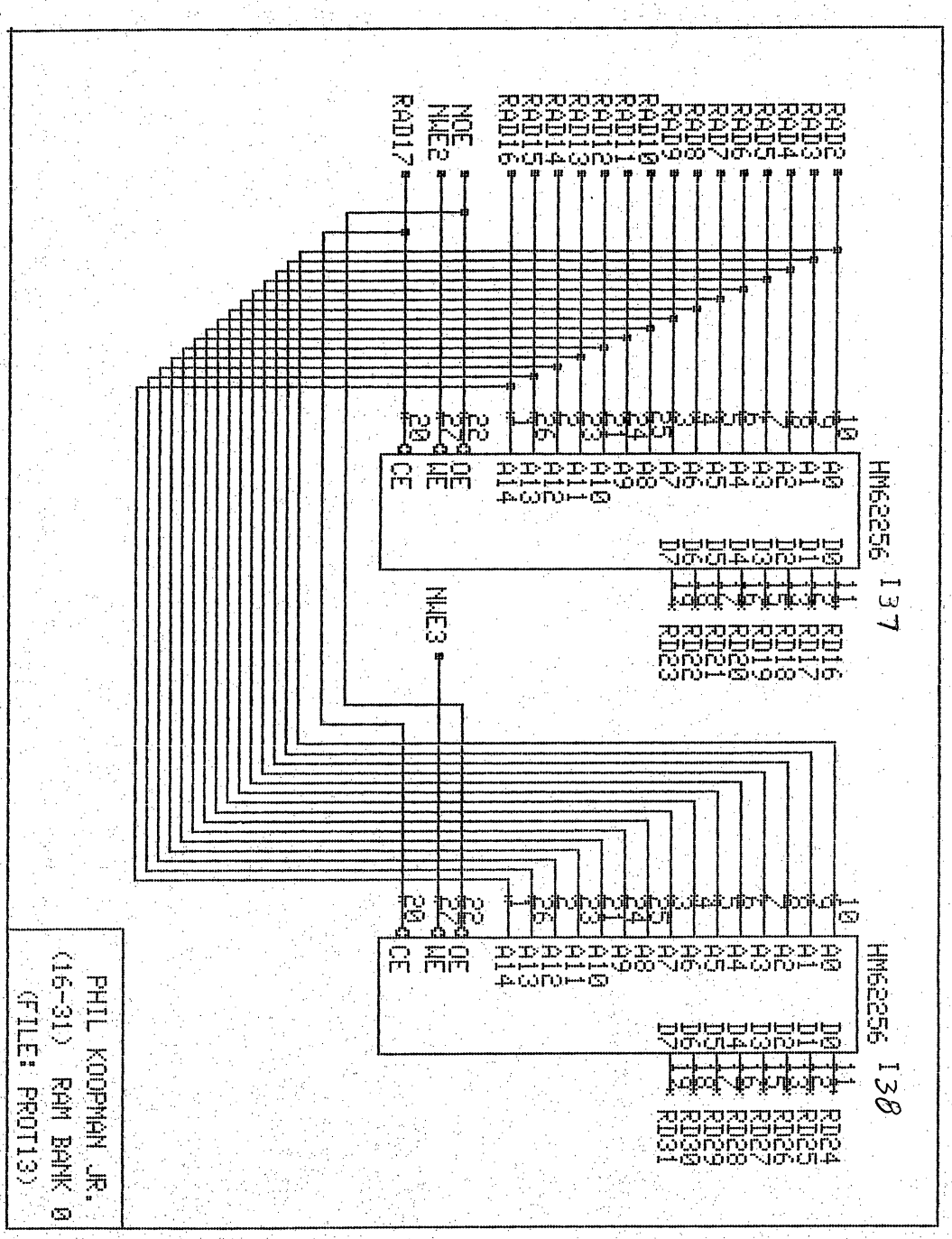
FILE: PROT11 DATE:03-04-1988



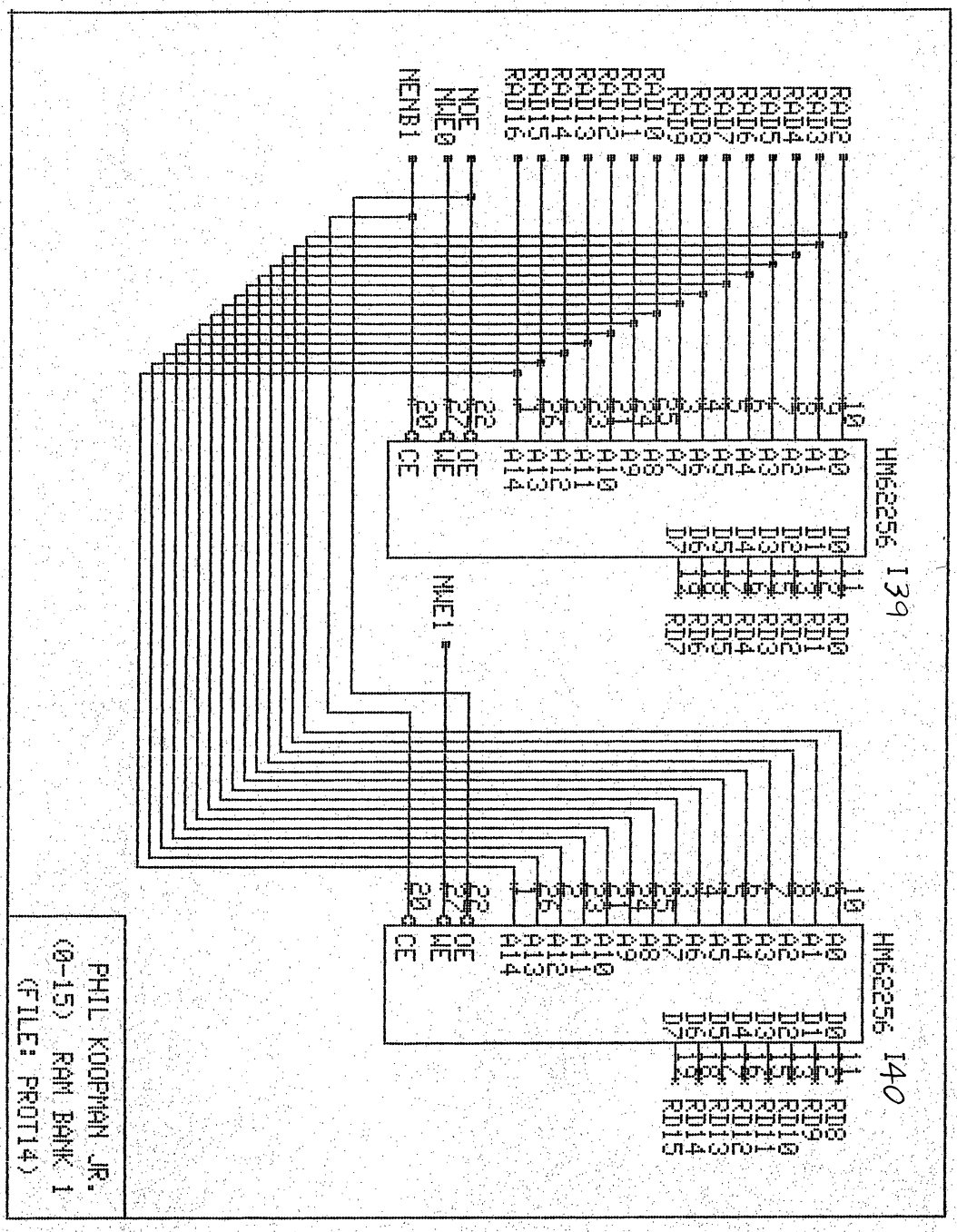
FILE: PROT12 DATE:03-04-1988



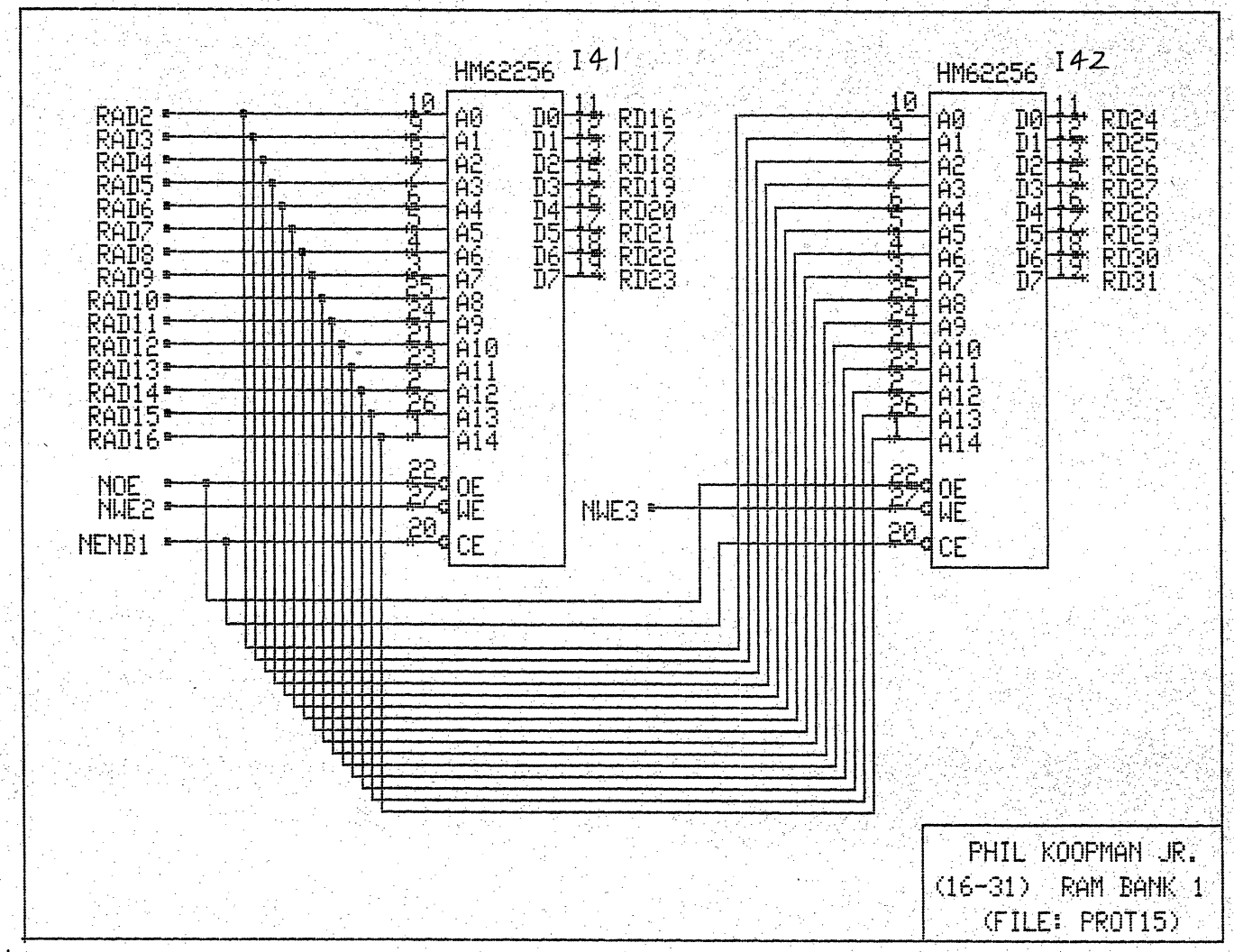
FILE: PROT13 DATE:03-04-1988



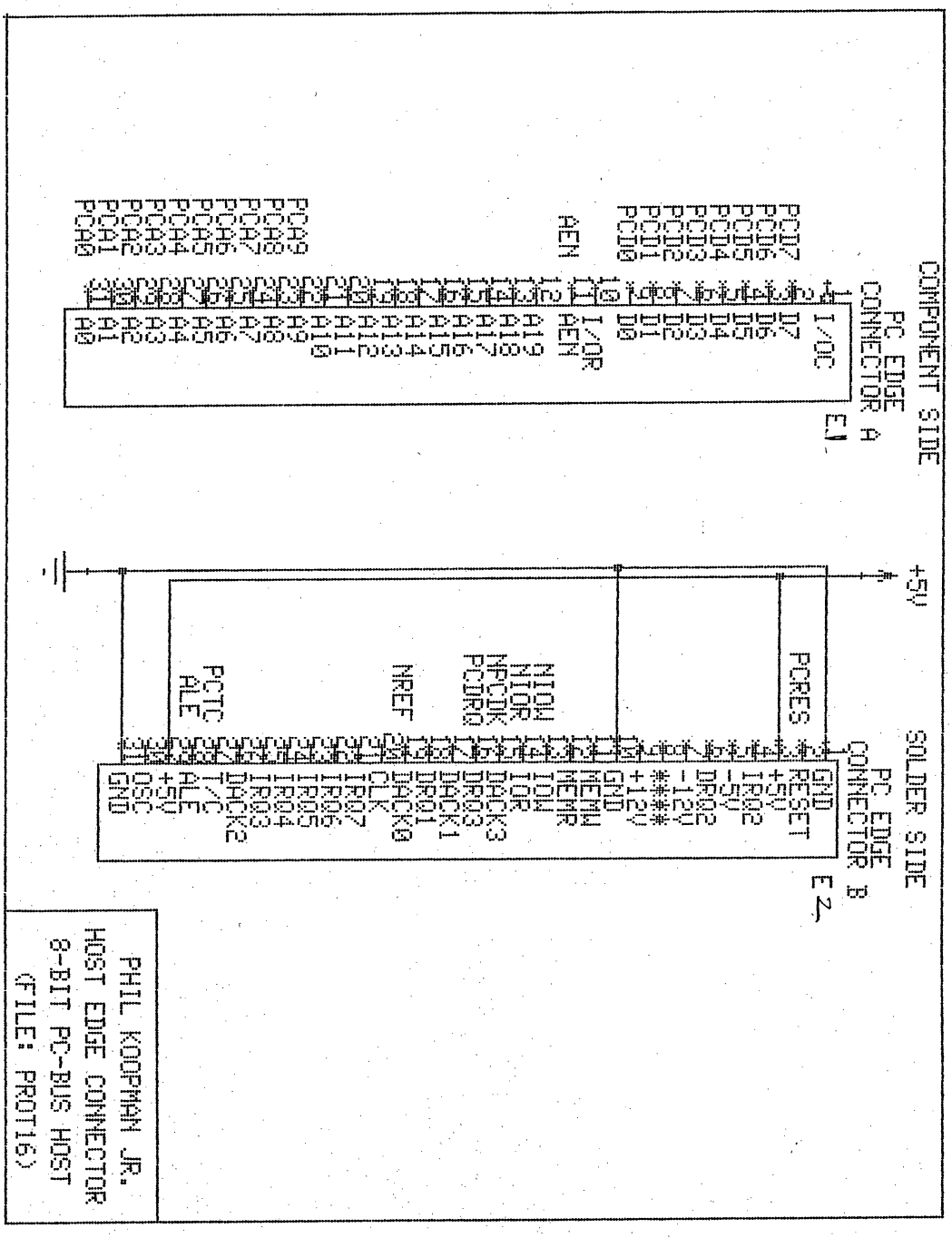
FILE: PROT14 DATE:03-04-1988



FILE: PROT15 DATE:03-04-1988

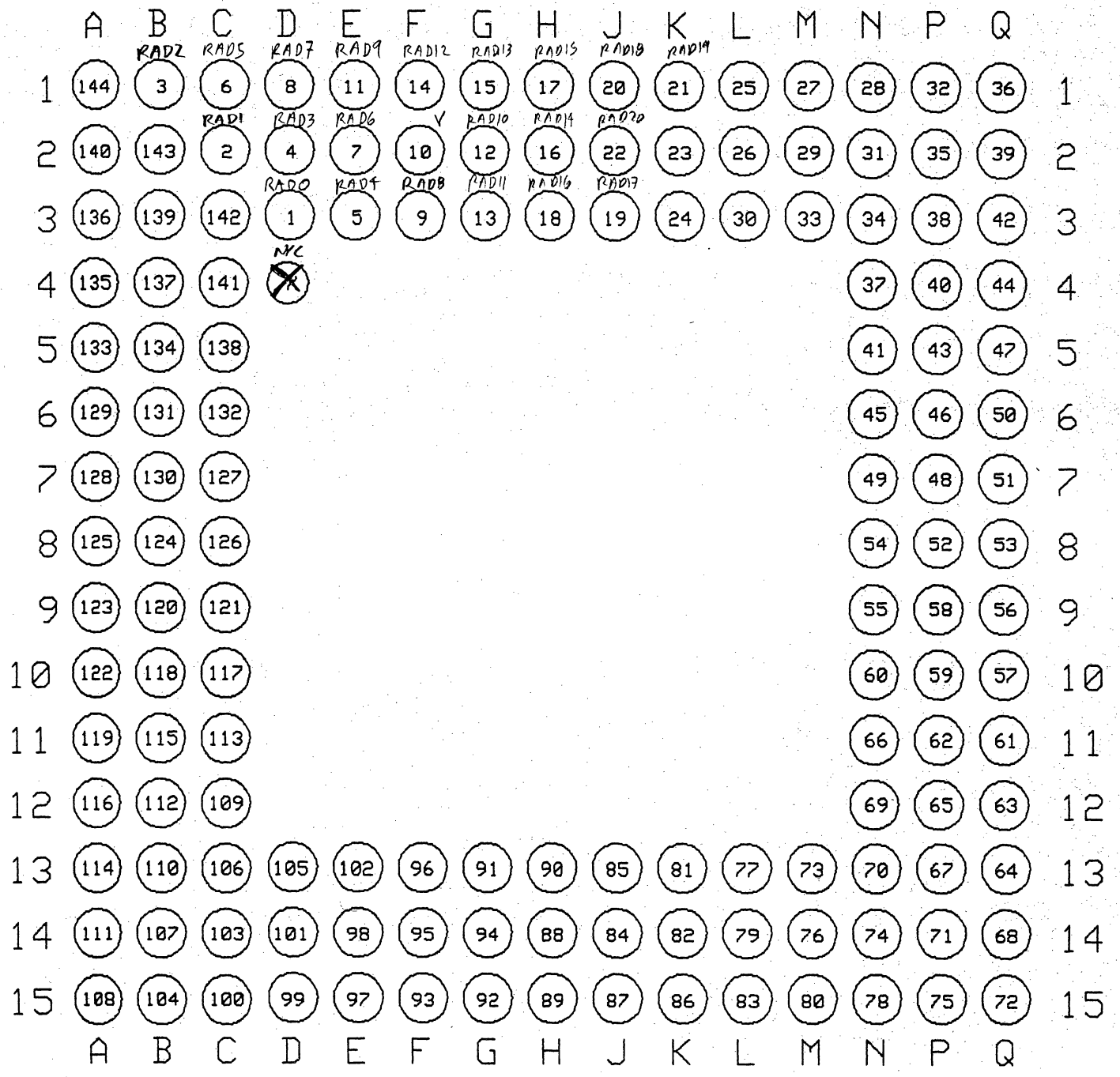


FILE: PROT16 DATE:03-04-1988



PROT-17A

145 PIN -- CONTROL CHIP



PIN SIDE UP

PROT - 17 B

HARRIS STANDARD CELL DEVELOPMENT GROUP C29

Wisc Processor Control Chip Pin List

PIN NAME	PACKAGE PIN #	BUFFER TYPE
RAD[0:8]	1:9	SC7200 - OUTPUT 100fp/10ns
VSS	10	SC7900 - DOUBLE VSS PAD
RAD[9:20]	11:22	SC7200 - OUTPUT 100fp/10ns
VSS	23	SC7900 - DOUBLE VSS PAD
RAD[21:22]	24:25	SC7200 - OUTPUT 100fp/10ns
RD[0:9]	26:35	SC7270 - BIDIRECTIONAL TTL INPUT W/PD
VDD	36	SC7910 - DOUBLE VDD PAD
VSS	37	SC7900 - DOUBLE VSS PAD
RD[10:21]	38:49	SC7270 - BIDIRECTIONAL TTL INPUT W/PD
VSS	50	SC7900 - DOUBLE VSS PAD
RD[22:31]	51:60	SC7270 - BIDIRECTIONAL TTL INPUT W/PD
NDIV	61	SC7200 - OUTPUT 100fp/10ns
NDDS	62	SC7200 - OUTPUT 100fp/10ns
NDDP	63	SC7200 - OUTPUT 100fp/10ns
NMROE	64	SC7200 - OUTPUT 100fp/10ns
MRXDR	65	SC7200 - OUTPUT 100fp/10ns
VSS	66	SC7900 - DOUBLE VSS PAD
NSMIR	67	SC7150 - TTL INPUT
NPRTY	68	SC7150 - TTL INPUT
NMAST	69	SC7150 - TTL INPUT
NDSRV	70	SC7150 - TTL INPUT
NDPER	71	SC7150 - TTL INPUT
VDD	72	SC7910 - DOUBLE VDD PAD
NDMIR	73	SC7150 - TTL INPUT
NDMA	74	SC7150 - TTL INPUT
NCYCL	75	SC7150 - TTL INPUT
NALU0	76	SC7150 - TTL INPUT
NACOT	77	SC7150 - TTL INPUT
DVOSC	78	SC7150 - TTL INPUT
DLOLO	79	SC7150 - TTL INPUT
ALU31	80	SC7150 - TTL INPUT
BUS[0:9]	81:90	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	91	SC7900 - DOUBLE VSS PAD
BUS[10:19]	92:101	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	102	SC7900 - DOUBLE VSS PAD
BUS[20:24]	103:107	SC7250 - BIDIRECTIONAL TTL INPUT
VDD	108	SC7910 - DOUBLE VDD PAD

VSS = GND

PROT - 17C

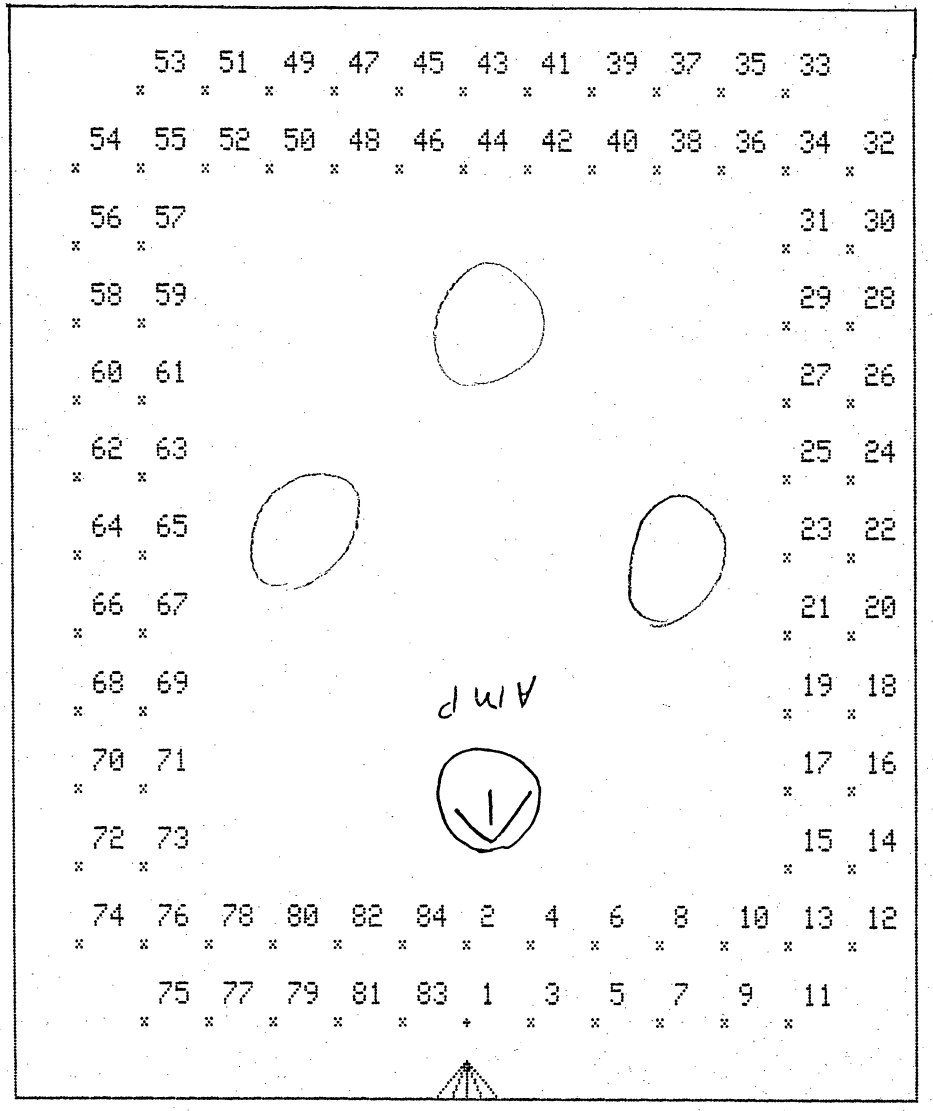
HARRIS STANDARD CELL DEVELOPMENT GROUP

Wisc Processor Control Chip Pin List *continued*

PIN NAME	PACKAGE PIN #	BUFFER TYPE
BUS[25:29]	109:113	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	114	SC7900 - DOUBLE VSS PAD
BUS[30:31]	115:116	SC7250 - BIDIRECTIONAL TTL INPUT
NMRCE	117	SC7200 - OUTPUT 100fp/10ns
NWMRA	118	SC7200 - OUTPUT 100fp/10ns
NSPC	119	SC7200 - OUTPUT 100fp/10ns
NSFLG	120	SC7200 - OUTPUT 100fp/10ns
NSDS	121	SC7200 - OUTPUT 100fp/10ns
NSDP	122	SC7200 - OUTPUT 100fp/10ns
NSDLO	123	SC7200 - OUTPUT 100fp/10ns
NSDHI	124	SC7200 - OUTPUT 100fp/10ns
NMULT	125	SC7200 - OUTPUT 100fp/10ns
NMRXE	126	SC7200 - OUTPUT 100fp/10ns
VSS	127	SC7900 - DOUBLE VSS PAD
NSINT	128	SC7200 - OUTPUT 100fp/10ns
NRAM	129	SC7200 - OUTPUT 100fp/10ns
NDRW	130	SC7200 - OUTPUT 100fp/10ns
NDRB	131	SC7200 - OUTPUT 100fp/10ns
MAD[10:3]	132:139	SC7200 - OUTPUT 100fp/10ns
VSS	140	SC7900 - DOUBLE VSS PAD
MAD[2:0]	141:143	SC7200 - OUTPUT 100fp/10ns
VDD	144	SC7910 - DOUBLE VDD PAD

VSS = GND

84-pin LCC (DATA CHIP) PRL 18A



SYMBOL <PL84> DATE:03-04-1988

pin 1 of top ↓

PROT 18B

P1

HARRIS STANDARD CELL DEVELOPMENT GROUP

Wisc Processor Data Chip Pin List

PIN NAME	PACKAGE PIN #	BUFFER TYPE
NMRCE	1	SC7150 - TTL INPUT
NWMRA	2	SC7150 - TTL INPUT
NSMIR	3	SC7150 - TTL INPUT
NSDP	4	SC7150 - TTL INPUT
NSDS	5	SC7150 - TTL INPUT
NSDLO	6	SC7150 - TTL INPUT
NSDHI	7	SC7150 - TTL INPUT
NMULT	8	SC7150 - TTL INPUT
NMRXE	9	SC7150 - TTL INPUT
NDIV	10	SC7150 - TTL INPUT
VDD	11	SC7910 - DOUBLE VDD PAD
NC	12	
VSS	13	SC7900 DOUBLE VSS PAD
NDDS	14	SC7150 - TTL INPUT
NDDP	15	SC7150 - TTL INPUT
NMROE	16	SC7150 - TTL INPUT
MRXDR	17	SC7150 - TTL INPUT
DVOSC	18	SC7150 - TTL INPUT
NCYCL	19	SC7150 - TTL INPUT
NDMIR	20	SC7150 - TTL INPUT
NMAST	21	SC7150 - TTL INPUT
VDD	22	SC7910 - DOUBLE VDD PAD
VSS	23	SC7900 - DOUBLE VSS PAD
BUS[0:7]	24:31	SC7250 - BIDIRECTIONAL TTL INPUT
VDD	32	SC7910 - DOUBLE VDD PAD
NC	33	
VSS	34	SC7250 - DOUBLE VSS PAD
BUS[8:15]	35:42	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	43	SC7900 - DOUBLE VSS PAD
BUS[16:23]	44:51	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	52	SC7900 - DOUBLE VSS PAD
VDD	53	SC7910 - DOUBLE VDD PAD
NC	54	
BUS[24:31]	55:62	SC7250 - BIDIRECTIONAL TTL INPUT
VSS	63	SC7900 - DOUBLE VSS PAD
VDD	64	SC7910 - DOUBLE VDD PAD
MAD[0:7]	65:72	SC7160 - CMOS INPUT

VSS = GND

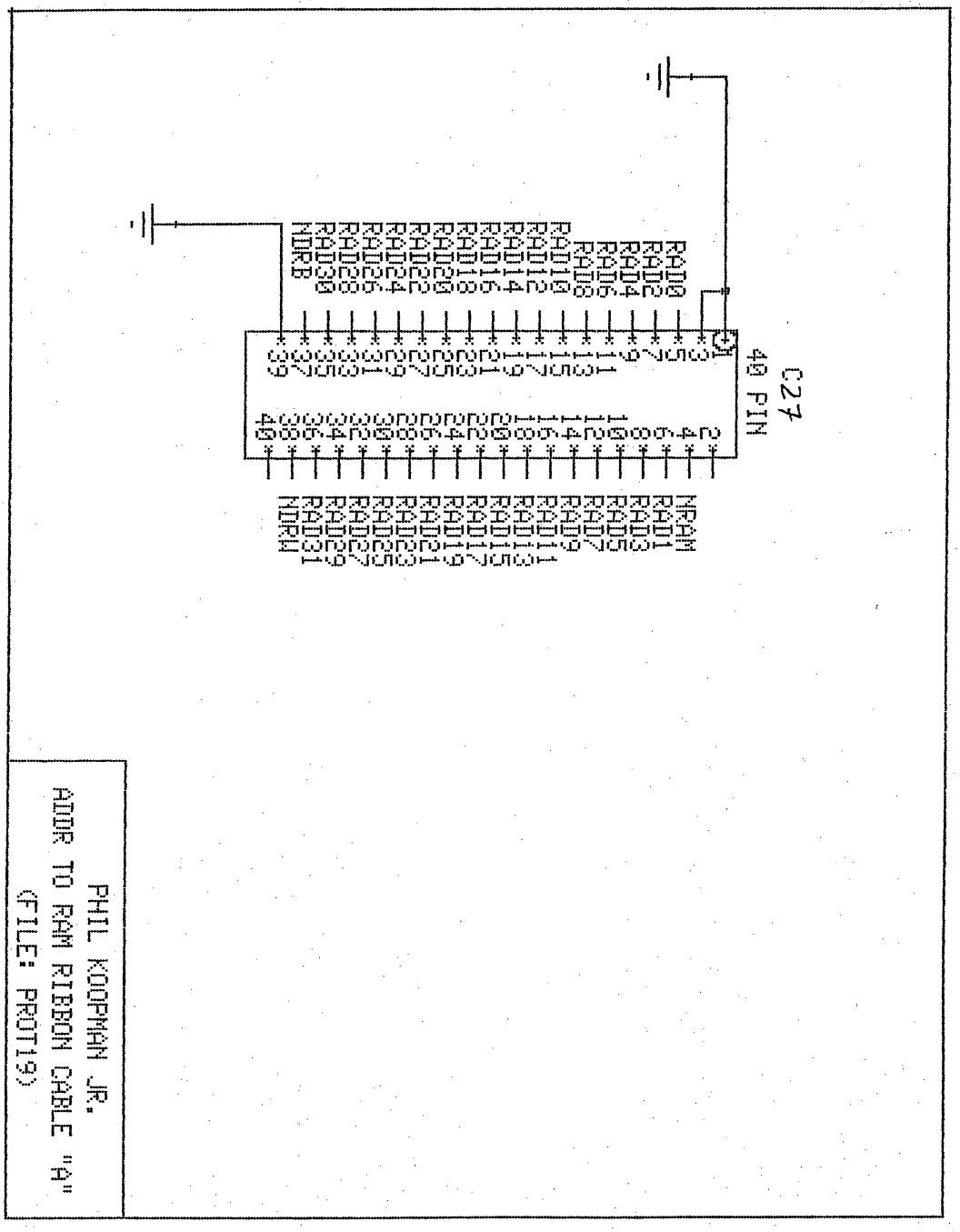
PROT 18C

Wisc Processor Data Chip Pin List *continued*

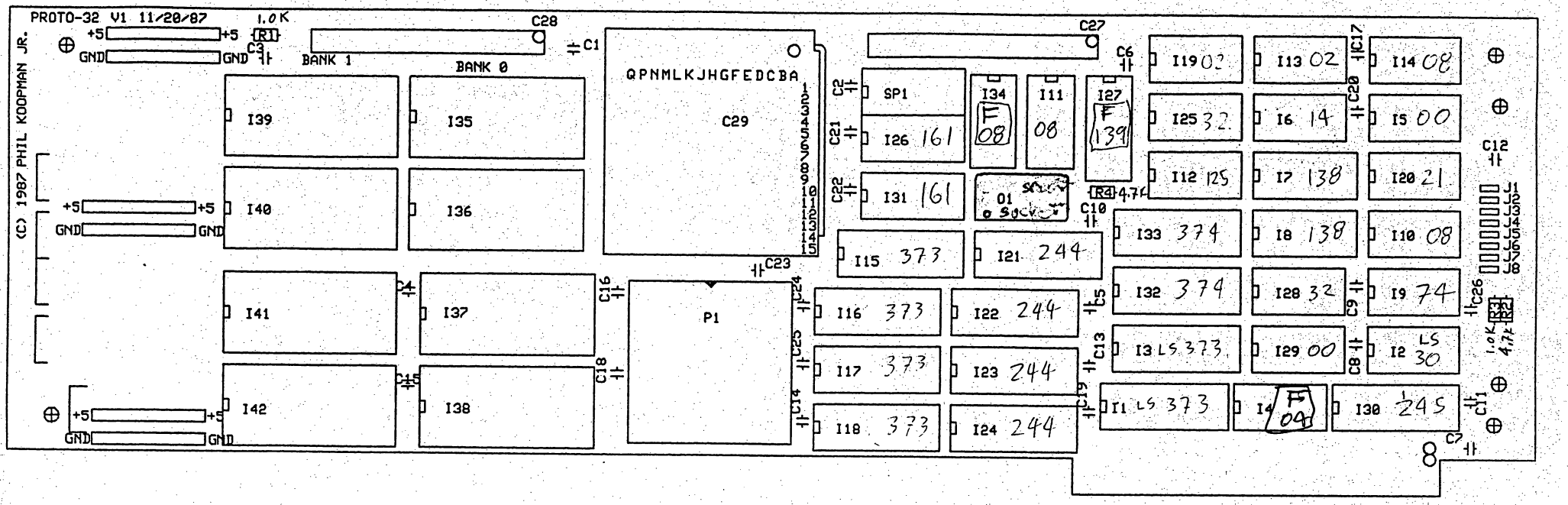
PIN NAME	PACKAGE PIN #	BUFFER TYPE
VSS	73	SC7900 - DOUBLE VSS PAD
MAD[8]	74	SC7160 - CMOS INPUT
NC	75	
VDD	76	SC7910 - DOUBLE VDD PAD
MAD[9:10]	77:78	SC7160 - CMOS INPUT
NDPER	79	SC7200 - OUTPUT 100pf/10ns
NALU0	80	SC7200 - OUTPUT 100pf/10ns
NACOT	81	SC7200 - OUTPUT 100pf/10ns
DLOLO	82	SC7200 - OUTPUT 100pf/10ns
ALU31	83	SC7200 - OUTPUT 100pf/10ns
VSS	84	SC7900 - DOUBLE VSS PAD

VSS = GND

FILE: PROT19 DATE:03-04-1988



FILE: PROT20 DATE: 11-09-1987


PROTO-32 V1 11/20/87
(C) 1987 PHIL KOOPMAN JR.
1.0K
R1
+5
GND
C3
C28
C1
BANK 1
BANK 0
QPNMLKJHGFEDCBA
C29
C27
C6
C17
C20
C12
C2
C21
C22
C10
C5
C9
C13
C8
C19
C11
C7
C23
C24
C25
C14
C16
C4
C15
C18
P1
SP1
I26 161
I34 74 08
I11 08
I27 F 139
R4 4.7K
I31 161
I15 373
I21 244
I16 373
I22 244
I17 373
I23 244
I18 373
I24 244
I40
I36
I41
I37
I42
I38
I39
I35
I19 02
I13 02
I14 08
I25 32
I6 14
I5 00
I12 125
I7 138
I20 21
I33 374
I8 138
I10 08
I32 374
I28 32
I9 74
I3 LS 373
I29 00
I2 LS 30
I1 LS 373
I4 F 04
I30 74 S 245
J1
J2
J3
1.0K
4.7K