# DESIGN AND MEASUREMENT OF A RECONFIGURABLE MULTI-MICROPROCESSOR MACHINE

by

Charles Zukowski

Submitted in partial fulfillment
of the requirements for the
degrees of

Bachelor of Science
and
Master of Science

at the

Massachusetts Institute of Technology

May, 1982



Signature of Author _____
Department of Electrical Engineering and
Computer Science, May 7, 1982

Certified by _____
Thesis Supervisor Donald E. Troxel, Associate Professor

Certified by _____
Company Supervisor Howard M. Brauer

Accepted by _____
Chairman, Departmental Committee on Graduate Students

# DESIGN AND MEASUREMENT OF A RECONFIGURABLE MULTI-MICROPROCESSOR MACHINE

by

Charles Zukowski

Submitted to the Department of Electrical Engineering
on May 7, 1982, in partial fulfillment of the
requirements for the Degrees of Bachelor of Science and
Master of Science.

## ABSTRACT

Multiple-microprocessor systems have become a cost effective method for implementing special high-throughput computer tasks. This thesis describes a flexible multi-microprocessor machine (MMU) designed for such tasks and measures its performance in actual operation. A simple model is developed for estimating its marginal performance. The hardware design is described in detail and compared to that of other multiprocessor machines designed in the past.

Thesis Supervisor: Donald E. Troxel
Title: Associate Professor of Electrical Engineering

## ACKNOWLEDGEMENTS

I would like to thank all of my friends at IBM Research and M.I.T. who were very helpful during the entire project presented in this thesis. In particular I would like to thank both Howard Brauer and Professor D. E. Troxel for their excellent supervision at IBM and M.I.T. respectively. I am also grateful to John Anthony and Pete Hennet for their help in making the MMU a reality. Also deserving mention are Dick Hadsell, Keith Milliken, and Martin Kienzle, as their ideas defined the software architecture of the machine necessary for its target application.

Table of Contents

# List of Figures

# Chapter 1

## Introduction

### 1.1 - Motivation of Thesis

Due to the recent availability of inexpensive LSI components, multi-microprocessors are now feasible in many applications. One of these applications is the processing and control of high-throughput data streams typically found in I/O paths of large computers. A few multi-microprocessors have been built for this type of application in the past, but questions of feasibility and performance have not been fully investigated yet.

Each new multi-microprocessor developed, along with its performance model, represents a significant experiment in evaluating the feasibility of alternative approaches. A useful multi-microprocessor experiment involves proposing an original hardware design and building a prototype to demonstrate its feasibility. The performance of the system must then be determined by modeling, by simulation, or by execution monitoring. A rough model for performance variations is essential for ascertaining both the range of applications for the system and the hardware requirements for a particular application.

## 1.2 - Thesis Project

This thesis describes and evaluates the design of a multi-microprocessor unit (MMU) flexible enough for use in a wide range of high-throughput applications. The prototype built is configured to monitor many real-time signal and data samples taken from an IBM mainframe computer.

The prototype application places certain constraints on the design of the MMU. First, the application requires the processing power of at least six 68000 microprocessors (1), chosen for the high power they deliver and for the convenience they provide in application software development. Next, the application requires a large shared memory for common data structures, along with local memories to reduce memory interference and increase throughput. Finally, provisions are necessary for the channelling of interrupts throughout the system.

The MMU design maintains generality while fulfilling the requirements of the prototype application. First, the design implements a flexible, modular architecture that is easily adaptable to different processing requirements. This increases the range of possible uses for the MMU and also allows the machine to change over the lifetime of a particular application. Second, the system components are kept as manageable as possible. Finally, the degradation of performance incurred by interprocessor communication overhead is minimized.

## 1.3 - Thesis Content

This thesis begins with a survey of related multi-processor projects undertaken in the past, including a discussion of how some of the techniques used in these past designs pertain to the goals of the MMU. A description of the system architecture along with its rationale follows, leading to a detailed description of all system components. Next, the performance of the MMU with various configurations and loads is analyzed. A simple method is proposed for estimating this performance. Finally, conclusions are drawn from the results of the experiment.

# Chapter 2

# Previous Work

## 2.1 - Introduction

Over the last decade, research has started in the area of using multi-microprocessing to improve cost-performance ratios in computers. Microprocessors have reached the performance levels reserved for much larger computers in the early 1970s and multiprocessing provides benefits such as potential modularity and reliability (1).

Recent research projects have included the design of both general-purpose computers and machines used in particular capacities within larger systems. Most of the documented projects on alternate architectures for multiple-instruction multiple-data stream (MIMD) multiprocessors is related to the work in this thesis. Although only a few successful, flexible multi-microprocessor systems have been built, they have contributed useful ideas about multi-microprocessor architectures. The architectures of some multi-minicomputers are also applicable. Some major design attempts in this area will be described in this chapter in chronological order.

## 2.2 Previous Designs of Similar Machines

### 2.2.1 C.MMP

The design of one of the first relevant multiprocessors built was started in 1971 at Carnegie-Mellon University (2). C.mmp, as the computer is called, was intended to experiment with multiprocessor issues as well as support a timeshared operating system.

C.mmp is designed to combine several slightly modified PDP-11 computers into a larger system. Each PDP-11 is able to run by itself if needed and contains private memory and I/O. Three attachments are added to each processor bus for connection to the C.mmp system.

The first attachment is a link to a central time counter, providing access to time stamps, and an interrupt unit, enabling each processor to send interrupts throughout the system. The second is an address mapping link to a crosspoint switch unit, providing access for each processor to any of sixteen shared memory modules. The third is another link to a crosspoint switch, providing access for each processor to several Unibuses containing I/O devices and secondary memory. Both crosspoint switches are controlled by one of the processors or by manual override.

The fast, pipelined crosspoint switch units in C.mmp illustrate one possible strategy for reducing memory contention in a multiprocessor system. The efficient design of such a unit is critical for the feasibility of a system like C.mmp.

2.2.2 Pluribus

Another early multiprocessor project, called Pluribus, was documented in 1973 at Bolt Beranek and Newman, Inc. (3). This machine was intended as a reliable message receiver for the ARPANET and not described as a general purpose computer.

The system consists of several buses, each contained in a separate chassis and supervised by an arbitration module. Seven processor buses are required, each one with two SUE microprocessors and a small amount of program memory. Two additional buses contain system memory and a final bus holds all necessary I/O modules. Bus transactions generally require more than one bus access and bus couplers are provided to link each processor bus with every non-processor bus. These bus couplers, consisting of two interface cards and a connecting cable, also link the I/O bus to the memory buses. The couplers provide for both segmented address mapping and protection against broken processors bringing down the whole system. The couplers between the I/O bus and the processor buses have bidirectional control for system initialization. Instead of providing facilities for interprocessor interrupts, Pluribus handles job allocation with a hardware job queue included on the I/O bus.

The design of Pluribus illustrates the modularity that can be obtained by dividing all system resources among expandable buses, and providing a mechanism for the addition of an arbitrary pattern of windows between them. Crosspoint connections between processor clusters and memory clusters can be produced with a net of cables instead of the separate crosspoint switch units used in C.mmp.

### 2.2.3 MCS

The MCS machine was designed at Honeywell in 1975 to experiment with multiprocessing in real-time control (4). Each MCS processor contains its own memory and I/O devices, allowing it to execute independent control functions. The machine also provides its processors with the ability to affect each other's memory, easing coordination between the various control functions.

Block transfers between processors and non-local memory pass over four global buses. Distinct paths are provided between each processor, its local bus, and its global bus interface. Global bus arbitration is implemented by providing each processor with a different time slot to send a block of data (up to 256 bytes long). I/O buffers are then provided at each interface to temporarily hold these data blocks. Any processor uses the first available global bus. A hardware semaphore device is included on the global buses to enforce mutual exclusion.

One goal of MCS is to simplify the writing of software by incorporating a complicated interface to the global buses to make message sending transparent. The utilization of four global buses illustrates another method to relieve bandwidth problems, as opposed to the last two examples.

### 2.2.4 HP Machine

A multiprocessor architecture was proposed at Hewlett-Packard in 1975 with a goal of facilitating VLSI implementation (5). The system is based

on one or more main buses that are compatible with processors, interrupt processors, and bus interface modules. Bus arbitration is implemented with a hierarchy of arbiter modules. Each module is designed to fit on one VLSI chip.

The main system bus contains thirty-eight signals, similar in operation to those on a PDP-11 Unibus. The signals are compatible with those of a proposed forty pin microprocessor. Microprocessors can be added to the bus until its bandwidth is fully utilized. All the devices placed on a single bus are controlled by a tree structure of arbiter modules that select the next bus master.

An interrupt processor module is provided as an interface between I/O requests and the microprocessors on a single bus. This module channels interrupts to particular processors if needed. It also has the highest priority on the bus when included in the system.

The final module, requiring sixty-six pins, is an interface between two buses. It provides a bidirectional window and simplifies system growth if inter-bus communication is kept at an acceptable level.

The HP system has a flexible multiple, uniform bus system similar to Pluribus, except for bus protocol and bus arbitration methods. The major goals of the HP system are to minimize the number of I/O signals on each module in the system, and to facilitate expandability.

### 2.2.5 Minerva

The Minerva Multi-microprocessor, developed at Stanford in 1976, was built to study the loading of a central bus in a multiprocessor system

(6). It contains two different kinds of processors, memory, and various other I/O devices, all connected through one main system bus. The main bus is asynchronous and arbitration is handled by a central arbiter. Interrupts are received by processors in response to accesses at fixed addresses on the bus. A fixed amount of local memory, not accessible from the bus, is included with each processor.

Since one of the processors has a thirty-two bit data path and the other has eight bits, a four byte prefetch buffer is incorporated into the CPU module of the eight bit processor. The thirty-two bit processor module contains a cache that monitors the main system bus to determine if its contents have become outdated.

The contribution of this project is experience with methods, such as the use of caches, for reducing bus bandwidth requirements in multiprocessors organized around a main system bus.

### 2.2.6 CM*

After completion of C.mmp, work started at Carnegie-Mellon in 1977 on a new multiprocessor called CM* (7). This new general- purpose machine consists of many LSI-11 processors, each having an associated portion of main memory on its local bus. Processors are able to access portions of main memory contained in other modules by sending packets over a network of interconnection buses forming a grid. Efficient operation depends on the locality of most memory accesses. Adequate buffering for packets in the switches of the interconnection network is necessary to prevent network deadlock.

This project experiments with the idea of distributing many processors and shared memory throughout a system and adding more overhead for a memory transaction as the distance between processor and memory increases. Such a system is expandable but is only efficient if memory references are kept as local as possible in software.

### 2.2.7 Lockheed Machine

In 1978 a multiprocessor based on the TI9900 microprocessor was designed at Lockheed to facilitate data processing at remote sites (8). The machine is built around one main system bus providing connections between processor modules, shared memory, and I/O devices.

The system consists of four processor modules, each of which contains a processor, some local memory, and an interface to a common bus. Internal to each processor module are three separate data paths: processor to local memory, processor to main bus, and main bus to local memory. Since decoding circuitry allows each local memory to be accessed directly from the main bus, programs are loaded from an I/O device placed on the main bus.

This multiprocessor project is another experiment using one main bus both for I/O and shared memory access. All synchronization between processors and synchronization with the I/O devices is accomplished through main memory, since no provisions are made for distributing interrupts throughout the system.

### 2.2.8 EPOS

In 1979, Toshiba Research Center reported work on a general-purpose multiprocessor called EPOS (9). Four homogeneous global buses provide the interconnection between processors in this system. I/O is controlled by special-purpose I/O processors attached to all the buses. Shared memory modules also interface with each bus. Bandwidth on the global buses is optimized by placing memory transactions into small synchronous time windows allocated after arbitration. Modularity is sacrificed to obtain better performance by the addition of a separate mutual exclusion module that is directly accessible to the microinstructions of each processor.

The use of four alternate paths into shared memory is the same bandwidth increasing technique used in the MCS. New features included special I/O processors on the global buses and fast semaphore operation.

### 2.2.9 AMP-1

During 1980 at the University of Illinois, a multiprocessor (AMP-1) was designed, based on a time division multiplexed global bus (10). The global bus interfaces with both processor and memory modules. Each of the eight processors is sequentially given a 125 nsec time window on the bus for streamed memory transactions. A small amount of local memory provides each processor space for local variables but the system is meant primarily for use with a high bandwidth, completely shared main memory. Extra interconnection circuitry, other than the main bus, is needed between the

processors and the memory modules to implement address mapping and timing for transaction streaming.

The AMP-1 project experimented with implementing a time division multiplexed global bus. With this strategy the speed of the bus must surpass that of each processor by the number of processors in the system.

### 2.2.10 Single User Multiprocessor

A multiprocessor for a single user was recently developed at the Australian Atomic Energy Commission (11). While its basic architecture is similar to the Lockheed machine, there are two major differences. First, each processor is able to protect a portion of its local memory from global access. Second, a separate interrupt bus is provided between the processors to allow the transfer of both interrupts and interrupt vectors. The hardware for this machine was developed with a programming language that supports multiprocessing in a single-user environment.

This project experiments mainly with adding hardware support for multiprocessor software. Memory protection capability is an example of such support. Also of interest is the choice of a special bus dedicated to sending interrupts.

### 2.2.11 iAPX432

Intel recently announced an iAPX432 microprocessor system capable of supporting multiprocessing (12). CPU chips in the system are added in arbitrary numbers, along with memory chips, to an interconnect bus.

Interface processors are also compatible with this bus and connect to I/O buses controlled by an 8086 attached I/O processor. The system supports capability based memory management and a high level language which is transparent to the number of processors on the interconnect bus.

While the iAPX432 system is designed for the construction of general-purpose computers with a high- overhead operating system, it illustrates that modular multiprocessing is now reaching the marketplace. It combines a flexible architecture similar to that of a system like Pluribus with hardware support for a general-purpose operating system.

## 2.3 Summary

The machines described in this section have contributed many useful concepts for use in multi-microprocessor designs. Issues pertaining to bandwidth problems of global system buses have been explored in some detail. Practical methods of connecting microprocessors to shared memory have also been examined. The implementation details of these machines have differed widely, though, and many other methods remain to be explored.

# Chapter 3

# System Overview

## 3.1 - Introduction

The MMU system designed in this thesis is aimed at special-purpose high-throughput applications. The architecture is described and compared to that of the machines presented in the previous chapter. Particular design decisions are explained and the prototype MMU serves as an example of system configuration.

## 3.2 - Overall Strategy

### 3.2.1 - Introduction

This section describes the main aspects of the MMU design strategy. All decisions are discussed in the context of the designs presented in the last chapter.

### 3.2.2 - System Memory

Multiprocessor systems must include some sharing of memory between processors to obtain close real-time synchronization. In general, this is accomplished by placing a switch between processors and memory that provides for all necessary data paths.

Since the MMU uses microprocessors as CPUs to reduce cost and complexity, the system must be limited to sequential microprocessor memory accesses. Only the addition of a cache or prefetch buffer on each processor would allow shared memory accesses to be pipelined through a switch as in C.mmp or CM*. The extra hardware needed for this is too large for a simple system like the MMU and the benefit is small. The special-purpose nature of the machine makes feasible the expectation that software can use some local processor memory as a non-transparent cache. As a result, each processor must complete one memory access (to either local or shared memory) before continuing with the next. This places the constraint on the shared memory switch that its throughput is inversely proportional to its total delay.

Many of the systems mentioned in the last chapter are designed as general-purpose computers. Therefore each processor needs access to the entire memory of the system, with varying degrees of overhead for different portions. In this way system jobs can be allocated among the processors in a general manner. The MMU does not need the overhead of this requirement as the local and shared memory in the system are dedicated for particular functions in its special-purpose application. As a result, a bidirectional network between processors and memory is unnecessary. This simplification over many of the previous designs allows the switch into shared memory to run faster than each local processor bus. One other system that dedicates switch accesses to a fast shared memory is the AMP-1. The streamed transactions in AMP-1, though, assume that all memory is shared and the switch can go fast enough to reach each processor once during each memory transaction. The MMU would not be able to meet

such a speed requirement, especially since the number of processors should be expandable. The MMU uses a switch with a speed that is raised to the highest reasonable level.

The shared memory switch specification proposed for the MMU involves creating a dynamically reconfigured switch that can operate from processors to shared memory in a fixed fraction of the time taken for one microprocessor memory cycle. The switch cannot connect one slow processor to another, because the response would be slower than that of the shared memory, but the switch can operate in reverse to allow certain I/O devices to have DMA access to each processor's local memory. The MMU switch must be similar to the ones used in Minerva, Pluribus, and the other shared bus machines, but the major difference is that a switch access can be made faster than internal processor accesses.

### 3.2.3 - System Organization

In terms of system organization, the MMU requires the most flexible design. The multiple-bus architectures used in systems such as Pluribus and the HP machine seem well suited for this purpose. For compactness, however, a grid of buses wired along a backplane is preferable to the cards and cables used in Pluribus. Connections between the intersections of these buses can be contained on one card and can implement the function of the shared memory switch. The resulting crossbar switch is a special case of the multiple-bus architecture and provides the necessary high-throughput connection for the MMU. The major difference between this multiple- bus architecture and the previous ones is that the shared memory

buses are slightly modified to obtain better speed. C.mmp uses a fixed crossbar switch to boost shared memory bandwidth, but one created from multiple buses is expandable in both directions.

The basic organization of an MMU system is pictured in figure one. A switch is formed by the crossing of the two bus types in the system, local and global. Each local bus is associated with one MMU processor and each global bus has a port on each local bus. A global bus contains devices shared by all processors. The number of local buses is adjusted to match the required processing power and the number of global buses is adjusted to match the required bandwidth to shared devices. The global buses do not provide a fixed number of alternate paths to shared devices as in MCS and EPOS, but instead are each associated with distinct shared devices. The overhead involved in providing multiple ports for shared devices is unnecessary in a system that can intentionally distribute accesses among global buses in its software. Instead of providing transparent address mapping, the MMU allows the addresses of all devices to be fixed at locations known to software.

The method of creating a flexible, high-throughput crossbar switch into shared memory proposed for the MMU is to form a grid of microprocessor buses wired on the backplane of the system. The shared memory buses are similar to but faster than the local processor buses, unlike in the general multiple bus systems. The simple technique used to help software distribute memory accesses among the global buses in the MMU is to allow interleaving of the address spaces of the global buses as seen by the processors. This method proves to be a useful technique as opposed to the MCS and EPOS method, at least for the special-purpose MMU.

### 3.2.4 - System Interrupt Handling

In those systems described in the last chapter that had provisions for processor interrupts, they were either handled by a special purpose processor or distributed to particular processors. Interprocessor interrupts could be obtained by a memory mapped access to a fixed location in the address space of another processor in the Minerva machine. The MMU uses a small portion of the bandwidth of a global bus to send interrupts as some other systems do, but greater flexibility is desired in the types and destinations of interrupts sent.

The method of distributing interrupts proposed for the MMU involves placing a special interrupt I/O device on a global bus. This device is programmable from any processor to send an interrupt with any vector to any combination of MMU processors. The sending of an interrupt is triggered either internally or externally to the MMU. The interrupts are buffered in the switch as they pass to the selected processors after being broadcast on the global bus.

## 3.3 - Architecture

### 3.3.1 - Introduction

The four basic logic modules in the MMU system include a CPU, Memory, Switch, and Interrupt Module. One CPU module is placed on each local bus, forming one MMU processor. The Memory modules are compatible with both bus types, forming the local and shared memories for the processors. The

Switch modules create the interface between each local and global bus. The Interrupt module sends interrupts along a global bus directed toward some or all MMU processors. Any MMU system contains these four basic modules along with I/O interfaces on either or both bus types. The I/O interfaces are application dependant. Any device placed on a local bus is dedicated to one processor while those placed on a global bus are shared by all processors.

The MMU combines some of the useful concepts tried in other machines with some unique features just described. The techniques used in the MMU are described in more detail in the following sections about the six basic components of the MMU system: the two bus types and four logic modules.

### 3.3.2 - Local Processor Bus

The local processor bus is a standard 68000 bus with sixteen bits of data and a one megabyte address space. All control signals follow the asynchronous protocol of the 68000 but their generation is expected to coincide with an edge of a universal clock common to all modules in the system. The 68000 controls bus arbitration through a daisy-chained grant signal reaching all switch modules and I/O devices on the local bus. All seven 68000 interrupt levels can be generated on the bus from any device but convention reserves one level for each device. Therefore the interrupt acknowledge signal is not daisy- chained.

The three types of bus transactions allowed on the local bus by a bus master are a memory write, a memory read, and an interrupt acknowledge. Any slave device can be addressed for the memory transactions. Figure two

gives a rough timing diagram for each of these operations. Further explanation is available in the 68000 users manual (13) and timing details are explained in the next chapter. All cycles are extended if the acknowledge signal is not received by the bus master before a certain point in the transaction.

### 3.3.3 - Global Bus

A global MMU bus is similar to a local bus in most respects, but has a few significant differences. Arbitration and memory transactions are synchronous, interrupt transactions follow a different protocol, and a bus transaction exists to logically connect the bus to one of the local buses through a Switch module.

Memory read and write transactions, as well as interrupt transactions, are allocated two global bus clock cycles as shown in figure three. No acknowledge signal is required from a slave device. Daisy-chained arbitration for bus ownership in each transaction occurs during the last clock cycle of the previous transaction. Since bandwidth is a bottleneck only on a global bus, the memory transactions are not constrained to be as slow as 68000 transactions. The signals and protocols on the two buses, however, are similar enough for a single memory module to be compatible with each. The MMU system combines the convenience of a universal bus type with the high throughput of fast global bus cycles.

During an interrupt transaction, an interrupt is broadcast to one or many of the Switch modules on the global bus. The bus interrupt lines

select a buffer in the Switch module(s) to hold the interrupt until its CPU responds. The highest order address bits are used to indicate the interrupt level, while the lower order data byte is used to indicate the interrupt vector. The high order data byte is used to mask which Switch modules will receive the interrupt.

The global bus also contains an additional signal allowing an I/O device to force one of the Switch modules to become a logically transparent port to its local bus. Once local bus arbitration is completed after this signal is asserted, the I/O device acts as if it were directly on the selected local bus and has direct access to the processor address space. This type of DMA transaction is not used by one processor to access another because Switch modules can only generate fast global bus transactions and not slow local bus ones.

### 3.3.4 - CPU Module

The CPU module controls the arbitration for a local processor bus and is the usual bus master. As bus master, it executes 68000 instructions. The complete CPU module is logically similar to the 68000 chip but contains some additional devices. Parity is generated for the data as it leaves the module and a programmable interrupt timer is included along with some diagnostic registers. Devices internal to the CPU module are not accessible to any local bus master other than the 68000. Appendix A1 contains a block diagram of the module.

### 3.3.5 - Memory Module

The Memory module is a slave device on an MMU bus providing a window of thirty-two or sixty-four kilobytes of read-write memory. The address window of the module is positioned in the bus address space by switches. Parity bits for the data are also stored in this module. A block diagram for the Memory is located in appendix B1. Depending on location within the MMU system, a Memory module serves as part of local or part of shared memory.

### 3.3.6 - Switch Module

The Switch module provides the interface between a local bus and a global bus. Besides being a bidirectional bus port, the Switch also contains interrupt buffers that hold interrupts for the processors until they are acknowledged. Appendix C1 contains a block diagram of the module.

As a global bus port, the Switch channels memory transactions at certain addresses on the local bus to the global bus. The address window on the local bus is variable, and can even be limited to odd or even word addresses. This allows the processor addresses of two global buses to be interleaved for even distribution of accesses between them. For a processor access to a global bus, the Switch requests the bus and returns an acknowledge after it is obtained. If there is no global bus arbitration delay, a processor write completes in the minimum five clock cycles while a processor read is extended from four to six clock cycles.

At least one cycle is required for global bus arbitration and two are required to carry out the transaction on the global bus.

As a local bus port, the Switch logically connects all local bus signals to the corresponding global bus ones as described in the section outlining operation of the global bus. This function is triggered by a special slave request global bus transaction.

An interrupt buffer in the Switch is loaded during global bus interrupt transactions and holds a processor interrupt until an acknowledge is received on the local bus. Each buffer stores an interrupt level and the interrupt vector to be returned during the processor acknowledge. If an interrupt arrives in a buffer before the previous one has been acknowledged, the new interrupt is lost and a pulse is generated for diagnostic monitoring.

### 3.3.7 - Interrupt Module

The Interrupt module is a global bus device dedicated to generating interrupt transactions on the bus. The module also supervises global bus arbitration. The interrupt vectors and masks for the interrupt transactions are stored in registers accessible to the global bus address space. The transmission of interrupts is triggered either by a global bus memory transaction at a particular address or an external signal. A block diagram of the module is included in appendix D1.

## 3.4 - Design Considerations

### 3.4.1 - Introduction

This section discusses the factors involved in some significant MMU design decisions. The explanations of smaller design decisions are contained in the next chapter along with the detailed description of the hardware. Advantages of the overall architecture are explained along with the architecture description.

### 3.4.2 - Degree of Global Bus Interleaving

The overhead for increasing the potential degree of global bus interleaving is substantial. An increase from the two-way interleaving implemented in the MMU to a four-way interleave requires an extra level of address multiplexing. As well as introducing extra hardware in the Switch module, this adds extra delay to an address path critical in some DMA operations.

Some degree of interleaving, though, is very useful. If only one global bus is used for shared memory, a block transfer into shared memory through DMA will cause severe performance degradation by locking out processors accessing shared memory. A second global bus will only eliminate this problem if it is interleaved to distribute all block accesses across both buses. Otherwise, one or the other of the buses will be locked out and any processor accessing it could stop for a long time.

As a result, two-way interleaving is used in the MMU system. When more than two banks of shared memory are used in the system, the software must distribute concurrent data accesses among non-interleaved blocks to improve performance. Distributing shared accesses among the global buses is no longer transparent to the software.

### 3.4.3 - Global Bus Arbitration Mechanism

Many algorithms exist for bus arbitration including static priority, dynamic priority, fixed time slice, and first-come first-served. Static priority is usually the easiest to implement, often with a daisy-chained bus grant signal, but does not distribute bus accesses fairly among processors (14). In a general purpose computer, this unfair distribution is considered undesirable because most users and jobs need equal treatment. In a special-purpose machine such as the MMU, however, the static priority is often useful because software can allocate the most critical functions to the highest priority processors. As a result, the MMU uses static priority for all its buses.

### 3.4.4 - Semaphore Operation

The indivisible test and set instruction is a general and convenient way to enforce process synchronization for critical events. In a multiprocessor environment, these instructions must not permit a memory location to be accessed between the read cycle and write cycle of their execution. The 68000 microprocessor provides a special test and set

instruction that holds the processor bus by not removing its address strobe between the cycles. This special instruction is not flexible and does not signal the Switch module that it is a semaphore operation until after the Switch has already released the global bus in the read cycle. As a result, the MMU uses an otherwise unused high order address bit to signal mutual exclusion in shared memory to the Switch. Any 68000 instruction with a read and write cycle can be used on a semaphore by offsetting its operand address. When the mutual exclusion bit is detected, the Switch keeps possession of its global bus between the read and write transactions, or until a timeout occurs in the event of an error. Therefore global bus memory transactions occasionally hold the bus longer than two clock cycles before another master is selected.

## 3.5 - Prototype Configuration as an Example

As mentioned previously, the prototype MMU is configured to control the monitoring of real-time hardware and software samples from a large computer. Six processors are used in this application to provide the necessary processing power.

All prototype MMU modules are implemented on separate boards with MC68000 chips along with SSI and MSI TTL integrated circuits. Each processor is contained in a separate chassis, one on top of the next. One final chassis on the top is dedicated to holding all shared hardware in the system. All MMU buses are wired on the chassis backplanes.

The prototype provides one I/O device on each local processor bus connecting the processors to a system controller. The controller is a

computer used to initialize the system as well as collect the results of the sampling. These I/O devices allow the controller to access each processor's address space, to interrupt each processor, and to reset each processor.

The prototype MMU uses three global buses, two interleaved ones for shared memory and one dedicated to a device obtaining the software samples. The global bus dedicated to the software sampler is used primarily for control of the device and for use by the device to load blocks of samples into MMU memory through DMA. The devices collecting hardware samples for the system are connected through circuits added to the Interrupt modules on each of the shared memory buses. In addition to providing MMU processor access to shared memory, these buses allow MMU processor access to the hardware sampler and provide a path for interrupts generated by the hardware sampler to enter the system. The interrupt traffic on the shared memory buses generated by the Interrupt modules uses only a small fraction of their bandwidth.

The prototype machine illustrates the method for constructing a particular MMU configuration. The number of processor chassis included was matched to the expected amount of processing necessary. The bandwidth to shared hardware required was distributed equally among the number of global buses required to avoid shared hardware bottlenecks. Estimates for choosing these numbers are provided in the chapter covering MMU performance measurement. Neither decision is fixed throughout the life of the machine if original estimates were incorrect.

# Chapter 4

# Detailed Hardware Description

## 4.1 - Introduction

The feasibility of the MMU architecture is demonstrated by the operation of the prototype machine. The details of implementation, as described in this chapter, involve no major unforeseen difficulties. The size and complexity of the system fall within reasonable limits.

The following six sections provide detailed descriptions of the six basic MMU building blocks. All descriptions use the prototype implementation as an example.

## 4.2 - Local Processor Bus

The local processor bus consists of fifty-nine active-low signals, most of which are on open-collector lines. Most of these are logically identical to 68000 pin signals as described in the MC68000 users manual (13). Only the system clock and the bus arbitration daisy-chain are not bidirectional, open collector lines with terminating resistors. Figure four illustrates the electrical schematic for most of the bidirectional bus lines. All compatible modules use AMD bus driver/receiver chips (26S10-12) to interface with the bus lines. In the prototype, all signals except the bus arbitration daisy-chain are wired across each card position on a chassis backplane. The daisy-chain signals jump from one card to the next, originating at the CPU side of the processor. Since the prototype

bus is less than two feet in length, signals are delayed at most by about four nanoseconds while crossing it.

### 4.2.1 - Data Signals

The data portion of the bus contains sixteen lines (MDAT0-15) which transmit two bytes in parallel. In addition, each byte is provided with a single parity line (MDATPH and MDATPL). Separate parity is provided for each byte to enable 8-bit 68000 data transfers. The 68000 includes this ability to maintain compatibility with 8-bit devices still on the market. All data lines can be driven and received by any local bus device.

### 4.2.2 - Address Signals

The local processor bus contains twenty address lines (MADR1-20), nineteen of which are used to create a one megabyte local address space. The most significant address bit is asserted during a read-modify-write instruction to maintain exclusion as described earlier. A switch module will maintain possession of its global bus between the read and write transaction only if the most significant address bit is being asserted. The address lines can be driven by any local bus master.

### 4.2.3 - Control Signals

The local bus contains all the 68000 control signals, except the function code bits which are decoded into one signal which indicates when

an interrupt acknowledge operation is occurring on the bus (MIACK). A separate interrupt line is provided for each of the seven 68000 interrupt levels (MINT1-7). Protocol for interrupts and interrupt acknowledges are identical to those described in the 68000 users manual. A bus error signal (MBERR) is also provided to signal errors during local bus transactions.

Bidirectional reset and halt lines (MRESET and MHALT) allow the processor to be initialized or stopped, while also allowing the processor to reset external circuitry and indicate when it has halted itself. The system clock (MCLK) is driven onto the local bus and received by all Switch modules, Memory modules, and by the CPU module. Clock synchronization with the rest of the system is handled internally to each of these modules.

Four bus signals are provided for arbitration of local bus control. A bus request line (MBRQ) is provided for either a Switch module or an I/O device to signal the 68000 that it would like to take control of the bus. When the bus is available, the 68000 sends a bus grant down the arbitration daisy-chain which is comprised of two signals at each card, grant-in (MBGRIN) and grant-out (MBGROUT). The final signal is bus acknowledge (MBACK), which is asserted by the new bus master for the duration of its local bus transaction.

The final five control signals are used for memory transactions on the bus. An address strobe (MAS) and two data strobes (MUDS and MLDS), one for each data byte, are used to confirm or request stable addresses and data. A read line (MRD) specifies the direction of the memory transfer and an acknowledge line (MDTACK) indicates that the transaction is being

completed. Detailed timing diagrams for both reads and writes on the local bus can be found in figure five.

## 4.3 - Shared Processor Bus

### 4.3.1 - Bus Signals

The global bus is pin compatible with the local bus and contains many of the same signals with slightly different uses. It is electrically identical but can incur more delay due to its long length in a system with a large number of processors.

The address and data portions (CADR1-23, CDAT0-15, CDATPH, and CDATPL) are identical to those on the local bus, except for the addition of three more high-order address lines. These address signals are not used to expand global bus address space, but rather to identify which of the eight possible processors is the source of any global bus transaction. The three extra address lines are also used to send interrupt levels during interrupt bus transactions and to select a processor during a DMA operation. Address bits nineteen and twenty are unused in the prototype and always remain inactive. They are available for expansion beyond eight processors.

A memory inhibit line (CINH) replaces the interrupt acknowledge line, but these act in an identical way on the memory modules. The memory inhibit signal prevents all slave devices on the bus from responding to a memory transaction. This is used, for example, during a DMA access into

one of the processors when all memory on the global bus must be disabled to prevent it from conflicting with the local bus address space.

The clock line on the global bus (CCLK) is only wired along the shared hardware portion of the bus. The Switch modules get their clock signals from their local bus. In the prototype MMU, the system clock is distributed through the I/O devices on each bus. The reset line (CRESET) is wired along all shared hardware on every global bus and to all I/O devices. As a result, any I/O device in the MMU prototype can generate a global reset.

The halt (CHALT), bus error (CBERR) and data acknowledge (CDTACK) signals are not used in the prototype global bus. An acknowledge signal is generated by memory modules but it is ignored on the global bus. Since the prototype global bus is not longer than about ten feet, the maximum possible delay for signals travelling along it is about twenty nanoseconds.

### 4.3.2 - Bus Arbitration

Synchronous bus arbitration occurs on the global bus in one clock cycle. On each negative edge of the system clock, all requesting devices latch their requests (CBREQ) and a bus grant signal is sent down the daisy-chain (CBGRIN and CBGROUT) if the bus is available. The first requesting device to receive a grant returns an acknowledge (CBACK) which must be returned before the next negative clock edge to suppress the next arbitration cycle. The acknowledging device becomes bus owner at this next clock edge and immediately removes its acknowledge if it is executing

a two cycle bus transaction. This allows arbitration for the next owner to overlap with the second clock cycle of the transaction. The timing for this operation is illustrated in figure six.

### 4.3.3 - Bus Transactions

The bus transactions used on a global bus fall into three categories. Memory transactions allow a bus master to read data from or write data to a slave device. Interrupt transactions allow an Interrupt module to distribute interrupts to Switch modules. Also, DMA transactions allow bus masters to access a local processor bus through one of the Switch modules. All three transactions are illustrated in figure six and are explained in the following sections.

#### 4.3.3.1 - Memory Accesses

The two clock cycle memory read and write transactions follow the protocol illustrated in figure six. During both operations, the address is sent after the first negative clock edge while the address strobe (CAS) and one or both of the data strobes (CUDS and CLDS) are asserted after the next positive edge. The read-write signal (CRD) is asserted along with the address. During a write operation the master sends data along with the address and the slave device must latch the data before the second negative clock edge. During a read operation, the slave must return data to the master slightly before the second negative edge so that the master

is able to latch it on that edge. In both cases each data strobe controls a separate data byte.

4.3.3.2 - Interrupt Transactions

A two clock cycle interrupt transaction begins with the bus master asserting address and data signals after the first negative clock edge. The high order data byte contains a one in each bit position corresponding to a processor (zero-seven) that will receive the interrupt. The low order data byte contains the interrupt vector used during the interrupt acknowledge and the most significant address lines indicate the level of the interrupt (one to seven). After the second negative clock edge, one of the global bus interrupt lines (CINT1 and CINT2) is asserted to strobe the interrupt information into one of the interrupt buffers on the selected Switch modules. All signals are removed after the final negative edge of the transaction.

4.3.3.3 - DMA Transactions

A DMA transaction is initiated by an I/O device on the global bus for gaining access to a local processor bus. The number of the selected processor is first asserted on the three most significant address lines and then the slave request line (CSRQ) is asserted. This signal is not used on a local bus. The slave request causes the selected Switch module to obtain control of its local bus and pass memory transactions through. The slave request must remain asserted during the entire DMA operation and

the I/O device must issue local bus memory transactions. It need not, however, run as slowly as the 68000 when accessing a fast local bus device. A DMA write to local memory takes at least two clock cycles and a DMA write to shared memory takes at least four clock cycles. Either may be extended due to bus arbitration delays. The memory inhibit line must also be asserted during DMA transactions to suppress the response of other global bus devices to the memory transactions transmitted to the local bus.

## 4.4 - CPU Module

The detailed schematics for the CPU module logic can be found in appendix A. The logic has been split up into twelve pages, A2-A13, each of which is described separately in this section. The 68000 chip is split up among many of the pages because of its large size and diverse functions. A block diagram of the entire CPU module can be found in appendix A1.

The CPU module is wired on a board compatible with the local processor chassis. The board contains sixty-three chips and approximately 550 wires.

### 4.4.1 - Terminating Resistors

Schematic A2 shows all the local bus signals connected to the split termination chips B2, B3, B9, and B10. These lines have the electrical configuration illustrated in figure four. The interrupt lines on the

local bus are connected to 4.3K ohm pullup resistors in chip D11. Chip E15 contains resistors used to tie inputs high throughout the board.

### 4.4.2 - Address Drivers

Schematic A3 follows the twenty least significant address lines of the 68000 from the microprocessor to the local bus. Since the 68000 outputs are not capable of driving the local bus signals, five 26S10 driver chips (A6-A10) are used to buffer the signals. The open collector drivers are disabled when the 68000 gives the local bus to another bus master, as indicated by the BGACK signal. The four least significant 68000 address bits are also used to address local CPU modules in schematics A6, A9, and A13. No path is provided to enable the local bus address bits onto the internal address lines because other devices on the local bus are not provided with the ability to access devices internal to the CPU module.

### 4.4.3 - Internal Data Bus

Schematic A4 illustrates the connection between the data pins of the 68000 and the internal CPU data bus. The 68000 has tri-state data outputs so no extra buffers are needed for interface with the tri-state internal data bus. Also pictured are the 68000 output signals VMA and E, used to access a 6800 compatible device in schematic A6.

### 4.4.4 - Upper Data Path

Schematic A5 contains the interface between the local data bus and the internal CPU data bus for the most significant data byte and the two data parity bits. Drivers A1-A3 are used to send data from the internal bus to the local bus when enabled by the signal DOUT. Receivers B4 and B5 enable data onto the internal tri-state bus when DIN is asserted.

### 4.4.5 - Lower Data Path and Programmable Timer

Schematic A6 contains the buffers between the lower byte of the internal and local data buses, as well as a programmable timer chip. The data bus buffers operate just as the ones for the upper data byte, with chips A4 and A5 driving the local bus and chip B6 driving the internal bus. The programmable timer (PTM) is an internal CPU device which only has eight data bits because it is a 6800 compatible device. The BE signal is provided in the 68000 to simulate 6800 memory cycles for such devices along with the 68000 derived BWR signal. Address decoding to be described later causes the CPU to generate BVMA for certain memory locations reserved for 6800 compatible devices. The four least significant address bits are used to address the PTM control registers. RST will reset the PTM during an internal CPU reset and PTMINT triggers 68000 interrupts on schematic A13.

### 4.4.6 - General Purpose Registers

Schematic A7 illustrates the two eight bit general purpose registers contained in the CPU module as internal devices. These registers can not be accessed from the local processor bus, but only by the 68000 itself. Register B14 latches its eight bits from the high order byte of the internal data bus while register B15 is connected to the low order byte. Tri-state buffers C10 and C11 are used to enable the outputs of both registers back onto the internal data bus when the 68000 reads their contents. Both registers are accessed through internal device location one, as reflected by the one appearing in all of their control signals. A positive edge on WH1 or WL1 will load the respective registers and the assertion of RD1 will enable the contents of the registers onto the internal data bus for a 68000 read. An internal CPU reset will blank the contents of both registers. The outputs of the registers can be connected to the backplane through unused edge pins for distribution to any desired location, such as display lights.

### 4.4.7 - Counters

Schematic A8 contains two eight bit counters also provided on the CPU module as internal devices. The counter chips (A12-A15) are connected to the internal data bus just as the general purpose registers are, and can be read through chips B12 and B13. Since the counters are writeable, they can be used as another set of general purpose registers. By connecting external signals to the clock inputs the counters can be used to monitor

the frequency of external events. This is a very useful tool for system debugging. The carry outputs are also brought out to the backplane so that the counters can be cascaded. Together the counters are accessed as internal device two, as indicated by the two appearing in the control signals WH2, WL2, and RD2.

### 4.4.8 - Address Decode

Schematic A9 contains the address decoding circuits for internal CPU devices. Since the lower twenty address bits of the 68000 map directly to the local bus address space, the three most significant 68000 address bits are left to select internal CPU devices. Address bit 23 is used to select internal 6800 compatible devices while address bit 22 is used to select all other internal devices. Gates A and B of chip E5 allow these devices to be selected during any memory cycle except an interrupt acknowledge, in which the address lines are always high. Gate B of chip E6 generates an internal 68000 acknowledge as soon as a memory cycle begins for a non-6800 internal device because all internal devices are fast enough to respond within the shortest possible memory cycle.

Gate A of chip E6 enables data from the internal data bus onto the local bus whenever the 68000 has not released control of the local bus and when the 68000 is executing a write memory cycle. Data is enabled onto the local bus during the loading of internal devices but it is ignored. Gate A of chip D9 only enables data from the local bus onto the internal bus during memory read cycles in which internal CPU devices are not

selected. In this case a bus conflict with the internal devices must be avoided.

Gate B of chip D9 disables the 68000 control signals from reaching the local bus during an internal reset, an access to an internal device, or a period when the 68000 is not the local bus master. When the control signals are disabled, they remain high and do not trigger memory cycles on the local bus. Gates A and B of chip D7 are necessary for fanout requirements.

Decoder C6 enables the reading of both non-6800 compatible internal devices. Decoders C7 and C8 enable the loading of both internal devices, one controlling the high order data byte and the other controlling the low one. BUDS and BLDS are strobe signals originating from the 68000 which can separately select each byte of the data bus.

### 4.4.9 - Control Drivers and Reset Logic

Schematic A10 contains the circuits driving local bus control signals along with a bidirectional reset buffer. Driver chips B1 and B7 send the 68000 memory control signals onto the local bus when enabled by COUT. These controls include the address strobe, the data strobes, the read/write signal, and the interrupt acknowledge signal. The interrupt acknowledge signal is produced by chip C3 which decodes level seven from the 68000 function code bits. Gates A and B of chip E11 produce strobes for each data byte during memory accesses from the local bus. These strobes are used to latch error conditions in the parity arriving from the

local bus. Gates C and D of chip D7 act as buffers for the bus grant signal being sent to the local bus from the 68000.

The flip-flops of chip D14 are used for synchronous arbitration of the bidirectional reset signal of the 68000. Latch A will send a reset to the 68000 while latch B will send one out to the local bus through driver B8. Latch A is only set if a new reset has arrived from the local bus, as indicated by latch D, and the 68000 is not generating a new reset, as indicated by latch C. Latch B is set with the opposite conditions. This arbitration allows resets to flow in both directions while insuring against the possibility of a deadlocked situation in which the resets cannot be withdrawn.

### 4.4.10 - Control Logic

Schematic A11 contains various control circuits external to the 68000 on the CPU module. Gate A of chip A11 receives the system clock signal from the local bus for use in the CPU module. Chip B11 then is used to synchronize the internal BCLK signal with the BCLK signals on all other modules in the MMU system. The range of five to twenty-five nanoseconds is enough to compensate for clock skew that is encountered in distribution. Gates A and B of chip C9 produce both phases of the clock signal for the module. The 68000 microprocessor uses the inverted phase of the clock so that it is clocked slightly ahead of the system clock.

The bus request and bus grant acknowledge inputs to the 68000 come directly from the local bus signals while the memory cycle acknowledge (DTACK) is derived through gate A of chip D6 either from the local bus

acknowledge or one generated for internal devices. The bus error condition in the 68000 is triggered through gate A of chip E8 either by a parity error in one of the data bytes, a bus error signal asserted on the local bus, or a device response timeout generated by counter E12. Chips E12 and E13 form an eight bit counter which is enabled to count by gate C of chip E7 when a 68000 memory access has begun and no acknowledge has been received. If this condition remains for 256 clock cycles, it is assumed that the accessed device is not responding correctly and a bus error is generated.

A halt request to the 68000 is generated by gate B of chip D5 when a halt is asserted on the local bus and the 68000 is not in the middle of a read-modify-write transaction. The 68000 receives the local bus halt signal as soon as it is finished with such a transaction. If the 68000 generates a halt, this is not passed on to the local bus. Since the reset is bidirectional, it requires both gate A of D5 to pass a reset to the 68000 and gate C of chip D15 to send a reset out if it is initiated from the 68000.

The VPA input to the 68000 is asserted during a memory cycle if a 6800 compatible cycle is desired and during an interrupt acknowledge if an autovector is desired. Thus gate B of chip E7 generates a VPA either during an internal 6800 compatible device access or during the acknowledge of an interrupt from the programmable timer.

### 4.4.11 - Parity Generator and Checker

Schematic A12 contains the CPU circuits dedicated to generating and checking parity for MMU data. Chips C4 and C5 act both as parity generators and checkers for the two data bytes transferred over the local bus. When data is being sent out to the local bus, PARINH and PARINL are tri-stated and the resistors in chip D11 are used to pull them high. Chips C4 and C5 then generate odd parity to be sent out to the local bus along with the data bytes. When data is being received, latches A and B of chip E10 receive a zero at the end of the transaction if their respective data byte did not have odd parity. Latches A and B of chip E9 then generate a bus error during the next transaction if there was an error and reset the error condition.

### 4.4.12 - Interrupt Encoder

Schematic A13 illustrates the 68000 interrupt generating circuits in the CPU module. Chip D3 is used to encode the interrupt lines on the local bus into the three signals which the 68000 uses as inputs. Chip D4 can assert one of these local bus interrupts if enabled by the programmable timer interrupt. The level of the timer interrupt is determined by the positions of the first three DIP switches at chip location D13. An open switch corresponds to a logic one due to the pullup resistors in chip D12. Chip D8 generates an acknowledge for a PTM interrupt if the 68000 acknowledges an interrupt on a level which matches the one reserved for the PTM.

4.4.13 - Critical Timing Paths

The timing of the CPU module is described relative to the timing diagrams for local bus transactions contained in figure five. All the delays are adjusted for efficient interface between the CPU module and the rest of the MMU system.

A valid address leaves the CPU board within 22 nsec of the negative edge of the clock that starts state two of a 68000 bus cycle. The system clock is being used for reference and not the inverted 68000 clock. The IACK line is valid within 16 nsec later. The address and data strobes enter the local bus within 16 nsec of the positive edge first occurring after they are generated. The RD line, which is the inverse of the 68000 WRITE signal, takes up to 6 nsec longer than the strobes. When data is leaving the CPU board the 16 data bits enter the local bus within 22 nsec of the negative edge of the clock that starts state four of a 68000 bus cycle, but the parity bits take 23 nsec longer.

When data is being returned to the 68000 over the local bus, it must arrive at the board at least 30 nsec before the positive edge of the system clock that will latch it (the end of state six in a normal 68000 read cycle without delay). DTACK must arrive 28 nsec before the positive edge of the system clock at which it is to be recognized. This would be at the end of state four in a normal fast read and the end of state six in a normal fast write. Any other asynchronous inputs (BERR, BGACK, BR, VPA, and IPL0-IPL2) must arrive at the 68000 chip at least 20 nsec before a positive edge of the system clock if they are to be detected then.

## 4.5 - Memory Module

The detailed schematics for the Memory module can be found in appendix B. The repetitive memory chips are pictured only once, with the positions of all the actual memory chips listed in tables on the schematics. The schematics are divided up into 8 pages, B2-B9, and are described in turn in this section. Appendix B1 contains a block diagram of the entire Memory module. The prototype Memory board contains seventy-five chips and approximately 1300 wires.

### 4.5.1 - Internal Address Bus

Schematic B2 illustrates how the least significant address lines from the backplane bus (local or global) are received by the memory module. The receiver chips distribute these address signals to all the memory chips in the module. The ten least significant bits feed the data storage chips and must be faster than the higher-order bits. The higher-order bits are only used on the parity storage chips which are not in the critical timing path. Therefore fast Schottky gates (A10-A12) are only used on the bottom ten. The rest are received by Schmitt trigger low-power Schottky inverters (chip A8).

### 4.5.2 - Upper Data Path

Schematic B3 contains the interface between the backplane data bus and the internal memory data bus for the twelve most significant bits. Drivers A3-A5 are used to send data from the internal bus to the backplane bus when enabled by the signal DOUT. Receivers B4 and B5 enable data onto the internal tri-state bus when DIN is asserted.

### 4.5.3 - Lower Data Path and Control

Schematic B4 contains the buffers between the lower four bits of the internal and backplane data buses, along with the parity bits and the receivers for some bus control signals. Chips A2 and A6 drive data signals onto the external bus and chip B6 receives them exactly as is done with the more significant bits in schematic B3. Chip A9 contains the inverters used to receive some of the control signals on the external bus and gate A of chip A8 receives the clock. Chip B8 is used to synchronize the internal clock signal with the rest of the modules in the system and both phases of the clock are distributed throughout the board by gate B of chip B7 and gate A of chip A9.

### 4.5.4 - Address Window Select

Schematic B5 contains the circuitry in the memory module that determines which bus addresses the module will respond to. Comparator A7 enables a response when bus address bits fifteen to eighteen match the

four values selected by switches S3 to S6. Gate A of chip B7 inhibits this enable when address bit nineteen is a logic one or the memory inhibit line is asserted. As a result, the memory can be placed in any 32K byte block in the bottom 512K of the bus address space.

When the module contains 32K bytes of memory, S1 must be open and S2 must be closed. Gate C of chip A12 then distributes a logic one to the latch input of all the 1K byte memory chips. When the module contains 64K bytes of memory, S1 must be closed and S2 must be open. Gate C of chip A12 then distributes address bit 15 to the 2K byte memory chips. When S1 is closed only three address bits are compared, enabling the module to respond to a 64K byte block of the bus address space. S3 must be open when S1 is closed to prevent address line fifteen on the bus from being pulled low.

Gate A of chip A13 enables the module to send a bus acknowledge through chip A2 only if the module has been selected for a memory transaction. Gate C of chip C6 enables the output data drivers during a bus read of the module.

### 4.5.5 - Address Decode

Schematic B6 illustrates how the enable signals for each memory chip in the memory module are generated. Decoders B9 and B10 create the enable signals for each of the sixteen pairs of byte-wide data storage chips. Address bits eleven to fourteen are used to make this selection. Decoder B11 creates the enable signals for the eight 4K by one bit high-order parity memory chips and decoder B12 creates those for the eight low-order

ones. When the memory module is configured for operation as a 32K byte memory, only the top half of the parity enables are generated.

### 4.5.6 - Data Storage

Schematic B7 illustrates how one pair of byte-wide static RAMs is connected to the internal address and data buses. While 1K byte chips are shown, the only difference in the 2K byte chips is that the latch input is replaced by another address bit. The tables at the top of the schematic list the chip locations of the high-byte and low-byte RAM chips associated with each of the sixteen enable signals.

### 4.5.7 - Parity Storage

Schematic B8 shows one of the eight pairs of 4K bit memory chips in the memory module. The tables at the top of the schematic list the chip locations of the high and low parity chips associated with each of the eight pairs of enables.

### 4.5.8 - Timing Circuits

Schematic B9 contains the two flip-flops that synchronize the memory module control signals. Gates A and B of chip C6 detect one of the data strobes being asserted on the external bus and flip-flop A latches this on the next falling edge of the clock. This flip-flop generates the acknowledge signal that is returned on the bus if the memory module is

selected. Gates A and B of chip C5 generate the write pulses sent to upper and lower byte memory chips during the half clock cycle after flip-flop A is set. Gate delays insure that the gates of chip C5 will be disabled before the flip-flops are cleared. Gate D of chip C6 enables data onto the internal data bus during memory cycles in which a write is occurring. Chip B15 contains the resistors used for pullups throughout the module.

### 4.5.9 - Critical Timing

The logic delays in the Memory module are adjusted for efficient operation on either a local or global MMU bus. Since the bandwidth of the global bus is a critical system bottleneck, the memory speed is greater than that needed on the local bus to allow a faster global bus cycle.

BSEL is generated within 18 nsec of the arrival of the last input to chip A7 or B7. This includes address lines fifteen to nineteen and either MIACK or CINH, depending on whether the module is on a local or global bus. Any input needs up to 12 nsec to affect the decoder outputs so the strobes must arrive at least 30 nsec after the address and IACK do to avoid glitches on the chip selects. The RD signal must arrive no later than AS so that the chip select will initiate the correct cycle.

During a read, the memory can respond within 120 nsec of getting an address or within 60 nsec of getting a chip select, whichever is longer. Adding in buffer delays, this means that data will leave the module within 148 nsec of the receiving of an address or within 86 nsec of the receiving of an address strobe, whichever is longer.

During a write, the address must arrive 30 nsec before the AS as before. The data strobe or strobes must arrive at least 18 nsec before a negative edge of the clock as should AS. The inverse of the read signal must arrive 27 nsec before this edge. Data must arrive with or before the data strobes and stay until 44 nsec after the next positive edge of the clock. The address need only remain valid until 7 nsec after the same positive edge. Both reads and writes take at least the two clock cycles used on the global bus. The timing is also compatible with the 68000 bus cycles as a DTACK signal leaves within 24 nsec of the first negative clock edge after the data strobes are detected.

## 4.6 - Switch Module

The detailed schematics for the Switch module can be found in appendix C. The schematics have been divided up into twelve pages, C2-C13, described in turn in this section. Appendix C1 contains a block diagram of the entire Switch module. The prototype Switch board contains ninety chips and approximately 1000 wires.

### 4.6.1 - Local Bus Window

Schematic C2 illustrates the selection of an address window on the local bus for the Switch module. Switches S1, S3, S5, and S7 select the use of address bits one, sixteen, seventeen, and eighteen respectively to determine the module window on the local bus. A logic one, corresponding to an open switch, enables the selected address bit to be sent to

comparator D6 along with its desired value, as determined by switches S2, S4, S6, and S8 respectively. Enabling fewer address bits to the comparator through chips C5 and C6 increases the size of the address window that the Switch responds to. Switches S2, S4, S6, and S8 will match a logic one on an address bit if they are closed. Switch S9 positions the window in the top 512K bytes of local bus address space if open or the bottom 512K bytes if closed.

Gates A, B, and C of chip C7 enable all unselected address bits onto the global bus so that the full address window can be mapped to the bottom of global bus address space. The selection of address bit one generates an INTRLEV signal that shifts all address bits down one location while passing to the global bus.

### 4.6.2 - Upper Address Path

Schematic C4 contains the path of the most significant address bits between the local and global buses. Chips A5 and A6 serve as local bus tranceivers while chips A13-A15 serve as global bus tranceivers. Multiplexers C14 and C15 shift the address bits passing from the local to global bus down one position each if the INTRLEV signal is asserted. The signal CADR enables addresses to pass from the local to global bus and MADR enables addresses to pass in the opposite direction.

Address bits twenty-one to twenty-three on the global bus are derived from the Switch module identification bits (CPID0-CPID2) when sent to the bus. When received from the global bus, these address bits are used to carry an interrupt level (BCADR21-BCADR23).

4.6.3 - Lower Address Path

Schematic C4 contains the inter-bus address path for the least significant address bits. All chips function as extensions of the previous schematic. Chips A7-A9 are tranceivers for the local bus and chips A16-A18 are tranceivers for the global bus. Chips C17-C19 are the multiplexers that can shift the least significant address bits when the interleave option is selected for the Switch.

4.6.4 - Upper Data Path

Schematic C5 contains the inter-bus data path for the most significant data byte, as well as a Switch mask detector. Chips A1 and A2 are local bus tranceivers and A10 and B11 are global bus tranceivers. Chips C1 and C2 are used to latch data passing from the global to the local bus when the signal LDAT is removed. The signal CDAT enables data to pass onto the global bus and the signal MDAT enables data to pass onto the local bus. Chip E2 selects the bit in a received high order byte which corresponds to the ID bits of the Switch. This determines whether the Switch is being selected during a global bus interrupt transaction.

4.6.5 - Lower Data Path

Schematic C6 contains the inter-bus data path for the least significant data byte along with both data parity bits. All chips function the same as the corresponding chips in schematic C5. Chips B1,

A3, and A4 are local bus tranceivers while chips B10, A11, and A12 are global bus tranceivers. Chips B2, C3, and C4 latch data passing from the global bus to the local bus.

4.6.6 - Interrupt Buffers

Schematics C7 and C8 contain the logic for the two interrupt buffers in each Switch module. Since only the gate and chip locations differ between the two circuits, only the first will be described.

Gate B of chip D15 generates an active low pulse when the corresponding Switch and buffer are selected during a global bus interrupt transaction. This pulse (BCINT1) is used by gate A of chip E9 to generate a pulse that loads a new interrupt vector into chip D11 if a previous interrupt is not pending. Flip-flop A of chip E4 is set to indicate a pending interrupt through gate A of chip E5. When an interrupt is pending, chip D1 asserts the appropriate interrupt line on the local bus to get the attention of the processor. The BCINT pulse also enables a new interrupt vector to be latched in chips C10 and C11 when a previous interrupt is not pending.

When the CPU is acknowledging an interrupt at the level contained in the buffer as detected by comparator D9 and gate A of chip E8, an ENVEC signal is generated. The ENVEC signal enables the buffer vector onto the internal vector bus and sets flip-flop A of chip E3. This flip-flop is then used to reset the interrupt flip-flop through gate C of chip E6 when the acknowledge cycle has completed.

### 4.6.7 - Interrupt and Global Bus Hold Circuits

Schematic C9 contains interrupt circuits common to both buffers as well as logic controlling the read-modify-write capability of the Switch and the clock drivers for the module. Gate D of chip E7 detects when either of the buffers is acknowledging an interrupt and enables drivers B3 and B4 to place the contents of the internal vector bus onto the local data bus for the CPU. Gate C of chip E8 then generates an acknowledge to be sent to the CPU. An acknowledge can also be generated by two other sources on the module, DTACK1 and DTACK2.

Gate A of chip B9 receives the clock signal from the local bus and F9 synchronizes it with the clocks on all other modules in the system. Gates A and B of chip F10 then generate both phases of the clock to be used throughout the module. The negative phase of the clock is slightly ahead of the positive one to speed up the generation of some signals in the Switch.

Flip-flop A of chip F11 is used to latch the read-modify-write condition in the Switch. When the global bus is taken for a read operation as indicated by gate A of chip F7, and the lockout address bit (twenty), is asserted, gate B of chip F13 sets the flip-flop. The signal CHOLD is then generated and the Switch keeps possession of the global bus until the write operation occurs. The write operation is detected by gate C of chip F13 and it resets the flip-flop. The flip-flop can also be reset by a timeout counter F12 or a local bus reset through gate C of chip F8. The counter begins counting when gate D of chip D13 detects the completion

of the read operation. Eight clock cycles are allowed before a write operation is expected.

4.6.8 - Inter-bus Control Path

Schematic C10 contains the drivers for the control signals on both the local and global bus. Drivers B5 and B6 are permanently enabled to send control signals to the local bus and driver B15 is enabled to send control signals to the global bus. Driver B7 is enabled to send signals to the local bus when MSTB is asserted and driver B16 is enabled to send strobes to the global bus when LDAT/ and DTACK1 are asserted. The address and data strobes can be enabled in either direction between the local and global buses depending on the enabling signals. The bus read line is similar except the local to global direction is controlled by CADR through gate A of chip E7. DTACK is sent from the local bus to the global bus through gate B of chip E7 and only enabled when the Switch is the local bus master. BREQ and BACK are generated for the global bus in the global bus arbitration circuit in schematic C11. DTACK is generated for the local bus in Switch control circuits on schematic C9 while BREQ and BACK are generated for the local bus in schematic C12. Chip B6 is also used to receive IACK, DTACK, and BACK from the local bus. Chip B5 receives the local bus reset for the Switch and gate A of chip F15 generates a local bus BERR either in the case of a global bus BERR or a read-modify-write timeout on the Switch. Gate A of chip F5 generates one of the sources of a DTACK signal for the local bus when a write operation is under way and

CADR is enabling signals onto the global bus. This synchronizes the 68000 write cycle with the corresponding global bus transaction.

4.6.9 - Global Bus Arbitration and Control

Schematic C11 contains the circuits that control global bus arbitration and memory transactions. Flip-flops A and B of chips E12 and E13 control the timing of a two clock cycle global bus memory transaction. The first is set at the negative edge marking the start of physical ownership of the bus by the Switch. The second is set at the next positive edge, the third is set at the second positive edge, and the fourth is set at the final negative edge.

Flip-flop B of chip E11 is set when it is determined that the local bus is requesting a memory transaction on the global bus. Gate B of chip E15 detects a valid request and gate A of chip F13 suppresses the setting of the request flip-flop only when possession of the bus has just been granted, as indicated by gate A of chip F8, or a global bus transaction has already been triggered, as indicated by gate D of chip D15. Gate B of chip B8 passes the bus grant daisy-chain along if the Switch module is not making a request during a particular arbitration cycle. Normally gate D of chip B8 generates a global bus acknowledge only during the clock cycle that the Switch receives a bus grant but a CHOLD signal from the read-modify-write logic extends the acknowledge and therefore global bus possession.

The first of the global bus timing flip-flops is set immediately after global bus possession is obtained. After the bus transaction is

completed, the flip-flops are reset one by one through gate A of chip D13 unless possession of the bus is being held. The absence of an address strobe can also reset the first flip-flop through gate A of chip D15 if bus possession is not being held. In either case, the final three flip-flops are reset when the data strobes are removed as detected by gate C of chip B8. The address strobe cannot be used for this because the 68000 contains a special test and set instruction which does not remove the address strobe between its read and write transactions. Gates E and F of chip E5 insure that the final flip-flop will be the last to be reset and glitches will not be produced on the global bus during transactions from another Switch module. The first flip-flop enables an address to be sent to the global bus by generating CADR and also enables data during a read operation through gate B of chip F7. The second flip-flop generates a DTACK for a read operation to synchronize the transaction with the 68000 memory cycle. This flip-flop also enables the global bus strobes. The last flip-flop latches data passing from the global bus to the local bus with the LDAT signal and also disables the global bus strobes.

4.6.10 - Local Bus Arbitration and Control

Schematic C12 contains the circuits that control local bus arbitration. Before SRQ is asserted on the global bus, or during a local bus reset, gate C of chip E7 resets all of the arbitration flip-flops. When SRQ is asserted and the three most significant global bus address lines match the Switch ID number as detected by comparator D8, gate D of chip F5 causes the request flip-flop, part A of chip E10, to be set. Gate

A of chip B8 only passes along the bus grant daisy-chain if no local bus request is pending. Gate A of chip D14 detects when all the conditions are met for obtaining ownership of the bus and triggers the setting of the acknowledge flip-flop, part B of chip E10. The acknowledge flip-flop resets the request flip-flop and returns a bus acknowledge to the CPU. Flip-flop A of chip E11 enables the global bus strobes to drive the local bus strobes at the next positive clock edge after the bus is obtained. Gate C of chip F7 detects the condition of the global bus writing data to the local bus, and gate A of chip E15 detects the condition of the local bus reading data from the global bus. Gate A of chip E14 allows either condition to enable the data drivers onto the local bus.

### 4.6.11 - Terminating Resistors

Schematic C13 shows all the global bus signals connected to the split termination chips B2, B3, B9, and B10. These resistor chips are only inserted in modules placed at one end of a global bus. The global bus signals with terminating resistors are electrically identical to their counterparts on the local bus. Chip D3 contains resistors used to tie inputs high throughout the module.

### 4.6.12 - Critical Timing

The logic delays in the Switch are critical for efficient interprocessor communication. The delays in all paths are tuned to mesh with the other modules in the system to require a minimum number of clock

cycles for each transaction. The timing constraints of global bus transactions are met and the delay involved in a local bus access to the global bus is kept small in the case of the bus being free.

During a shared memory access by a processor, the PSEL signal is generated within 60 nsec of the time that the address arrives from the local bus. By then the address has long since arrived at the global bus output buffers and is waiting to be enabled when the bus is next free. The local bus address strobe must arrive at least 45 nsec before a negative edge of the system clock to start global bus arbitration at that edge. The address must arrive 90 nsec before the same edge to initiate a cycle in the correct Switch module. IACK must arrive before the address strobe. A local bus acknowledge leaves within 32 nsec of the first negative edge of a global bus transaction during a write and within 27 nsec of the following positive edge during a read. The acknowledges are timed to cause the 68000 memory cycles to extend just long enough to encompass the global bus cycles. During a read operation, the data must arrive from the global bus at least 35 nsec before the final edge to get latched and held for the CPU.

During an interrupt acknowledge by the CPU to one of the Switch buffers, an acknowledge is returned within 74 nsec of the address strobe. The address must have arrived slightly before the address strobe. The interrupt vector is returned within 37 more nsec in time to be latched by the 68000. The interrupt acknowledge runs at top 68000 speed.

During a slave request transaction on the global bus, the SRQ signal must arrive at the Switch at least 45 nsec before a negative edge of the clock. The Switch ID bits (address 21-23) must have stabilized 56 nsec

before the same clock edge. Once the Switch has obtained control of the local bus, the global bus appears the same as the local bus except for some added signal delay. Addresses take 30 nsec to cross the Switch while data takes 30 nsec passing to the global bus and 57 nsec passing to the local bus.

During an interrupt transaction on the global bus, an INT signal must arrive at the Switch 53 nsec before the final negative clock edge in the global bus cycle. The interrupt level must be stable at the Switch 35 nsec before that. The interrupt vector need only arrive 32 nsec before the final negative edge but the interrupt mask requires 59 more nsec for decoding by the selected Switches.

## 4.7 - Interrupt and I/O Module

The detailed schematics for the Interrupt and I/O module (IMOD) can be found in appendix D. The schematics are divided up into ten pages, D2-D11, described in turn in this section. Appendix D1 contains a block diagram of the entire IMOD. The prototype IMOD board contains fifty-six chips and approximately 600 wires.

### 4.7.1 - Global Bus Window

Schematics D2 and D3 contain the address decoding circuits for the IMOD that detect an access from the global bus. Switches S1-S12 select a 128 byte window on the global bus address space. A closed switch corresponds to a logic one on each corresponding address bit. Comparators

before the same clock edge. Once the Switch has obtained control of the local bus, the global bus appears the same as the local bus except for some added signal delay. Addresses take 30 nsec to cross the Switch while data takes 30 nsec passing to the global bus and 57 nsec passing to the local bus.

During an interrupt transaction on the global bus, an INT signal must arrive at the Switch 53 nsec before the final negative clock edge in the global bus cycle. The interrupt level must be stable at the Switch 35 nsec before that. The interrupt vector need only arrive 32 nsec before the final negative edge but the interrupt mask requires 59 more nsec for decoding by the selected Switches.

## 4.7 - Interrupt and I/O Module

The detailed schematics for the Interrupt and I/O module (IMOD) can be found in appendix D. The schematics are divided up into ten pages, D2-D11, described in turn in this section. Appendix D1 contains a block diagram of the entire IMOD. The prototype IMOD board contains fifty-six chips and approximately 600 wires.

### 4.7.1 - Global Bus Window

Schematics D2 and D3 contain the address decoding circuits for the IMOD that detect an access from the global bus. Switches S1-S12 select a 128 byte window on the global bus address space. A closed switch corresponds to a logic one on each corresponding address bit. Comparators

A17-A19 detect an address which matches the specified high order address bits along with gate B of chip B17. Gate A of chip B17 disables the selection of the IMOD if address bit 19 or the memory inhibit line are asserted.

Gate A of chip E7 generates a pulse during an access to the top sixty-four bytes of the IMOD address space. This pulse is used to trigger the sending of an interrupt from buffer zero. Since any processor can trigger this buffer by accessing global bus address space, it is used for interprocessor interrupts.

Gate A of chip C9 signals a write operation to any of the bottom sixty-four bytes of the IMOD address space. This signals a write to one of the vector or mask registers in the interrupt buffers. Gate B of chip C9 and gate C of chip B17 signal a read operation to one of these registers. Gate C of chip D8 then enables the IMOD to send data to the global bus either during such a read or during the sending of an interrupt to the Switches.

### 4.7.2 - Data Bus Tranceivers

Schematic D4 contains the tranceivers on the IMOD for the data lines of the global bus. Chips A11-A14 and D18 receive the global data bus onto the internal input bus and drive the internal output bus onto the global bus. The signal DOUT enables all the drivers.

### 4.7.3 - Interrupt Vector and Mask Registers

Schematics D5 and D6 contain the memory chips used for the interrupt buffer control registers. Schematic D5 contains those used for the interrupt vectors on the low data byte and schematic D6 contains those used for the interrupt masks on the high data byte. In each case a table at the top provides the chip locations for the corresponding chips in each interrupt buffer, as only one set is pictured. For example chips D13, D14, and E18 serve to store the sixteen interrupt vectors for buffer zero. Each register is stored along with parity. Only one interrupt vector and interrupt mask can be sent from an interrupt buffer at once.

### 4.7.4 - Internal Address Bus

Schematic D7 pictures the four bit internal address bus that selects which of the sixteen interrupt types in each buffer is being accessed. Receiver A15 normally drives the tri-state bus with the four least significant bits of the global address bus for global bus memory transactions to the interrupt buffer registers. During and interrupt transaction, though, chip D15 selects the interrupt type if buffer zero is being used and chip A9 selects the type if buffer one is being used. Chip D15 is loaded with the four least significant data bits during a write that triggers an interprocessor interrupt and chip A9 is loaded with four external bits during an externally triggered interrupt from buffer one.

Driver A10 asserts one of the global bus interrupt lines, depending on which buffer is being used, during the second clock cycle of an interrupt

transaction. ITRANA is asserted during both clock cycles of the transaction and ITRANB is asserted only during the second.

4.7.5 - Global Bus Control

Schematic D8 contains the control circuits for the IMOD. Flip-flop A of chip C6 stores a pending interrupt from buffer zero and flip-flop B stores a pending interrupt from buffer one. If one of the interrupts is pending, gate B of chip D8 loads a request for the global bus into flip-flop B of chip D9. This request latches a selection of which buffer is being serviced into flip-flop B of the same chip. Buffer zero is given priority if both have pending interrupts. A request also inhibits further bus grants through gate A of chip D8 and gate A of chip E12. Gate B of chip E12 detects when the previous bus owner is finished and the interrupt transaction can begin.

An interrupt transaction starts with the setting of flip-flop C in chip C10. Flip-flop D of the same chip is set at the start of the second cycle of the interrupt transaction. ITRANA enables a new bus request to be sent from flip-flop A of chip C10 during the second cycle of the transaction to overlap arbitration for the next global bus owner. Gates A and B of chips C7 and D7 clear a pending interrupt either during a system reset or after the interrupt has been sent.

Bus grant is normally sent to the global bus except during an interrupt transaction and during the clock cycle following a bus acknowledge. Flip-flop B of chip C10 stores the acknowledge to indicate

that another module is in any but the last clock cycle of a global bus transaction.

### 4.7.6 - Memory Timing Circuits

Schematic D9 contains the IMOD clock drivers, the memory timing circuits, and some further address decoding. The clock is received by gate C of chip B11 and synchronized with the rest of the system by the delay element in location B9. Gates A and B of chip B10 generate both phases of the clock for distribution inside the IMOD. The negative phase is slightly ahead of the positive phase for pre-clocking of some critical signals.

The two flip-flops in chip D10 are used to generate write pulses for the interrupt buffer registers. Gate A of chip E10 produces the pulse for the low byte and gate B of the same chip produces the pulse for the high byte. Gate A of chip E8 causes the flip-flops to enable the pulse during the half clock cycle following the first negative clock edge after the data strobes arrive. Gates C and D of chip E9 enable one or both of the write pulses depending on whether the memory transaction is for a byte or a word. Gate C of chip C7 and gates A and B of chip E9 clear the write flip-flops after the transaction has completed and the gates producing the write pulses have been disabled.

The gates in chip E11 are used to select one of the buffers both during control register accesses and interrupt transactions. Gate A is used during an interrupt transaction to assert SEL0 only if the arbitration flip-flop has selected buffer zero. Gate B enables a global

bus access to buffer zero registers only if the bus address is in the bottom thirty-two bytes of the IMOD address space. The next thirty-two bytes of address space are reserved for buffer one.

### 4.7.7 - Interrupt Level Generator

Schematic D10 contains the generator for the interrupt level sent on the high order address lines during an interrupt transaction. The levels used for each interrupt buffer are set by switches S13-S18. An open switch corresponds to a one in the corresponding bit of the level. The output of the buffer arbitration flip-flop is used to select a level through multiplexer C18. Driver B18 then places the level on the global bus address bits during an interrupt transaction.

### 4.7.8 - Terminating Resistors

Schematic D11 contains all the global bus signals connected to split termination chips B12, B13, B15, and B15. These resistor chips are only inserted in an IMOD if it is placed at one end of a global bus and are identical to those on the Switch modules. Chip E13 contains resistors used to tie inputs high throughout the IMOD.

### 4.7.9 - Critical Timing

The logic delays in the IMOD are adjusted to meet the requirements of the fast, synchronous global bus transactions. Accesses to the interrupt buffer registers require no more than the allotted two clock cycles and interrupt transactions are also transmitted to the Switch modules within this time limit.

During a normal global bus arbitration cycle, bus grant is sent within 17 nsec of the negative clock edge and the acknowledge must arrive 14 nsec before the next negative edge. The six processors on the prototype bus meet this requirement easily but the size cannot be expanded much without changing the arbitration circuits.

During an interrupt transaction on the global bus, the address (interrupt level) is valid within 51 nsec of the first clock edge. The data (vector number and selection mask) is valid on the bus before the second negative clock edge and one of the INT lines is lowered within 35 nsec after that second negative edge. This allows the selected Switch modules to strobe in a valid interrupt by the final clock edge of the transaction.

# Chapter 5

# Performance Measurement

## 5.1 - Introduction

The performance of the MMU was measured by two methods: a software timing simulator and a trial run of test programs. These provide a general method for measuring the performance of any MMU type machine. The accuracy of both is confirmed by a comparison of their results. An analytical model is developed that is shown to provide a simple method for marginal performance estimation. The important results are summarized first, followed by details of the experiments.

## 5.2 - Measurement Results and Marginal Performance Model

For the purposes of this experiment, equivalent performance is defined as the sum of the performance of each processor. These performances are defined as the fraction of time which a processor is running its program as opposed to waiting for a turn on a global access to shared hardware. For example if six processors were running and each was spending about one third of its time waiting for shared memory access, the MMU would have an equivalent performance of four.

The performance of the MMU is a function of the number of global buses in the system, the number of processors running, and both the frequency and distribution of shared hardware accesses coming from each processor. Each of these variables was adjusted across an interesting range to

illustrate its effect on performance. The results are summarized in graphs one and two of appendix E. Graph one represents an MMU configuration with only one global shared memory bus while graph two represents one with two interleaved global buses. In each graph the horizontal axis indicates the number of processors in the system and varies from one to ten. The vertical axis indicates the equivalent performance measured in each configuration. The four curves graphed in each case represent four of the different program types run on the processors during the experiments. L0 corresponds to a program that would require possession of a global bus 1.5 percent of the time if running alone. L1 to L4 correspond to global bus loadings of 11.4, 19.0, 29.8, and 39.5 percent respectively.

One important factor in the decision of the number of processors feasible for a given configuration and application is the point where the extra performance gained by adding an additional processor is not worth the cost. Of course minimum throughput requirements and how well the application tasks can be partitioned among many processors are also factors. Since the marginal performance is a function of the number of processors, the load from those processors, and the load expected from the additional processor, the number of variables in the system is unwieldy.

If the system seen by the marginal processor can be represented by a black box characterized by one variable, the problem becomes more manageable. The variable chosen in this thesis is the fraction of available cycles in which the black box makes new bus requests that block the marginal processor. This fraction is easy to measure and is also the

probability that a random request from the marginal processor will be blocked.

The model contains two key approximations that have been tested for validity. The first is that a marginal processor will have a negligible effect on the performance and behavior of the rest of the system. Since it has lowest priority, this is not completely unreasonable. The second involves the distribution of bus requests from the system. By assuming that the blockage probability is independent from cycle to cycle, the performance degradation seen in a marginal processor running a given load can be calculated. This approximation also assumes that requests derived from many different processors running different programs are random in nature. The fact that requests are delayed until the bus is free will tend to cluster bus requests, but the total effect of this on marginal performance turns out to be small in the situations measured.

The model proved to be a fairly accurate predictor of marginal performance under many different circumstances. Graphs five and six picture the request probability (black box variable) measured under all the conditions of system operation measured in graphs one and two. Graphs seven to eleven picture the performance degradation calculated for a marginal processor running each of the five sample load programs (L0-L4) when added to systems characterized by various request probabilities. In all cases measured in graphs one and two, the approximation for marginal performance of an additional processor was close to that observed.

Another variable related to the request probability is bus utilization. Graphs three and four indicate how this measure of the fraction of global bus cycles used varied over the same experiments.

Because of the random nature of shared memory interference, marginal performance starts to fall off before the global bus bandwidth is fully utilized.

## 5.3 - Performance Measurement Tools

### 5.3.1 - MMU Test Programs

The test programs written for bringing up the MMU include two major functions. First, they send test patterns of data to all the memory modules and check for errors in retrieving them. Second, they record their execution time to monitor their performance under different system loads. This combination of memory testing with performance measurement produces benefits for each function.

Performance measurement serves as an additional hardware diagnostic check. Unexpected deviations between the results of the test routines and performance estimations indicate hardware timing problems. One timing bug in an MMU I/O device was actually found through the performance measurement test alone.

The memory testing package helps to verify the validity of the performance measurement. For example, if the data patterns are sent to local instead of global memory, the results of the performance measurement are severely affected. The memory testing guards against such a mistake, while at the same time confirming the reliability of the path to the global memory space where the system clock resides. The system clock is used by the test programs to measure their execution time.

The major difficulty encountered in combining these two functions is distributing memory accesses in a random manner. Memory test routines typically involve tight loops that step through memory and produce a very regular pattern of accesses. This is incompatible with performance measurement because these routines may synchronize when running concurrently and reflect extreme deviations from the contention expected under normal conditions. To avoid this problem, a subset of a special 68000 assembler was developed that can expand loops into longer ones with varying numbers of null instructions inserted between the original instructions. The random pattern of null instructions only repeats after each block of 1000 original instructions. As reported in other experiments (15), the distribution of memory accesses has little effect on memory interference levels in multiprocessor systems as long as it is fairly random. Therefore, the inserted null instructions are adequate to simulate interference in application programs.

All simulations and MMU performance measurement programs make two additional simplifying approximations when estimating performance. The first assumption is that a very small portion of the global bus bandwidth will be devoted to sending interrupts in most applications. Since the 68000 microprocessor takes many clock cycles to jump to an interrupt handler while only two global bus cycles are needed to send each interrupt, the processors would not be capable of handling a significant amount of interrupt traffic efficiently. For this reason the global bus interrupt traffic is ignored. The second approximation has to do with the length of bus transactions. While a standard memory access takes two clock cycles of bus ownership, a read-modify-write access will take at

least eight. These special read-modify-write instructions, though, are envisioned to be used primarily for semaphore operators and should only occur upon entry into critical sections of code. As a result, they should not make up a significant portion of shared memory accesses and all transactions are assumed to be two cycles in length.

The random assembler generated the MMU test programs requiring global bus bandwidth at each of the levels (L0-L4) described earlier. L3 and L4 had to be placed in shared memory to achieve their level of bus usage. Also, these programs running at the highest possible level of bus loading had to tolerate a large amount of regularity in compacting shared memory requests. L4 uses a series of block write instructions to approach forty percent global bus usage and contains no null instructions. The 68000 only uses two of its five clock cycles in a write instruction to complete its global bus transaction. L0, L1, and L2, however, maintain a fairly high degree of randomness. These programs, built from typical 68000 instructions but with accesses distributed randomly, should closely reflect the distributions of requests found in typical application programs.

An additional piece of hardware is provided in the MMU system for use with the test programs. Since the test programs can not generate high levels of bus contention in a system with a small number of processors, an alternative method must exist to generate this load. In this way the performance of a test program on one processor can be measured over a wide range of interference levels. Therefore, a dummy load board exists that generates global bus requests in a random manner at an adjustable level. Once in possession of the bus, the dummy load does nothing but block other

users for two clock cycles. Request probabilities between successive cycles are independent and adjustable in 1/16 increments up to 15/16. The hardware design of the dummy load is described in appendix F.

Performance measurements were taken by running combinations of the five different program types on the processors available at the time of the experiments. The execution times of the programs were compared in all cases to their stand-alone execution times and a level of performance degradation was calculated. Measurements were also taken with one of the programs running in competition with the dummy load hardware. In each case the dummy load was adjusted through all sixteen of its possible interference levels. The results obtained are compared to those of the timing simulator and the analytic model as presented later.

### 5.3.2 - Timing Simulator

This section describes the software timing simulator used to estimate the performance effects of global bus contention in the MMU system. Such simulations are simple in microprocessor systems involving sequential memory requests without separate request queues. As explained in chapter three, a shared memory access requires at least five processor clock cycles, including one used for global bus arbitration and two used for the global bus transaction. If the processor is unable to obtain the bus during its arbitration cycle, it extends the length of the bus operation and continues requesting the bus until it is available. Arbitration for the next user of a global bus is overlapped with the second cycle of the previous transaction.

In simulating the bus arbitration, a distinction is made between physical ownership and logical ownership of the bus. A processor has physical ownership during the cycles it enables data and address signals onto the bus. Logical ownership starts in the cycle that the processor gets selected as a bus master and ends before the cycle when the next master is selected. The simulations and models of the MMU arbitration deal with logical ownership for convenience. See figure seven for an illustration of bus arbitration and ownership. The figure also distinguishes between arbitration cycles, in which a new master can be selected, and closed cycles, in which the bus is not ready for a new master.

The input of the software simulator takes the form of a bit vector for each processor running in the system during the simulation. Each bit vector is derived from the instruction stream that the corresponding processor executes and it describes the pattern of its accesses to the global bus address space. Each element in the bit vector indicates whether the instruction stream will request a global bus during a given cycle. A write into shared memory takes five clock cycles in the stand-alone case, so a write instruction in the stream would contribute five elements, one of which would be the actual bus request cycle. In the case where more than one global bus is involved, the vectors must add another dimension to specify each bus separately.

In running the simulation, the simulator steps through clock cycles one after another and delays all instruction streams by one cycle if they must wait at a given point. After one instruction stream is completely executed, the simulator stops and compares the total delay of each stream

with the total execution time. In this way a percentage of performance degradation can be calculated for each processor. The simulator also keeps track of bus utilization and frequency of requests during the execution period.

Input streams for the simulator can be produced which closely represent the request distributions characteristic of all five bus loads that have been produced by the random assembler. These only differ from the actual test programs by the small overhead of loop control and timing computation that exists only in the test programs. The random number generator was run independently for each version of the programs. Input streams can not be produced which reflect the request pattern of the dummy bus load because the dummy load can not queue requests until the bus is free. If the dummy load makes a request which is ignored, there is no greater chance that it will make a request on the next cycle. Therefore, direct software simulations of the dummy load were not made.

### 5.3.3 - Comparison of Measurement Tools

The results of both the performance measurement programs and the software simulator were compared to verify that both were working correctly. Two MMU processors and two global buses were available so that all the basic functions of the system could be tested.

The five memory test and performance measurement programs were run in all possible combinations on both MMU processors with only one global bus. As expected, there was a high degree of synchronization when the high bus loading programs were used, but the rest produced results in which the

degradation was generally a monotonically increasing function of bus loading.

All of these experiments which were run on the MMU were also run on the software simulator. Agreement was very close in most cases and well within the range of random fluctuations. As a result it appears likely that both operate very close to correctly. Graphs twelve to fifteen illustrate both sets of data for comparison. Processor zero refers to the highest priority 68000 while processor one refers to the next one down the priority chain. In all cases the performance degradation on one processor due to competition from the other is measured. L0-L4 refer to the five different programs used with L4 being the greatest global bus user. The six curves pictured in graphs twelve and fourteen represent three different programs (L0-L2) run on both the simulator and the MMU. The curves in graphs thirteen and fifteen represent three different levels of loading (L0-L2) for the competing program. In all cases the MMU and simulator results remain in pairs, illustrating the agreement between the results of both. The non-linearity of the curves represents synchronization effects in some program combinations.

## 5.4 - Marginal Performance Model Development

### 5.4.1 - Analytic Model of Marginal Processor Performance

Many analytic models have been developed for multi-processor systems such as C.mmp (16,17). None of these, however, seem to be useful for

systems such as the MMU with microprocessor memory cycles and local memory for instructions.

The analytic model developed for the MMU deals with the simple case of one processor contending with the variable dummy load. This has the potential for being able to estimate the performance of a processor in conflict with a wide range of interference levels, without unnecessary complication. The dummy load represents the approximate behavior of the black box system seen by a marginal processor. The request probability of the dummy load is taken to match that of the black box it is representing, as indicated in figure eight.

This section calculates the degradation of performance which should be seen in a processor running a program characterized by a request frequency of L, when it is placed below a dummy load with an independent conflicting request probability of P for each arbitration cycle. This is exactly the situation which exists when a program is run on an MMU processor in conflict with the dummy load hardware. First, the reciprocal of L indicates the number of cycles per processor request and each request is extended by an average amount of E wait cycles. The degradation D is then E/(1/L) or EL. The problem then becomes one of finding the expected wait time E.

The expected value of time in which the bus is free for the processor after one transaction is over is 0P+1P(1-P)+2P((1-P)*2)+ . . . or (1-P)/P. Since the chance the bus will be blocked is two cycles per transaction divided by those two cycles added to the expected time which the bus is free, the chance that the bus is in use is 2P/(1+P). The chance that the bus is free is then 1-(2P/(1+P)) or (1-P)/(1+P). Since the bus

spends equal time in both halves of a transaction, this means that there is a chance of P/(1+P) that a request will hit either half. As a result, the expected wait time for the processor can be computed as (0(1-P)/(1+P))+(P/(P+1))((1(1-P)+3(1-P)P+5(1-P)P*2+ . . .)+ (2(1-P)+4(1-P)P+6(1-P)P*2+ . . .) or (P/(1+P))(3+4P/(1-P)). By substitution into the equation derived in the last paragraph, the processor running in conflict with the dummy load will experience performance degradation of (LP/(1+P))(3+4P/(1-P)).

The analytic model was compared to measurements taken on the MMU with the dummy load hardware. The dummy load can be adjusted to have a request probability P which ranges from 0 to 15/16 in increments of 1/16. The dummy load was placed above processor zero in priority and all five measurement programs were run on processor zero using one global bus. This was repeated with the dummy load set at each possible request level.

The results for each measurement program are illustrated in graph sixteen and graphs seven to eleven show a comparison of each curve with data points calculated from the analytic model. It is clear that the match was almost perfect, increasing one's confidence in both the dummy load hardware and the calculations used in the model. If system global bus requests can be approximated by a dummy load with a given request probability, the additional performance obtained from an additional processor in the system can be calculated if its instruction mix is known.

### 5.4.2 - Testing of Dummy Load Approximation

This section describes the testing of the two approximations made when using the dummy load (black box) model for calculating marginal performance.

The first of these approximations comes from the fact that the addition of a processor to the system will subtract some total performance from the higher priority processors. This effect can only be ignored if the extra performance obtained from the additional processor is always much greater than that lost over all of the rest.

The second approximation concerns the expected value of wait time used in the calculations of the last section. Even if all the processors in the load are generating requests with a random distribution, the fact that they can queue requests will make the request probability P dependent on the history of bus utilization. In the calculations of E, though, it was assumed that P was independent from cycle to cycle. Therefore the actual expected value in some cases could vary significantly from that used in the calculations.

Across the entire range of data, the first approximation was tested by calculating the ratio of performance lost in original system processors upon addition of another processor, to the performance added in the additional processor. All of the points obtained are shown in graph seventeen and since most of the fractions are insignificant, the very worst being only about 30 percent, the first approximation appears reasonable. If an additional processor was estimated to add the equivalent of half a processor to the system (100 percent degradation),

even the extreme case of a thirty percent error would mean that the processor might only add the equivalent of a third of a processor due to degradation of performance in the original processors. Since the average error is only a few percent, the one case of thirty percent is also likely to be the result of statistical variations in the simulation programs.

Graphs eighteen to twenty-one show a comparison of the performance degradation seen in each additional processor of different loads with the calculated estimate. The request probabilities used were taken from the data in graph five. The fact that all of these graphs show close results indicates that the second approximation appears to be reasonable.

At least over the simulated ranges of MMU operation, the performance which can be obtained by an additional processor can be reasonably estimated by using graphs five and six and the equations derived in the last section. Graphs eleven and twelve are interesting in that they clearly show a rough number of processors which saturate the global bus bandwidth under different program loads. Beyond a certain point extra processors do not increase performance at all. This point occurs roughly at the point where the total global bus bandwidth equals the sum of the bandwidth requirements of all the system processors. Under certain combinations of program loads, it is even conceivable that overall performance could be decreased beyond this point.

# Chapter 6

# Conclusions

## 6.1 - Summary of Experiment and Results

The experiment described in this thesis involved the designing and building of a general multi-microprocessor machine (MMU) tailored for a particular high-throughput application. Performance of the machine was measured in different configurations and a simple method for estimating the performance of this type of machine was verified.

The MMU machine utilizes the flexibility and bandwidth obtained from a crossbar switch in the accessing of global resources. Simplicity and expandability are obtained by creating the switch from a grid of microprocessor buses. Features such as using an interrupt module to distribute interrupts on the global buses and allowing variations between local and global bus protocols proved simple to implement and very powerful.

The processing power of the machine is apparently scalable up to the point where the combined memory traffic of all the processors comes close to saturating the bandwidth of all the global buses. High performance is dependant on the ability of software to keep the use of global resources, as opposed to local ones, to a minimum.

When estimating the increase in system performance obtained from the addition of one more processor, a simple probabilistic model serves well. At least in the case of static priority schemes and fairly random memory

accesses across the entire system, the model predicted well the simulated performance changes observed in a wide range of situations.

For special purpose high-throughput applications, multi-microprocessor machines such as the MMU provide a feasible alternative to the conventional high-speed uni-processor approach. The trade-off not investigated here is the relative penalties in the difficulty of software development.

## 6.2 - Future Research

The work in this thesis brings to light additional interesting questions that warrant further investigation. As well as looking into the appropriateness of other applications to the MMU, and exploring other multi-microprocessor architectures, more work is possible on the MMU design itself.

One important question involves adding the capability of reliable operation to the MMU. In the present system, there are many failure modes of a single processor which can bring down the whole machine. Many possible techniques for minimizing this problem and allowing dynamic reconfiguration of the system need to be investigated and evaluated.

Another important question deals with the method of arbitration used on the global buses. A shift from static priority would affect both performance and the performance model.

Also of possible interest would be the addition of other kinds of global bus traffic to the proposed interference model. In some

applications, interrupt, DMA, and semaphore traffic might become significant and require incorporation in the performance estimates.

Another important experiment for the MMU is the writing of a complete software system for a given application. Such software development might uncover desirable additions to the hardware architecture. Measuring the performance of the MMU in an actual application run containing variable loads would provide an interesting test of the proposed performance estimates. Such an experiment would also measure the difficulty of software development for a special purpose multi-microprocessor machine.

# OVERALL MMU ARCHITECTURE

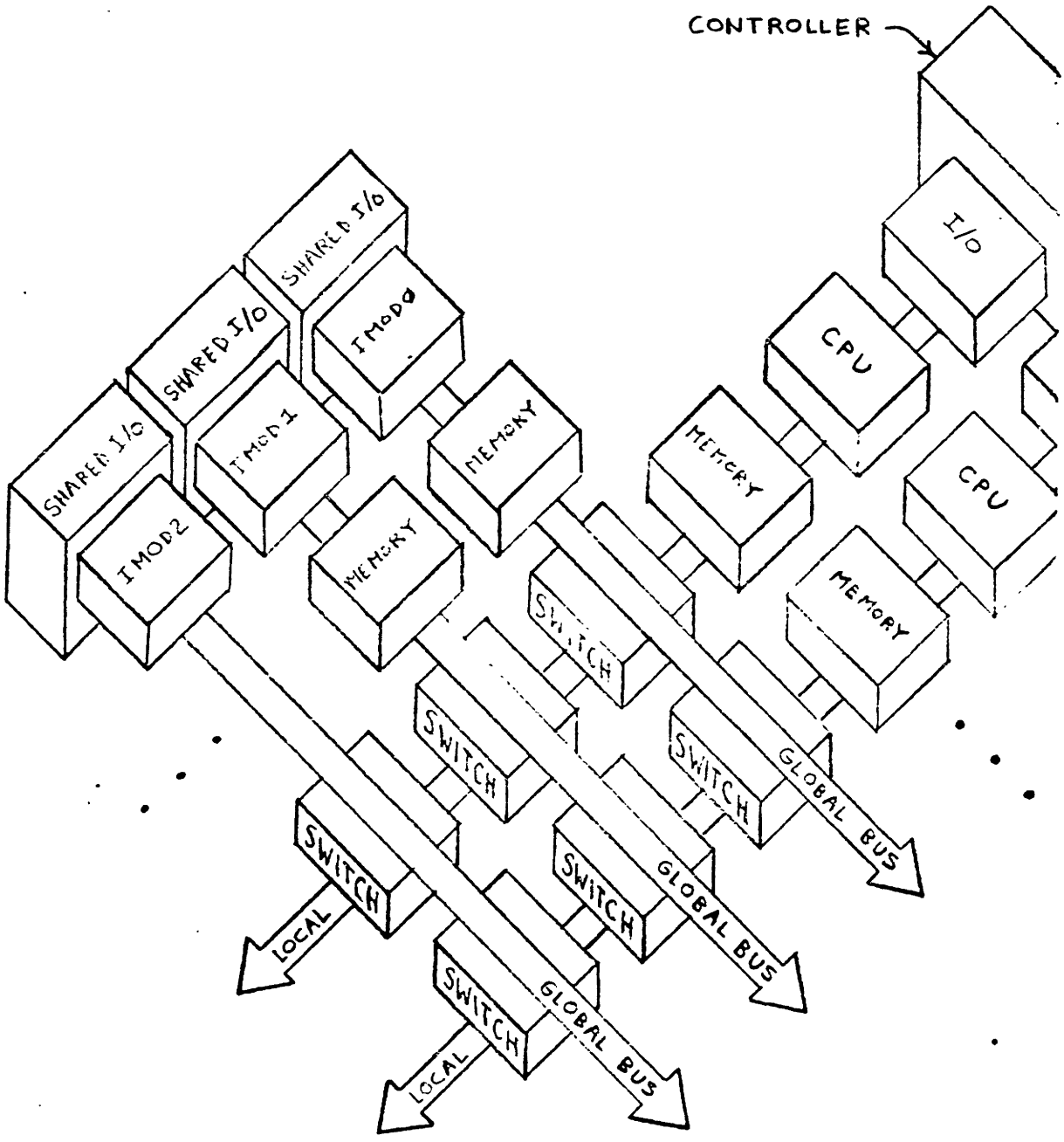
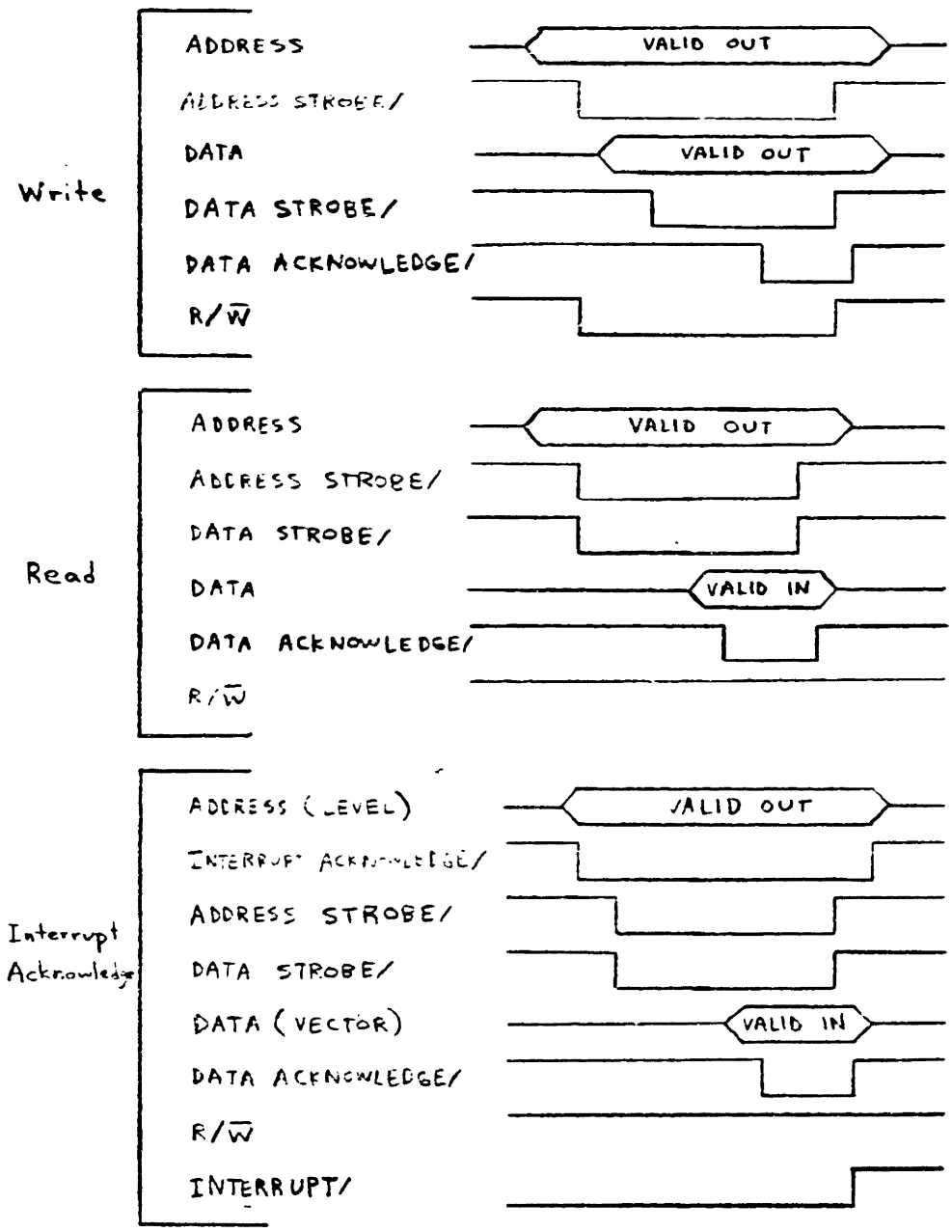
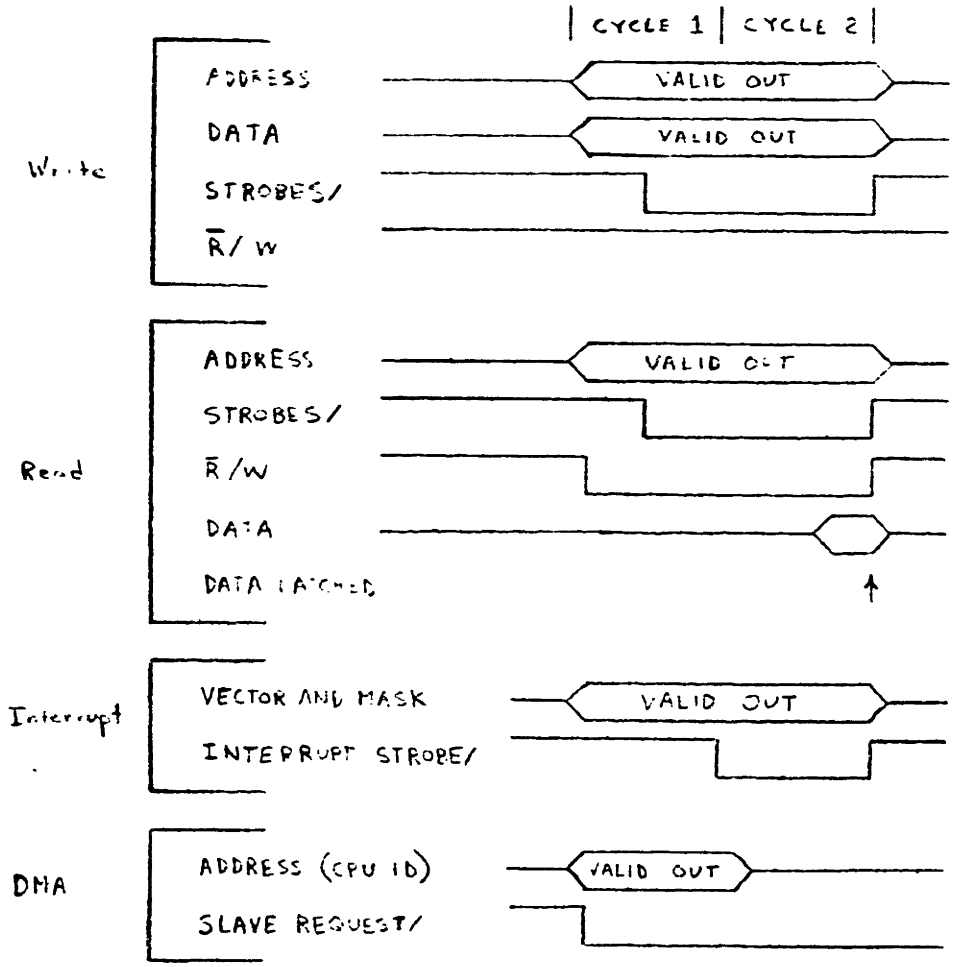



Figure 1



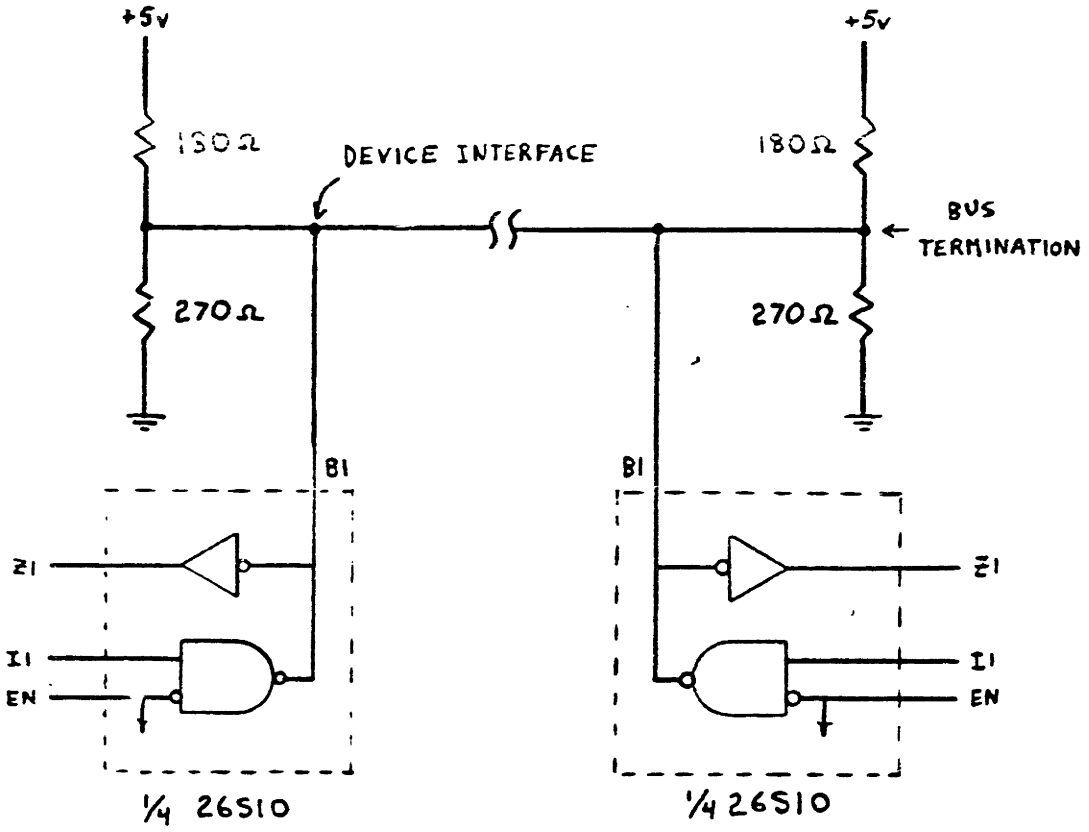

# LOCAL BUS TRANSACTIONS

Figure 2




GLOBAL BUS TRANSACTIONS

Figure 3



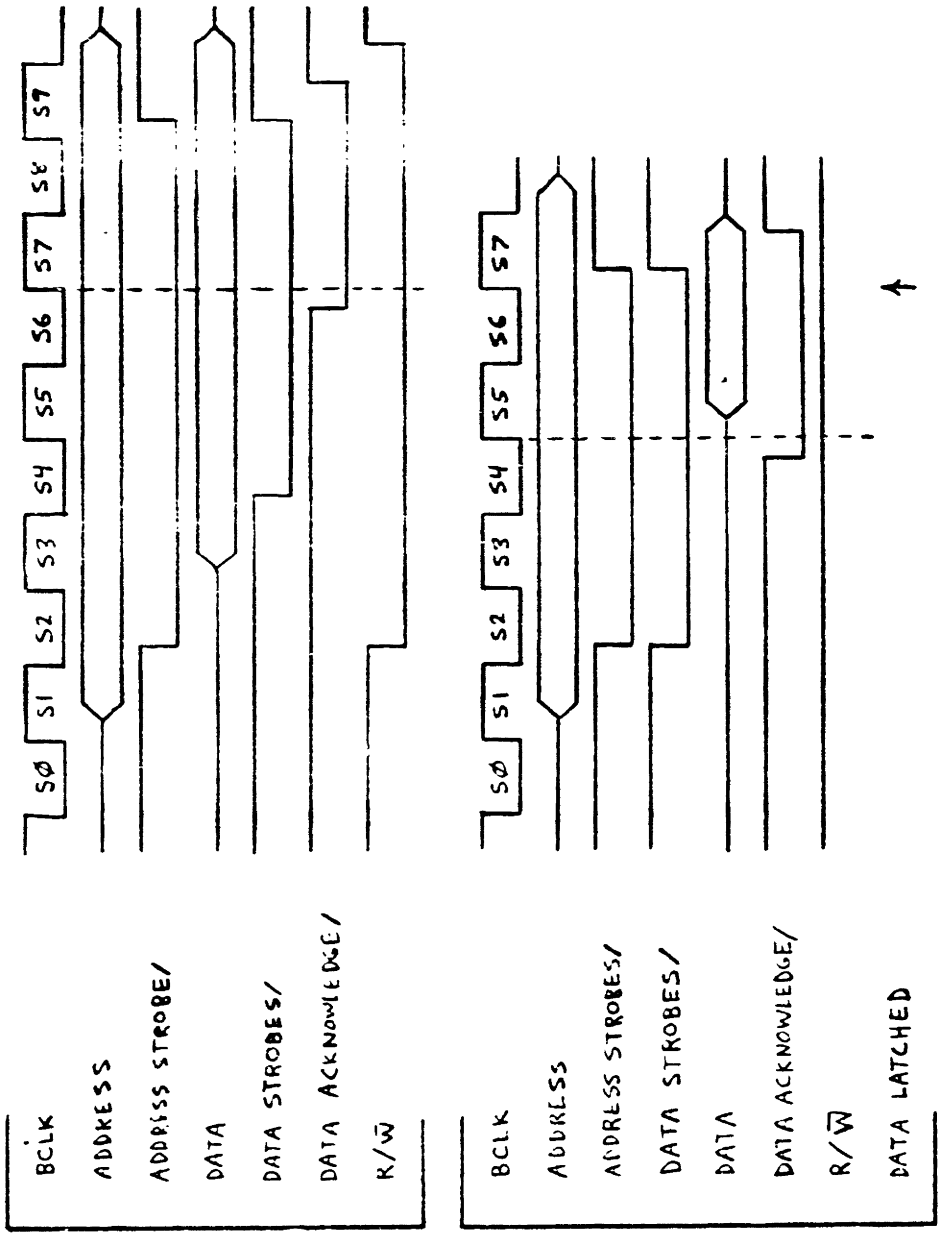

$180\Omega // 270\Omega \approx 100\Omega$ line impedence

$\left(270\Omega / 180\Omega + 270\Omega\right) 5v \approx 3.3v$ TTL high

$I_{OL}$ on bus for 26S10 = 100ma > $5v / 90\Omega$

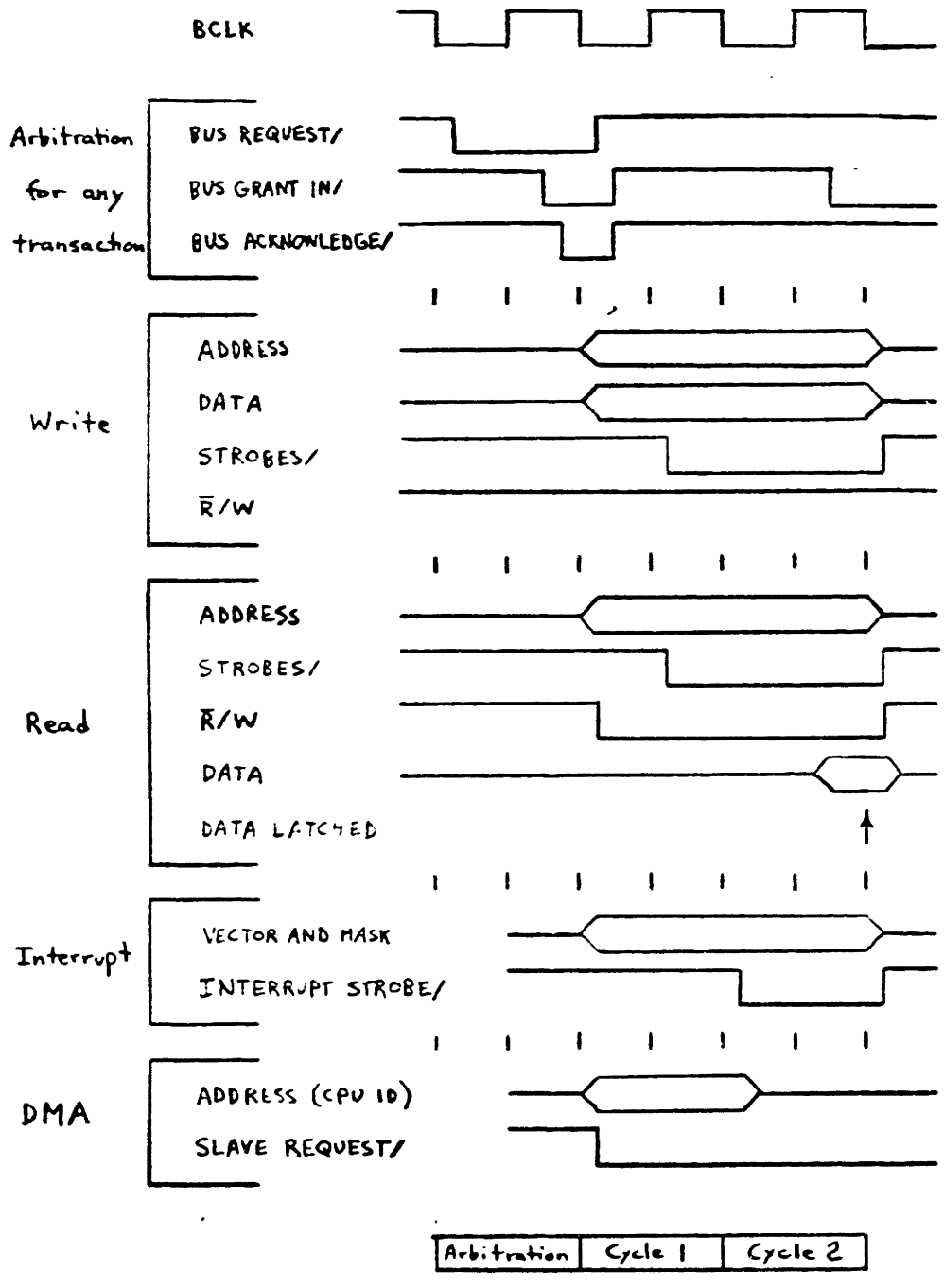
BUS ELECTRICAL SCHEMATIC

Figure 4



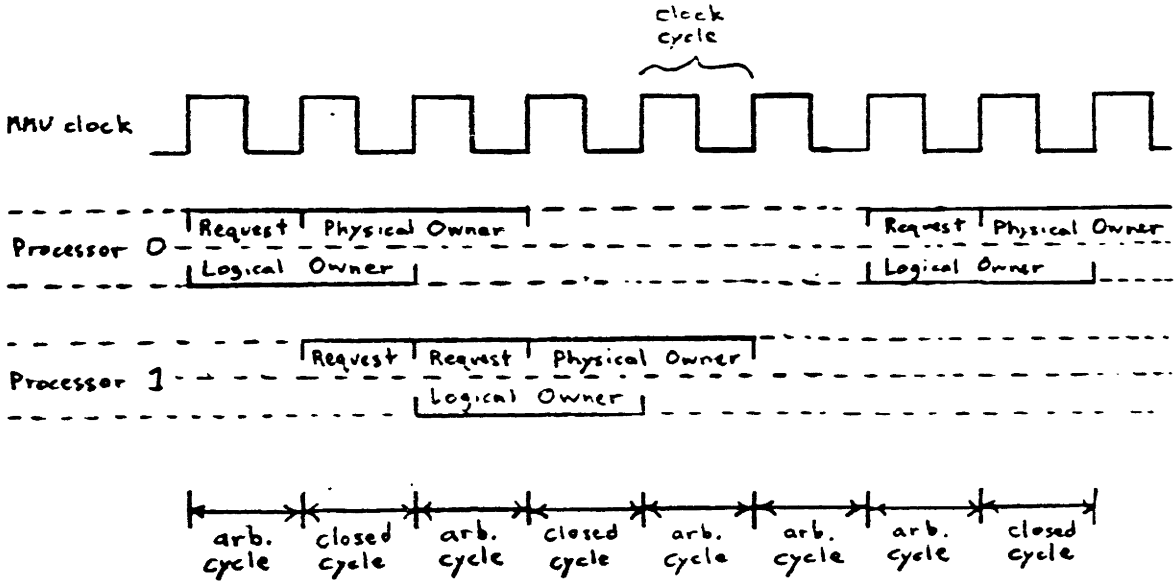

LOCAL BUS TIMING

Figure 5



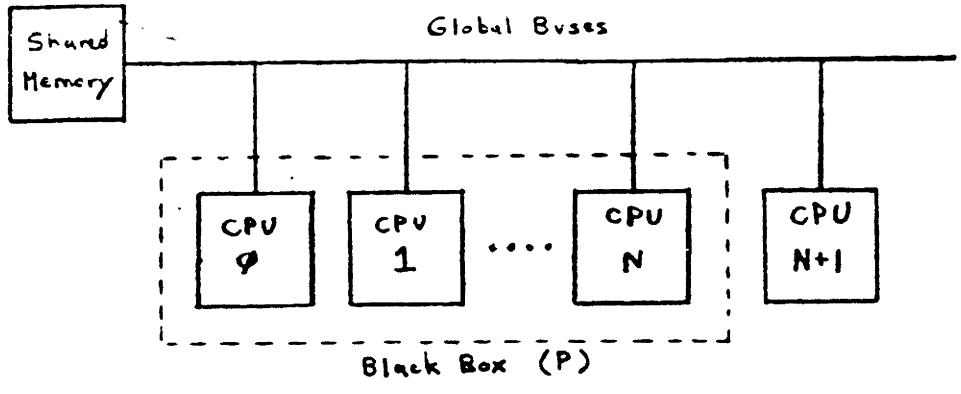

GLOBAL BUS TIMING

Figure 6




GLOBAL BUS ARBITRATION

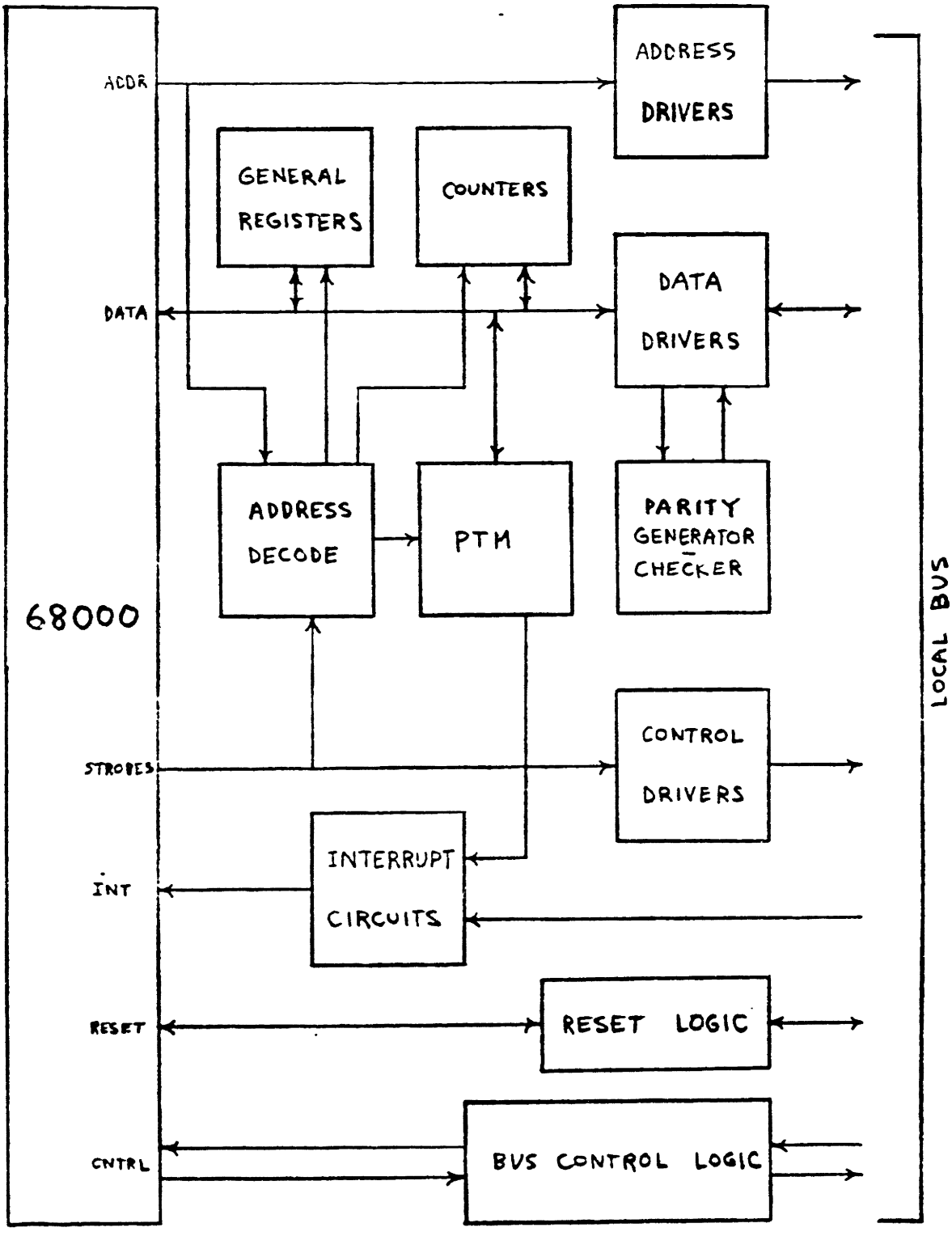
Figure 7



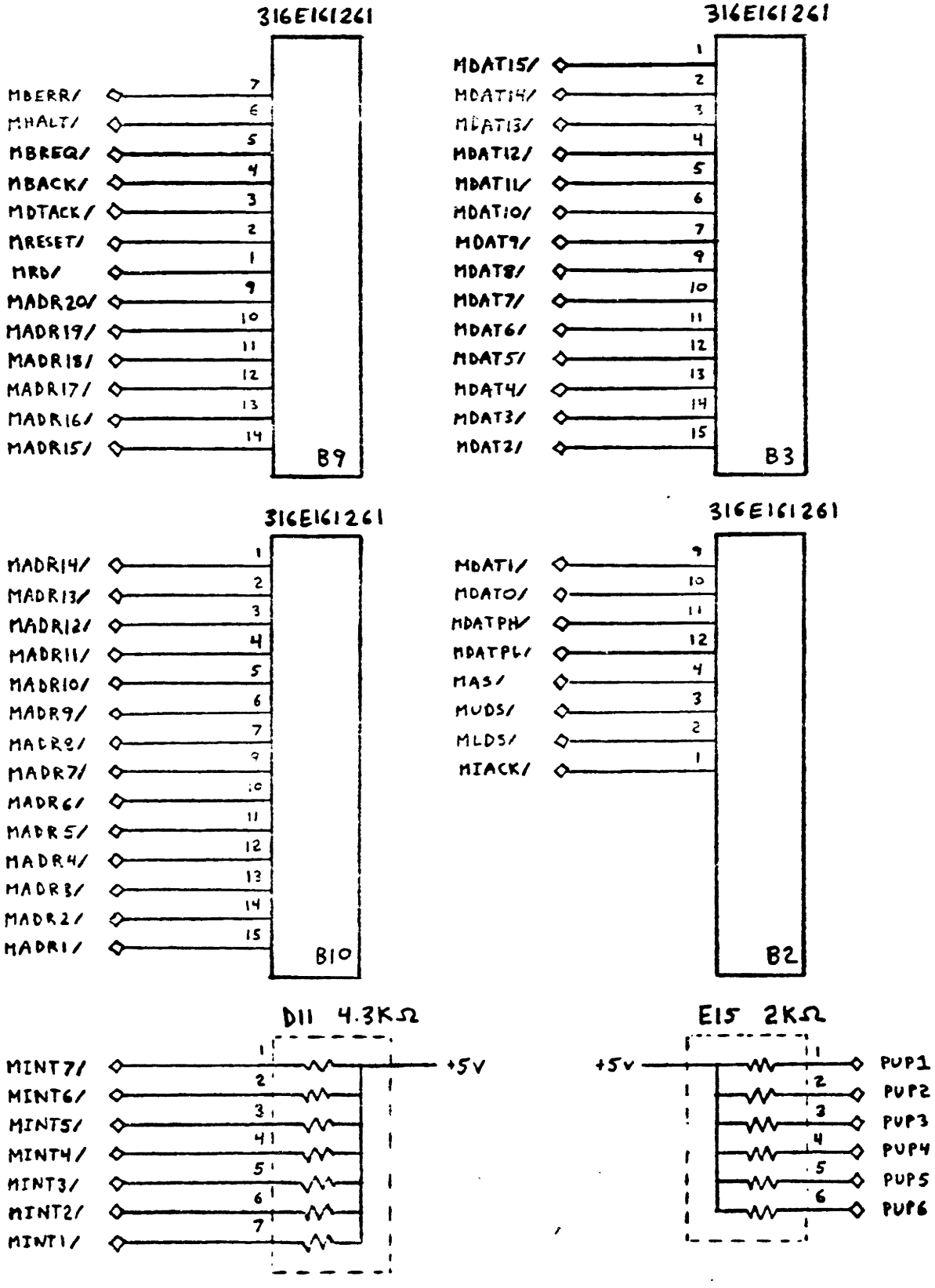

MARGINAL PERFORMANCE MODEL

Figure 8

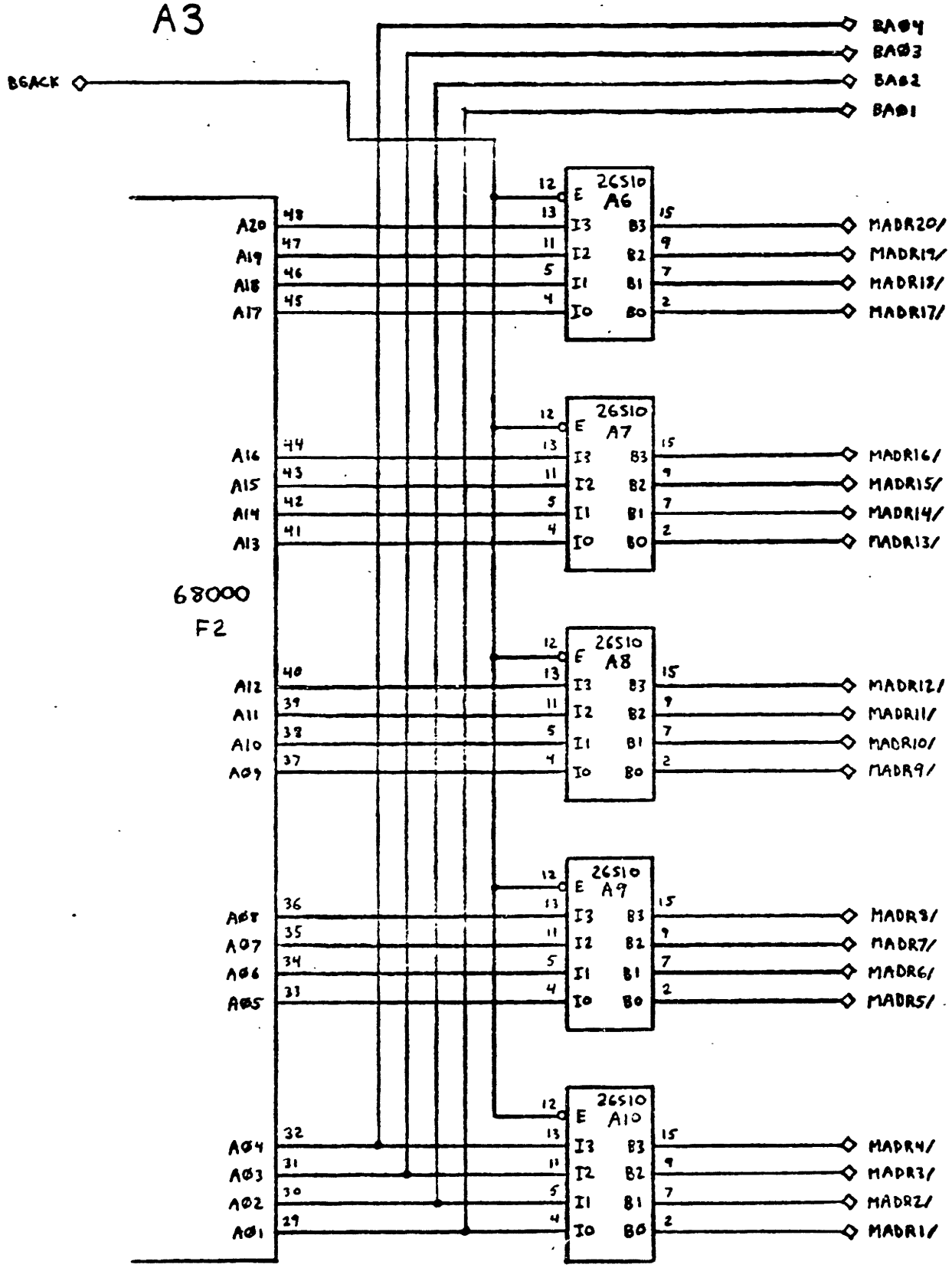
# A1 - CPU


ADDR
DATA
STROBES
INT
RESET
CNTRL
68000
ADDRESS DRIVERS
GENERAL REGISTERS
COUNTERS
DATA DRIVERS
ADDRESS DECODE
PTM
PARITY GENERATOR CHECKER
CONTROL DRIVERS
INTERRUPT CIRCUITS
RESET LOGIC
BUS CONTROL LOGIC
LOCAL BUS

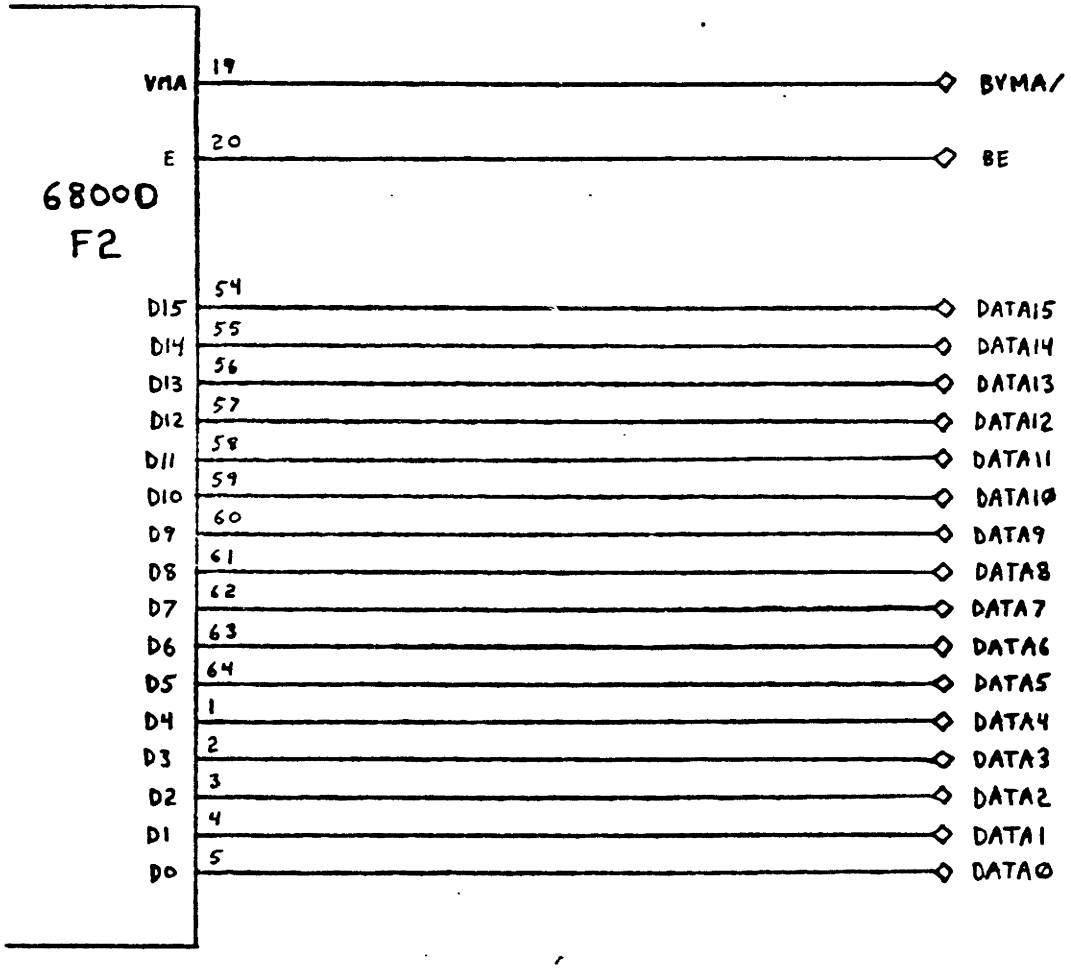
# A2


316E161261
MBERR/ 7
MHALT/ 6
MBREQ/ 5
MBACK/ 4
MDTACK/ 3
MRESET/ 2
MRD/ 1
MADR20/ 9
MADR19/ 10
MADR18/ 11
MADR17/ 12
MADR16/ 13
MADR15/ 14
B9
316E161261
MDAT15/ 1
MDAT14/ 2
MDAT13/ 3
MDAT12/ 4
MDAT11/ 5
MDAT10/ 6
MDAT9/ 7
MDAT8/ 9
MDAT7/ 10
MDAT6/ 11
MDAT5/ 12
MDAT4/ 13
MDAT3/ 14
MDAT2/ 15
B3
316E161261
MADR14/ 1
MADR13/ 2
MADR12/ 3
MADR11/ 4
MADR10/ 5
MADR9/ 6
MADR8/ 7
MADR7/ 9
MADR6/ 10
MADR5/ 11
MADR4/ 12
MADR3/ 13
MADR2/ 14
MADR1/ 15
B10
316E161261
MDAT1/ 9
MDAT0/ 10
MDATPH/ 11
MDATPL/ 12
MAS/ 4
MUDS/ 3
MLDS/ 2
MIACK/ 1
B2
D11 4.3KΩ
MINT7/ 1
MINT6/ 2
MINT5/ 3
MINT4/ 4
MINT3/ 5
MINT2/ 6
MINT1/ 7
+5V
E15 2KΩ
+5V
PUP1 1
PUP2 2
PUP3 3
PUP4 4
PUP5 5
PUP6 6

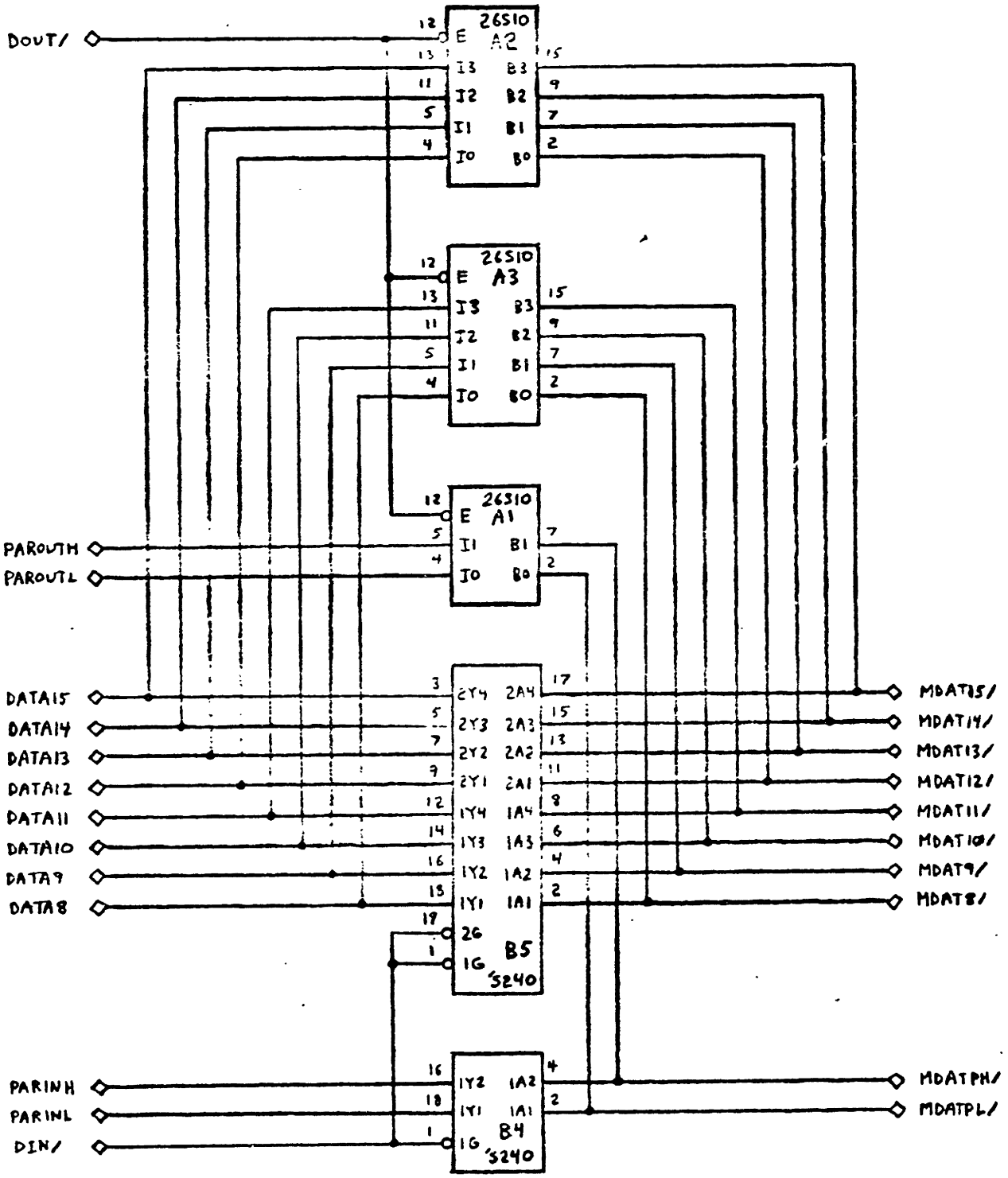
# A3


BGACK
BA04
BA03
BA02
BA01
68000
F2
A20
A19
A18
A17
A16
A15
A14
A13
A12
A11
A10
A09
A08
A07
A06
A05
A04
A03
A02
A01
26S10
A6
A7
A8
A9
A10
E
I3
I2
I1
I0
B3
B2
B1
B0
MADR20/
MADR19/
MADR18/
MADR17/
MADR16/
MADR15/
MADR14/
MADR13/
MADR12/
MADR11/
MADR10/
MADR9/
MADR8/
MADR7/
MADR6/
MADR5/
MADR4/
MADR3/
MADR2/
MADR1/

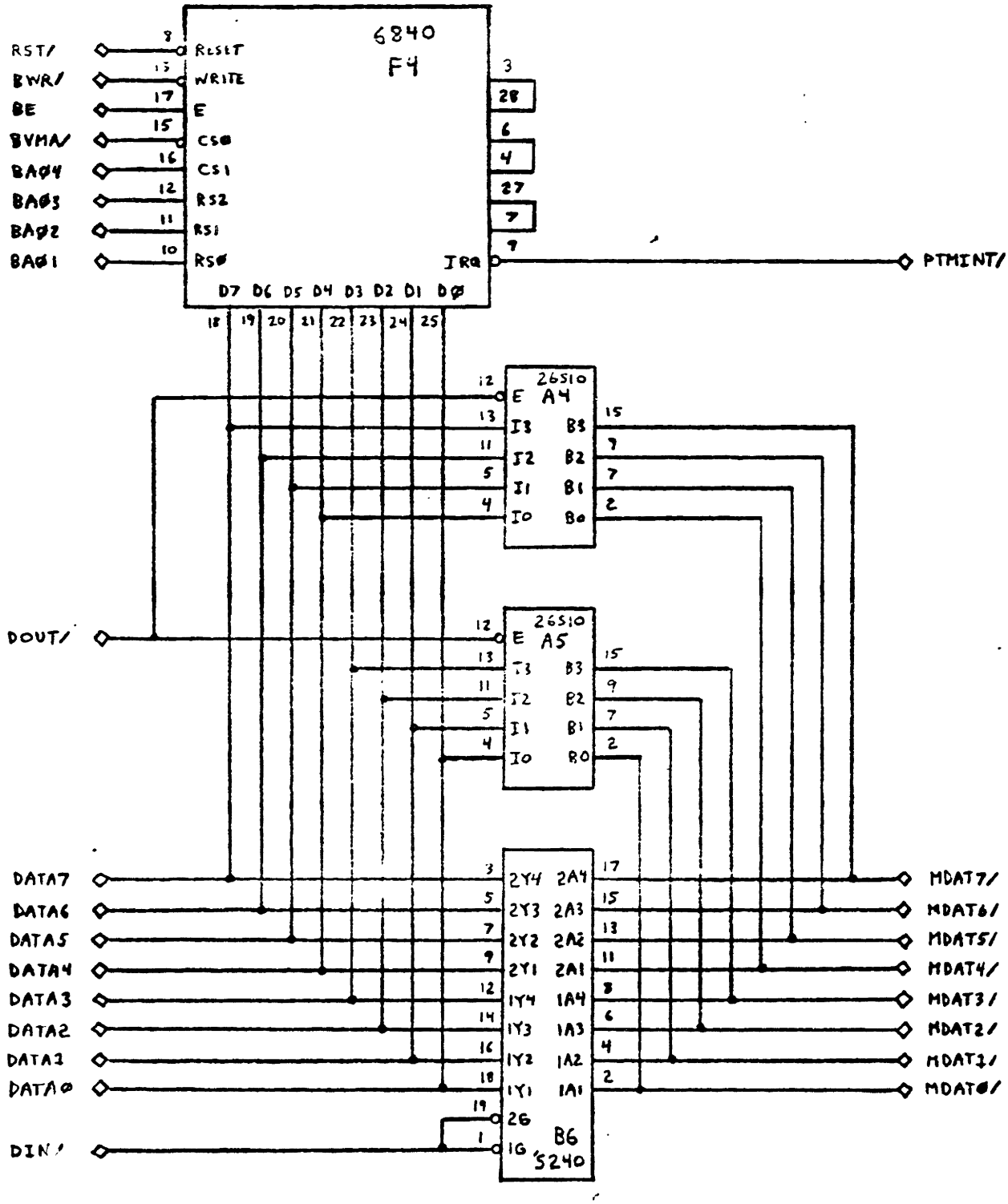
# A4


6800D
F2
VMA
19
BVMA/
E
20
BE
D15
54
DATA15
D14
55
DATA14
D13
56
DATA13
D12
57
DATA12
D11
58
DATA11
D10
59
DATA10
D9
60
DATA9
D8
61
DATA8
D7
62
DATA7
D6
63
DATA6
D5
64
DATA5
D4
1
DATA4
D3
2
DATA3
D2
3
DATA2
D1
4
DATA1
D0
5
DATA0

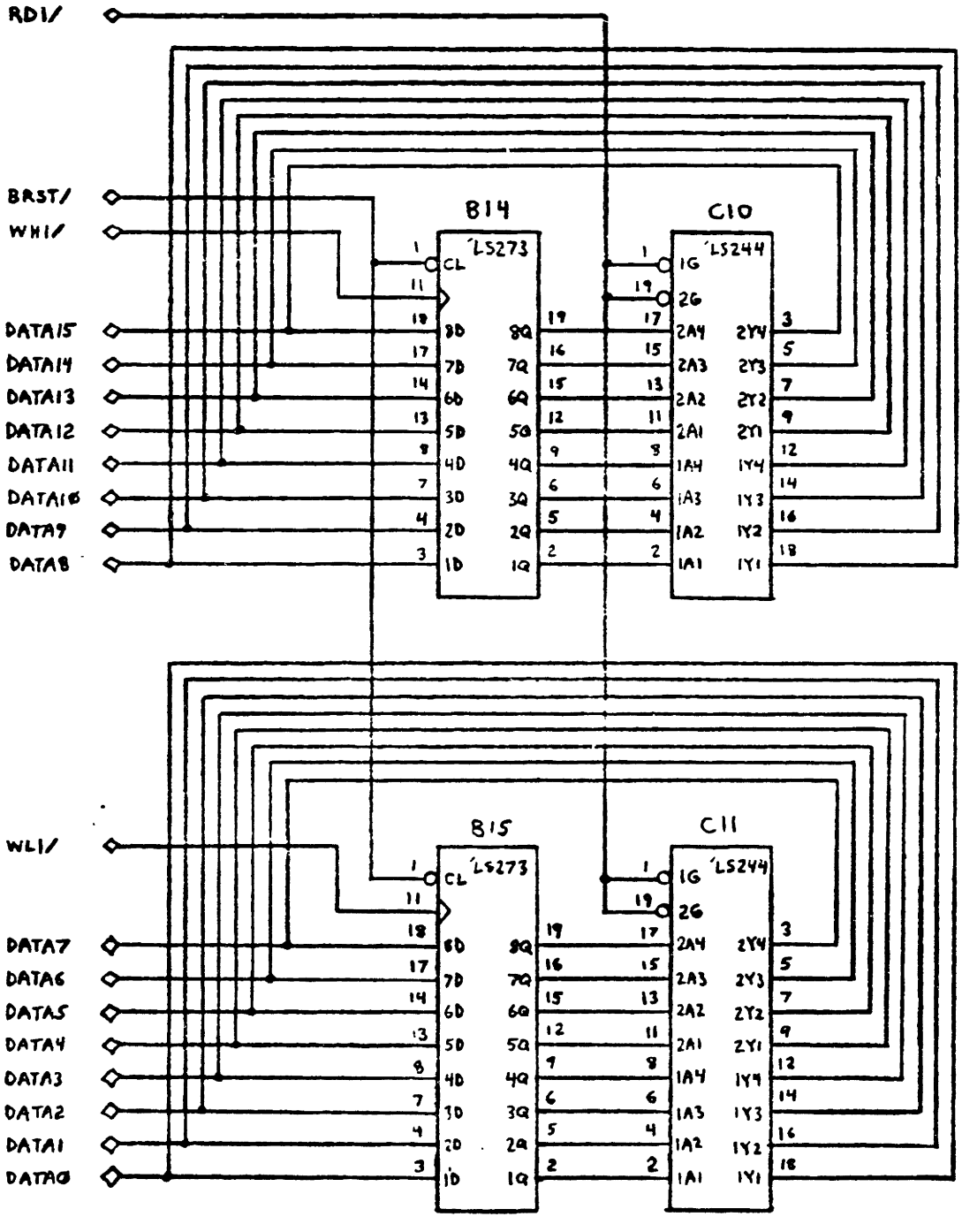
# A5


DOUT/
26S10 A2
26S10 A3
26S10 A1
PAROUTH
PAROUTL
DATA15
DATA14
DATA13
DATA12
DATA11
DATA10
DATA9
DATA8
B5 'S240
PARINH
PARINL
DIN/
B4 'S240
MDAT15/
MDAT14/
MDAT13/
MDAT12/
MDAT11/
MDAT10/
MDAT9/
MDAT8/
MDATPH/
MDATPL/

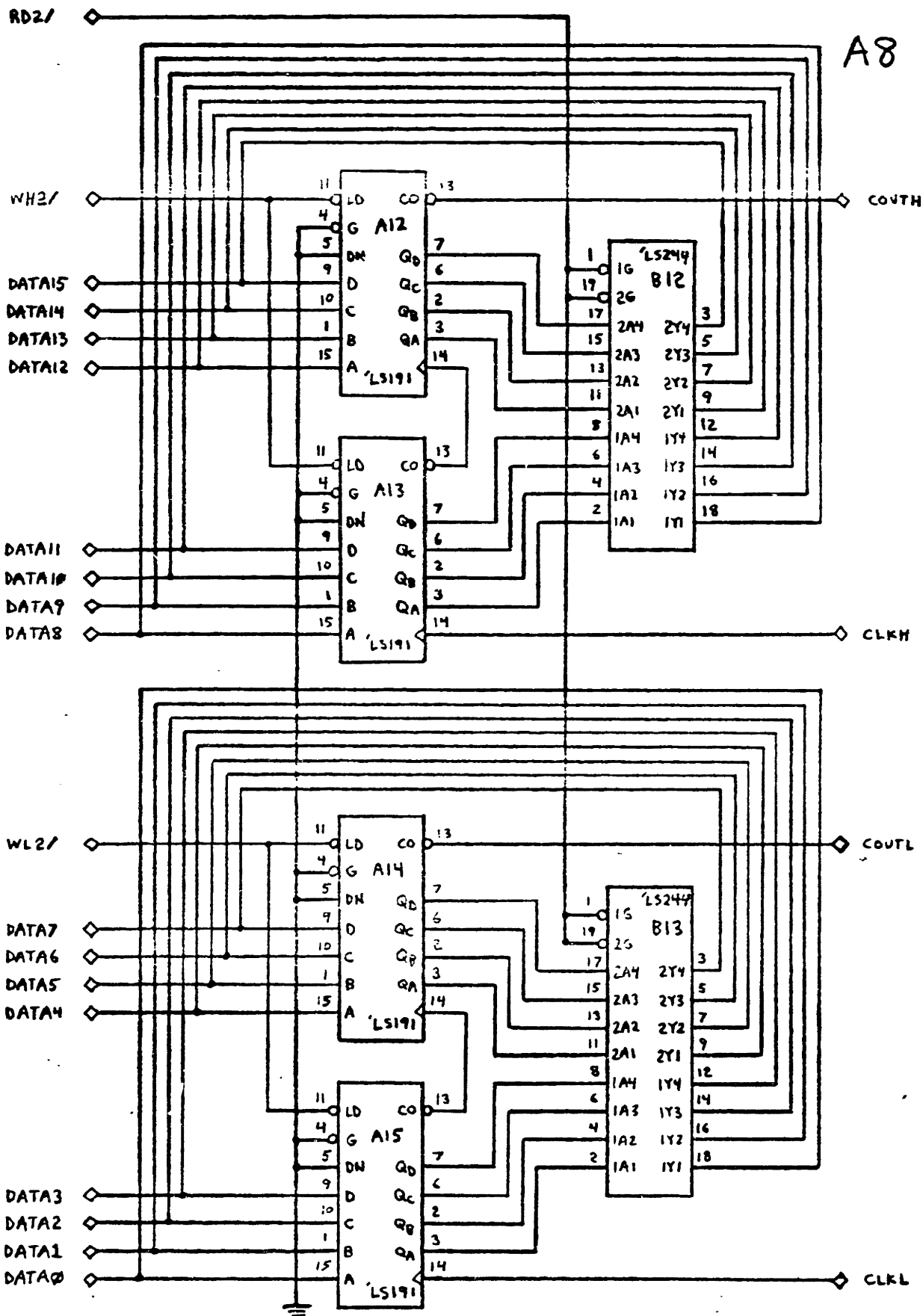
# A6


6840
F4
RST/
BWR/
BE
BVMA/
BA04
BA03
BA02
BA01
RESET
WRITE
E
CS0
CS1
RS2
RS1
RS0
IRQ
PTMINT/
D7 D6 D5 D4 D3 D2 D1 D0
26S10
E A4
I3 I2 I1 I0
B3 B2 B1 B0
26S10
E A5
I3 I2 I1 I0
B3 B2 B1 B0
DOUT/
DATA7
DATA6
DATA5
DATA4
DATA3
DATA2
DATA1
DATA0
DIN/
2Y4 2Y3 2Y2 2Y1
2A4 2A3 2A2 2A1
1Y4 1Y3 1Y2 1Y1
1A4 1A3 1A2 1A1
2G
1G
B6
'S240
MDAT7/
MDAT6/
MDAT5/
MDAT4/
MDAT3/
MDAT2/
MDAT1/
MDAT0/

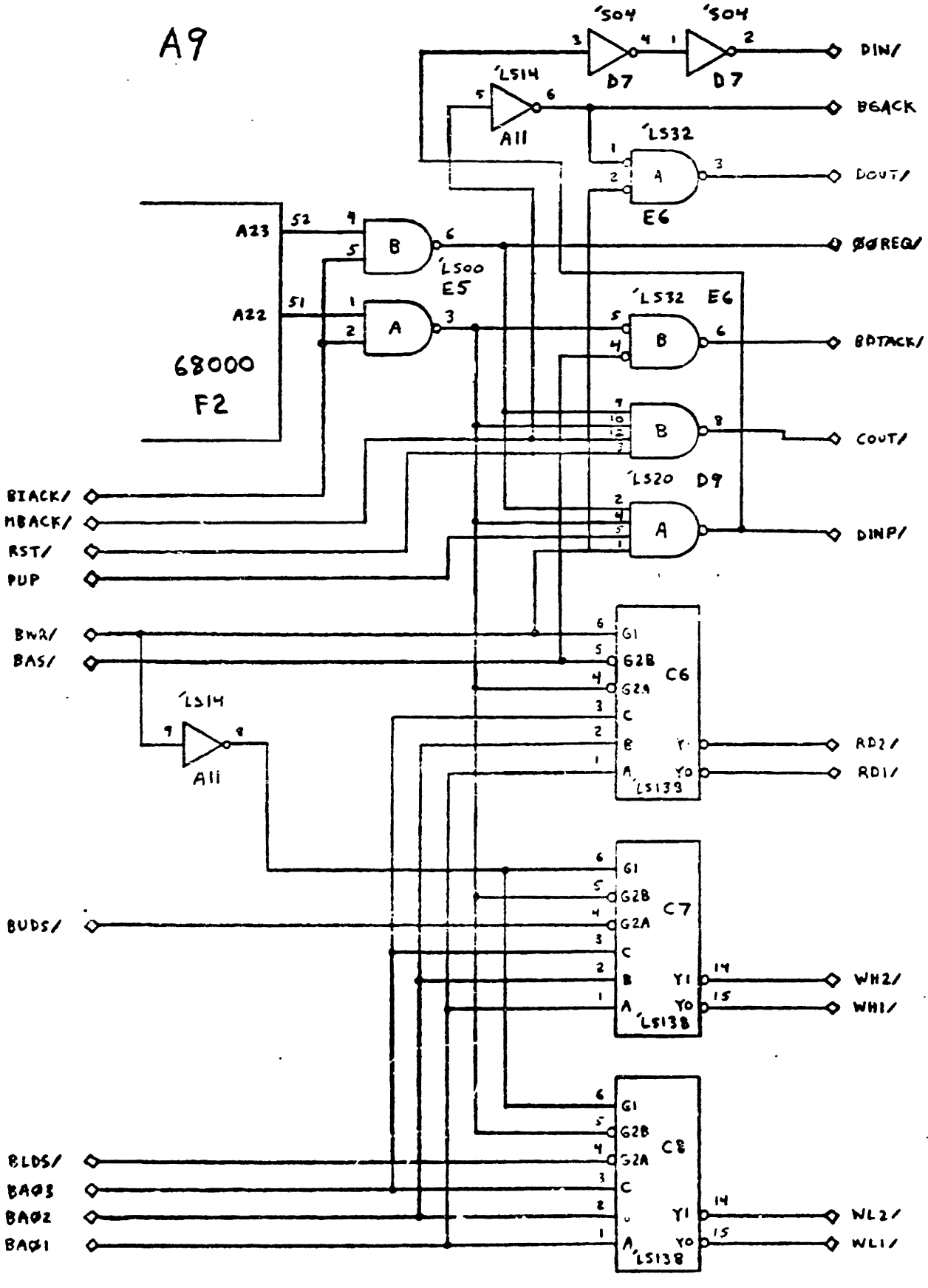
# A7


RD1/
BRST/
WH1/
DATA15
DATA14
DATA13
DATA12
DATA11
DATA10
DATA9
DATA8
B14
'LS273
CL
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
C10
'LS244
1G
2G
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
WL1/
DATA7
DATA6
DATA5
DATA4
DATA3
DATA2
DATA1
DATA0
B15
C11

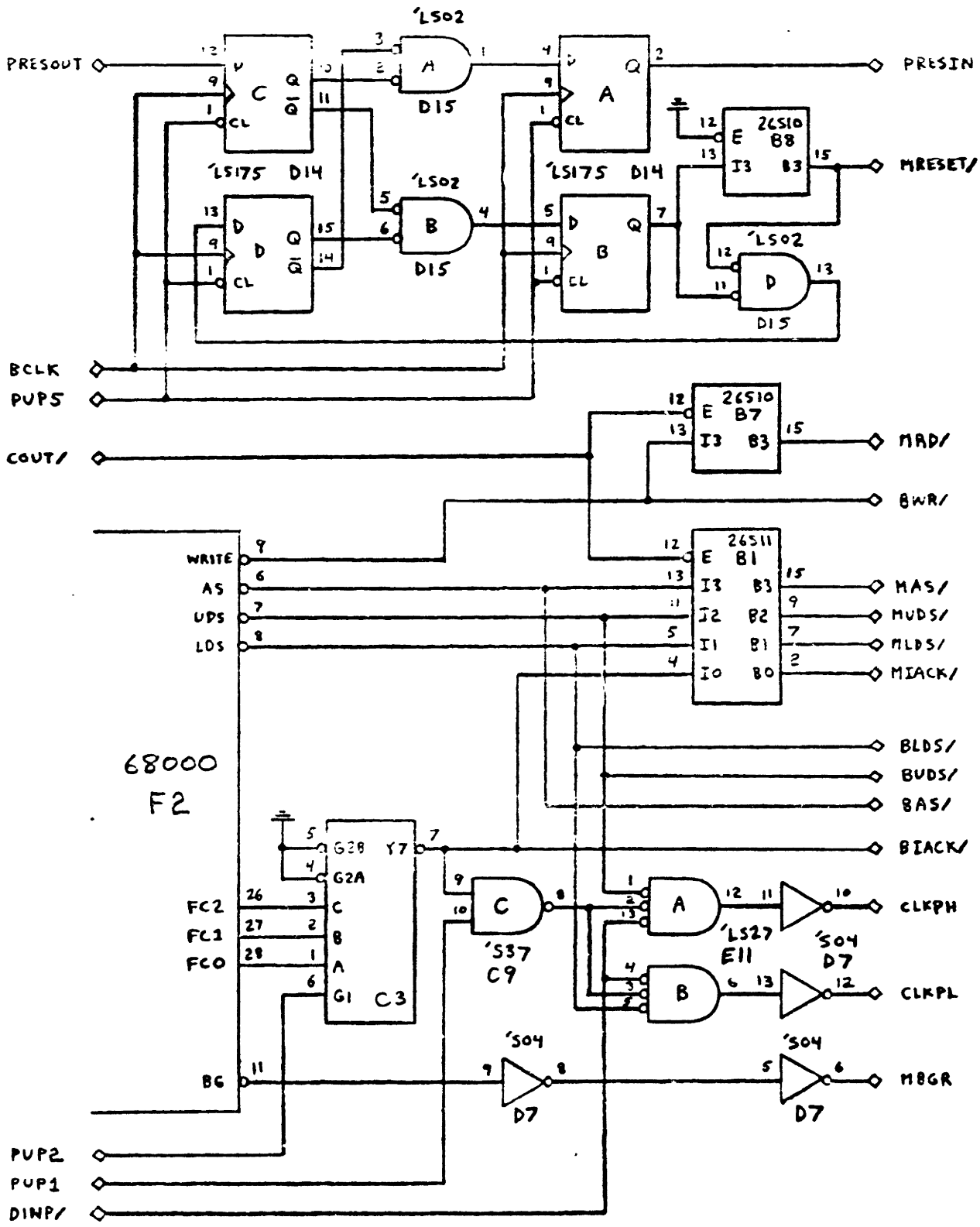
A8

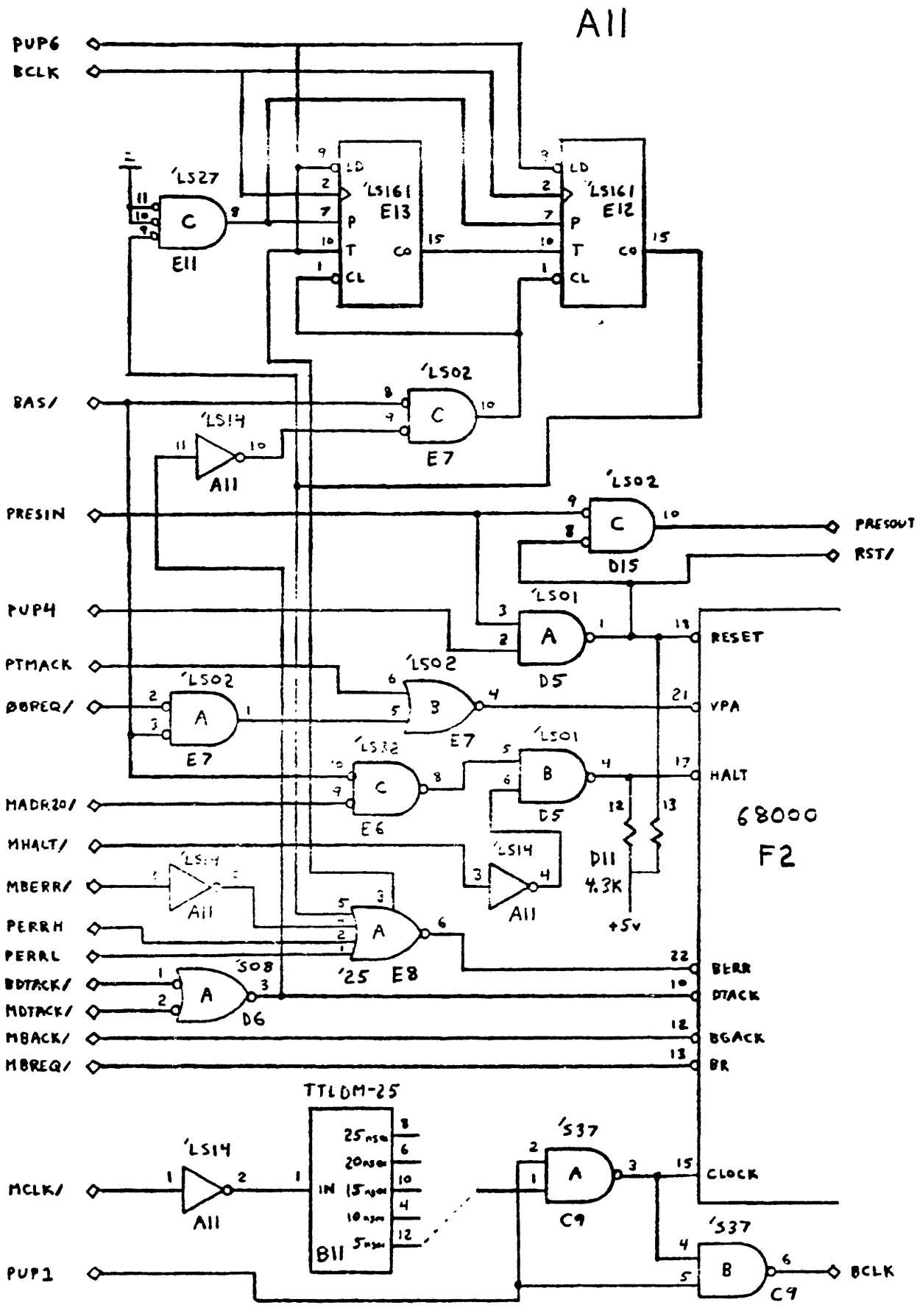

RD2/
WH2/
DATA15
DATA14
DATA13
DATA12
DATA11
DATA10
DATA9
DATA8
WL2/
DATA7
DATA6
DATA5
DATA4
DATA3
DATA2
DATA1
DATA0
A12
A13
A14
A15
'LS191
'LS244
B12
B13
COUTH
CLKH
COUTL
CLKL

# A9

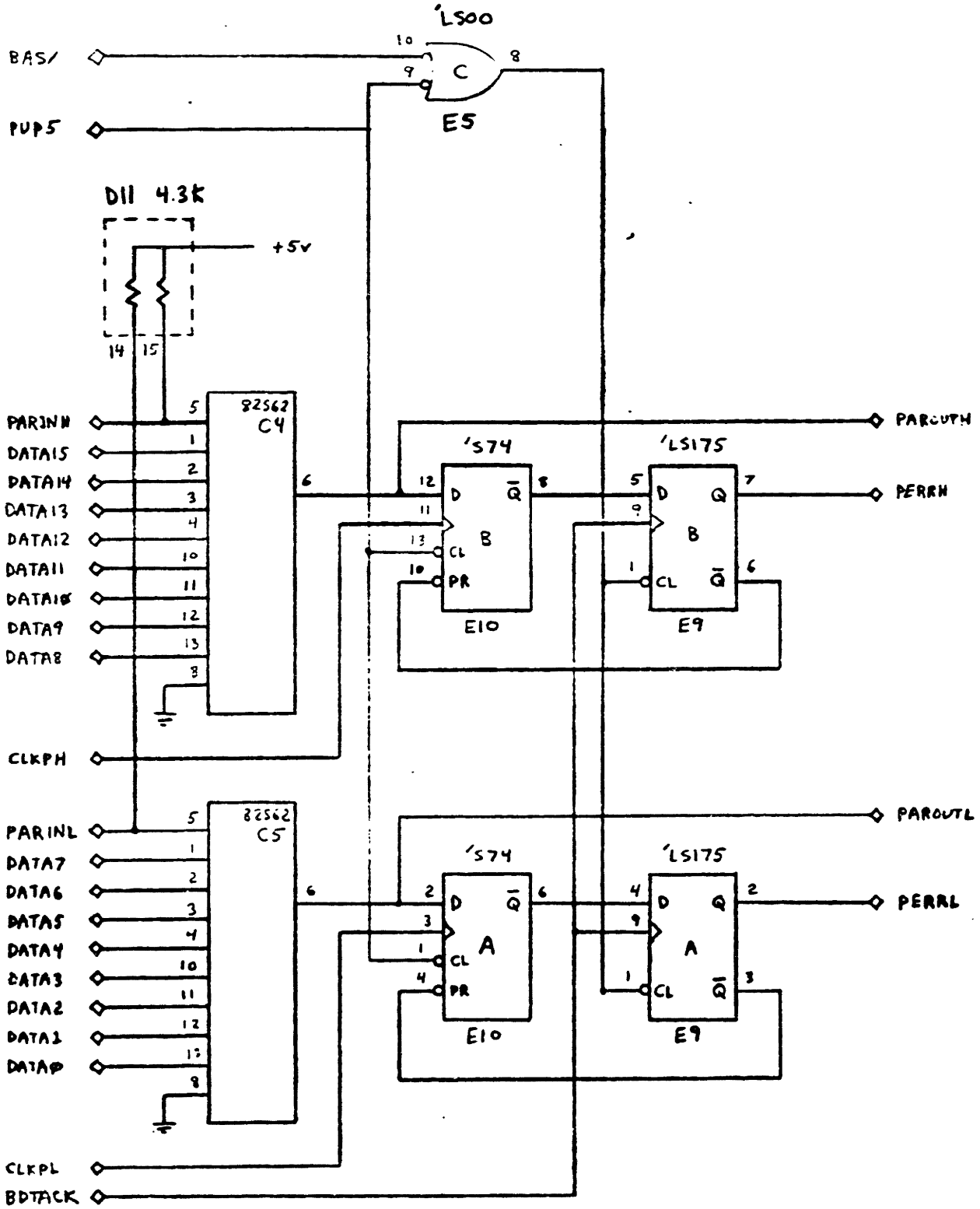

'S04
'S04
D7
D7
DIN/
'LS14
A11
BEACK
'LS32
A
E6
DOUT/
A23
A22
68000
F2
B
'LS00
E5
ØØREQ/
'LS32 E6
B
BDTACK/
B
COUT/
'LS20 D9
A
DINP/
BIACK/
MBACK/
RST/
PUP
BWR/
BAS/
'LS14
A11
G1
G2B
G2A
C
B
A
C6
'LS139
Y1
Y0
RD2/
RD1/
BUDS/
C7
'LS138
WH2/
WH1/
BLDS/
BAØ3
BAØ2
BAØ1
C8
WL2/
WL1/

# A10

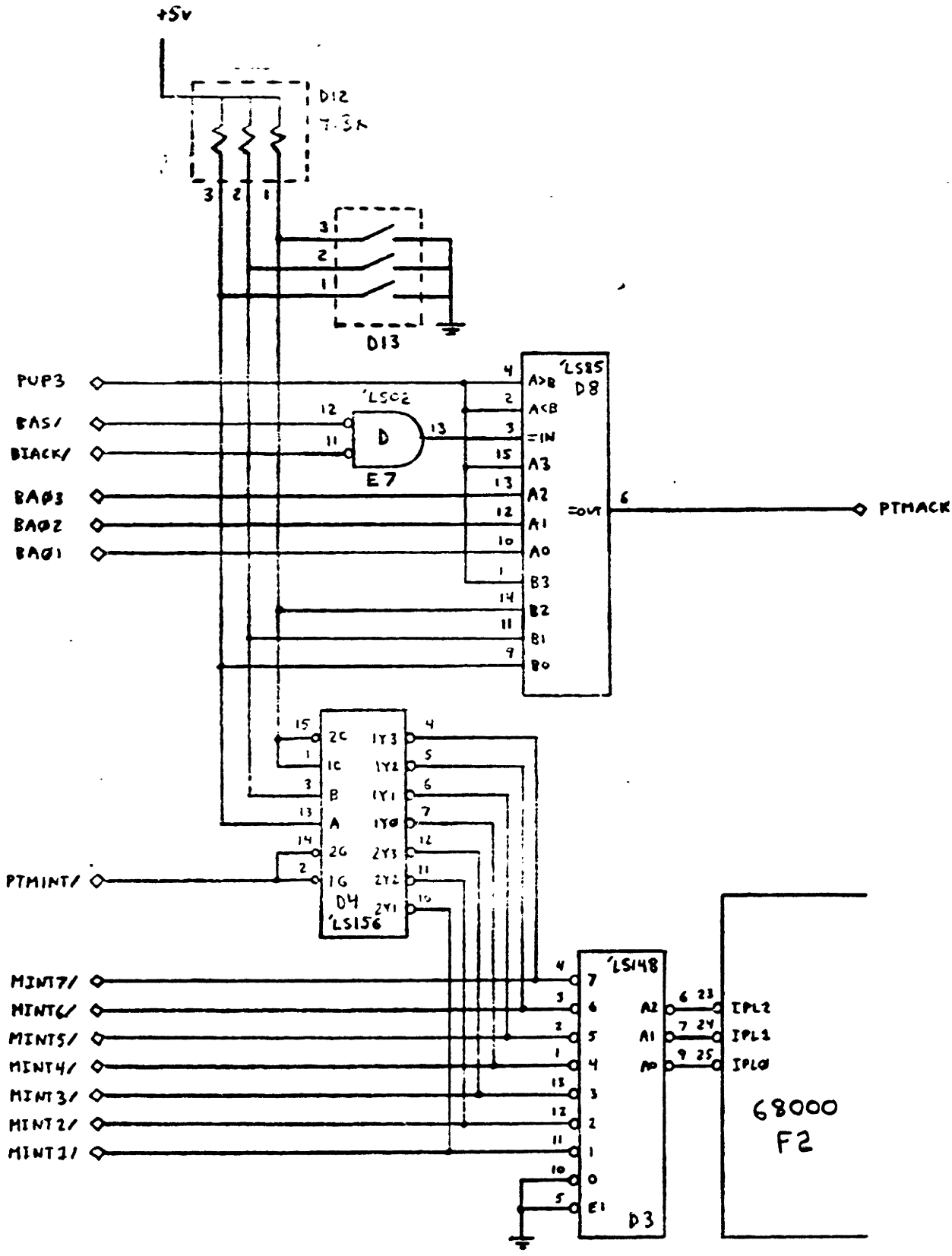

PRESOUT
PRESIN
'LS02
D15
'LS175 D14
26S10 B8
MRESET/
BCLK
PUPS
COUT/
26S10 B7
MAD/
BWR/
26S11 B1
WRITE
AS
UDS
LDS
MAS/
MUDS/
MLDS/
MIACK/
68000 F2
BLDS/
BUDS/
BAS/
BIACK/
G2B
G2A
FC2
FC1
FC0
C3
'S37 C9
'LS27 E11
'S04 D7
CLKPH
CLKPL
BG
MBGR
PUP2
PUP1
DINP/


A11
PUP6
BCLK
'LS27
C
E11
'LS161
E13
'LS161
E12
LD
P
T
CO
CL
BAS/
'LS02
E7
'LS14
A11
PRESIN
D15
PRESOUT
RST/
PUP4
'LS01
D5
RESET
PTMACK
BGREQ/
A
E7
B
VPA
'LS32
E6
HALT
MADR20/
MHALT/
MBERR/
PERRH
PERRL
D11
4.3K
+5v
68000
F2
'25 E8
BERR
BDTACK/
MDTACK/
'S08
D6
DTACK
MBACK/
BGACK
MBREQ/
BR
TTLDM-25
B11
25nsec
20nsec
15nsec
10nsec
5nsec
IN
MCLK/
'S37
C9
CLOCK
PUP1
BCLK

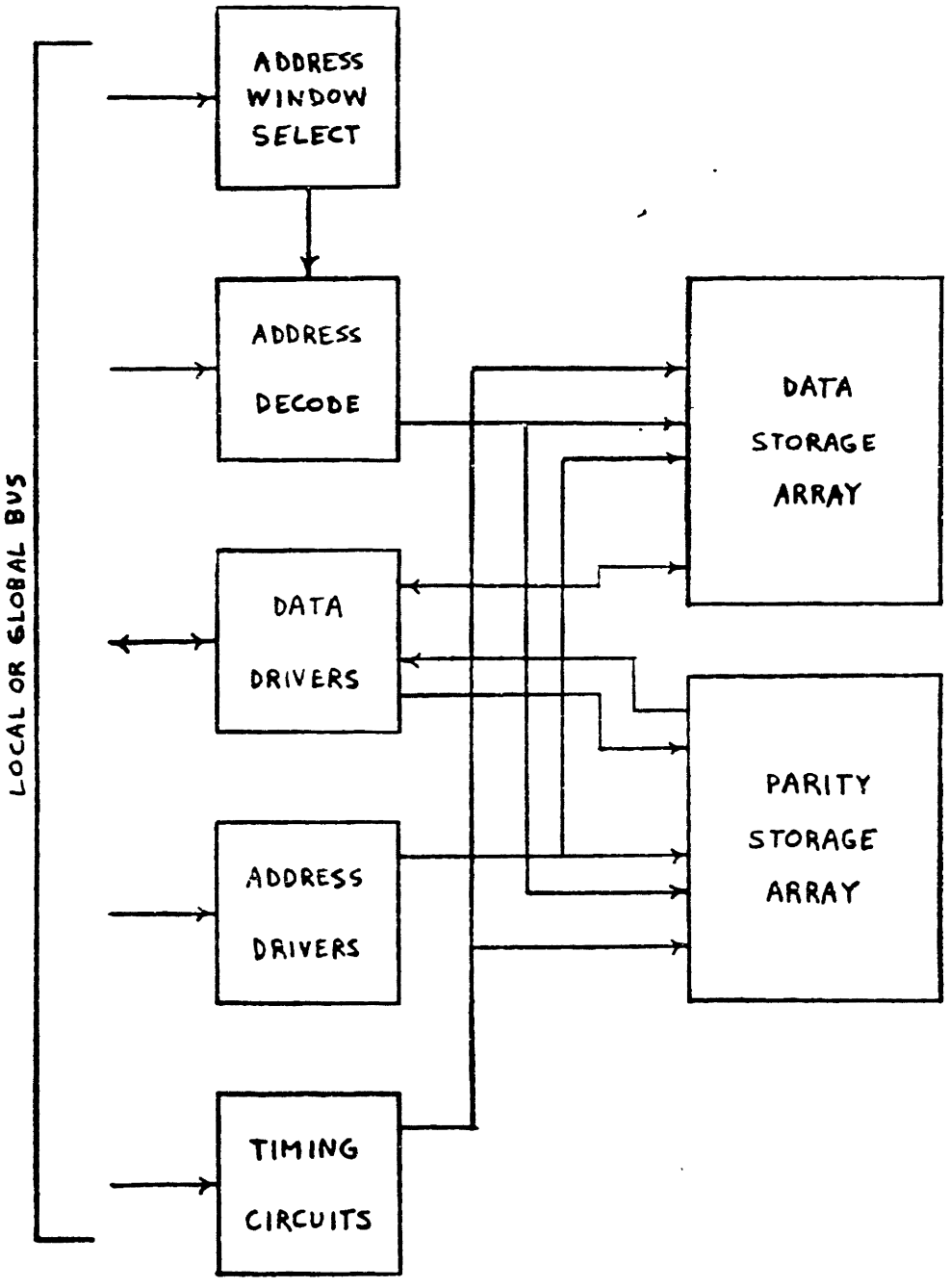
# A12

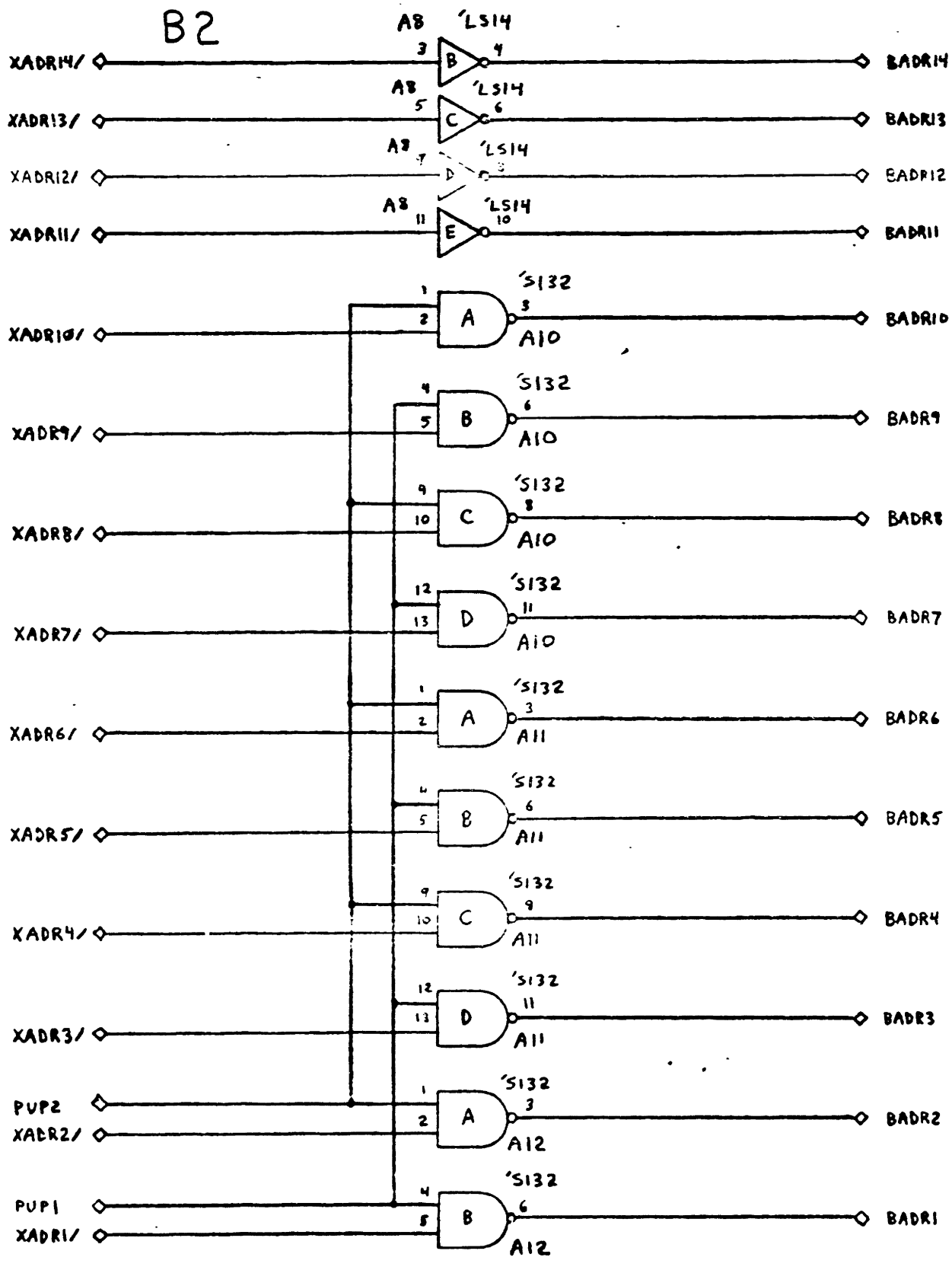

'LS00
BAS/
PUP5
C
E5
D11 4.3K
+5V
14
15
82S62
C4
PARINH
DATA15
DATA14
DATA13
DATA12
DATA11
DATA10
DATA9
DATA8
CLKPH
'S74
'LS175
B
E10
E9
PAROUTH
PERRH
82S62
C5
PARINL
DATA7
DATA6
DATA5
DATA4
DATA3
DATA2
DATA1
DATA0
CLKPL
BDTACK
A
PAROUTL
PERRL

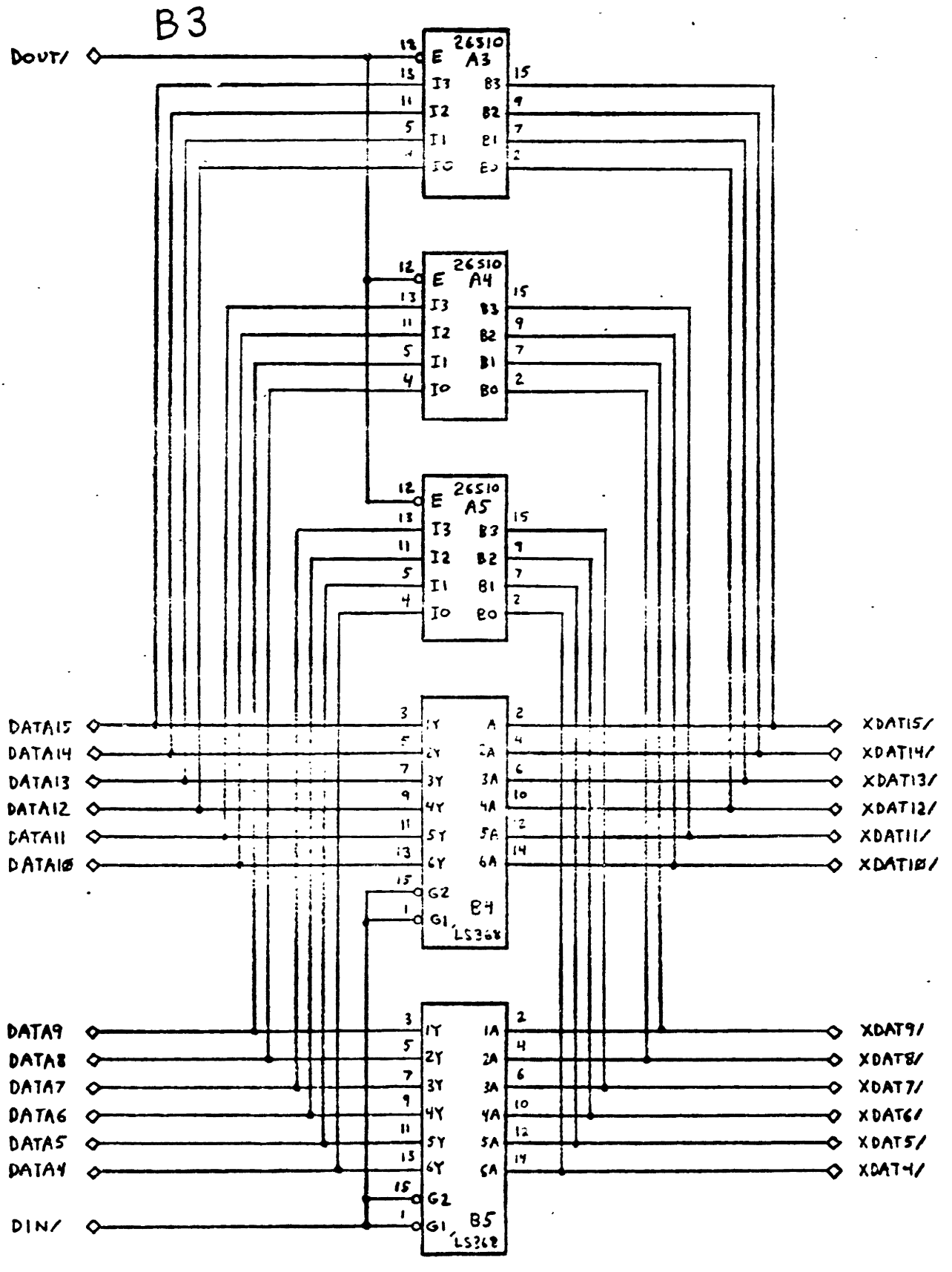
# A13

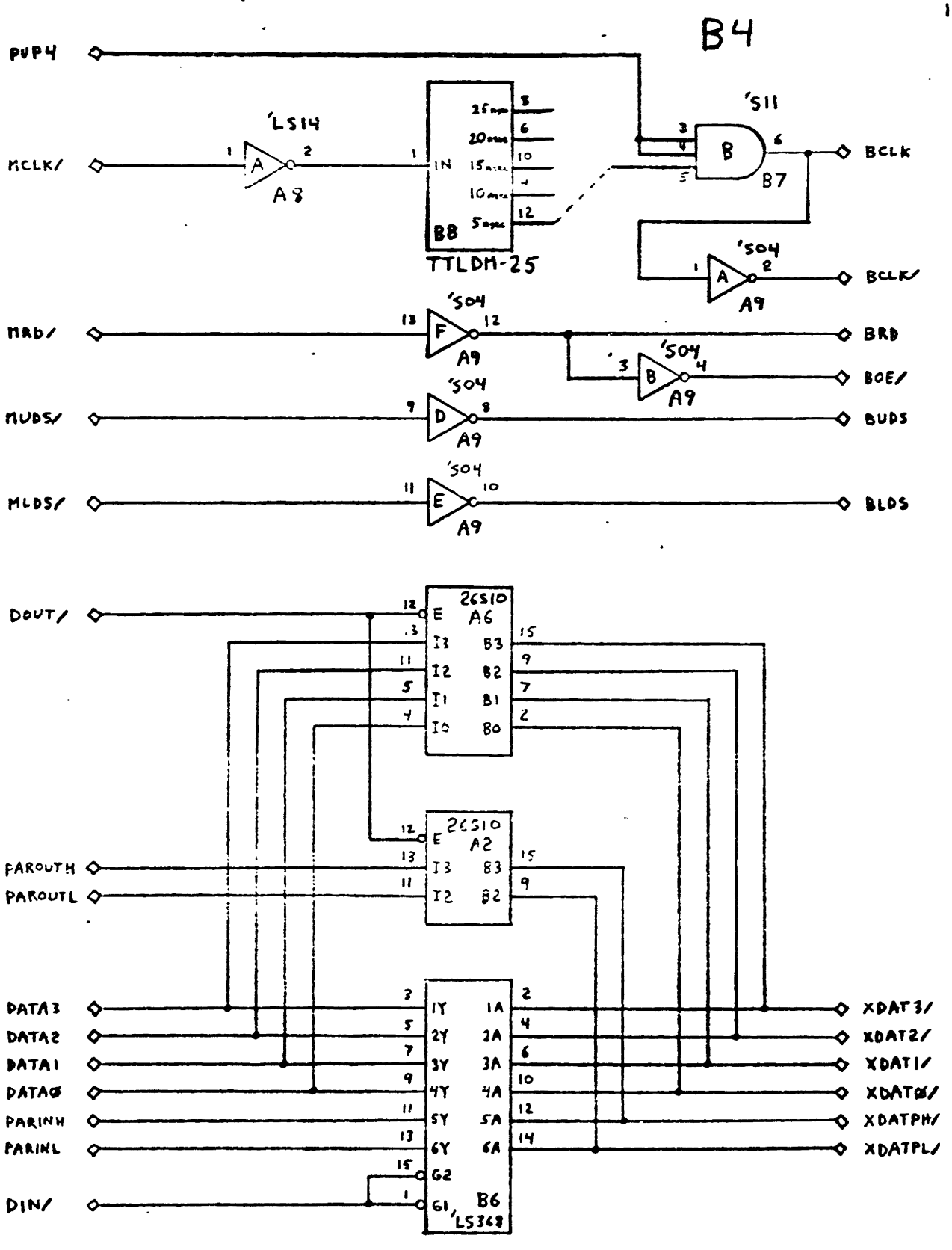

+5v
D12
7.3K
3
2
1
D13
PUP3
BAS/
BIACK/
BAØ3
BAØ2
BAØ1
'LS02
D
E7
'LS85
D8
A>B
A<B
=IN
A3
A2
A1
A0
B3
B2
B1
B0
=OUT
PTMACK
PTMINT/
2C
1C
B
A
2G
1G
1Y3
1Y2
1Y1
1Y0
2Y3
2Y2
2Y1
D4
'LS156
MINT7/
MINT6/
MINT5/
MINT4/
MINT3/
MINT2/
MINT1/
'LS148
A2
A1
A0
E1
D3
IPL2
IPL1
IPL0
68000
F2

# BI-MEMORY

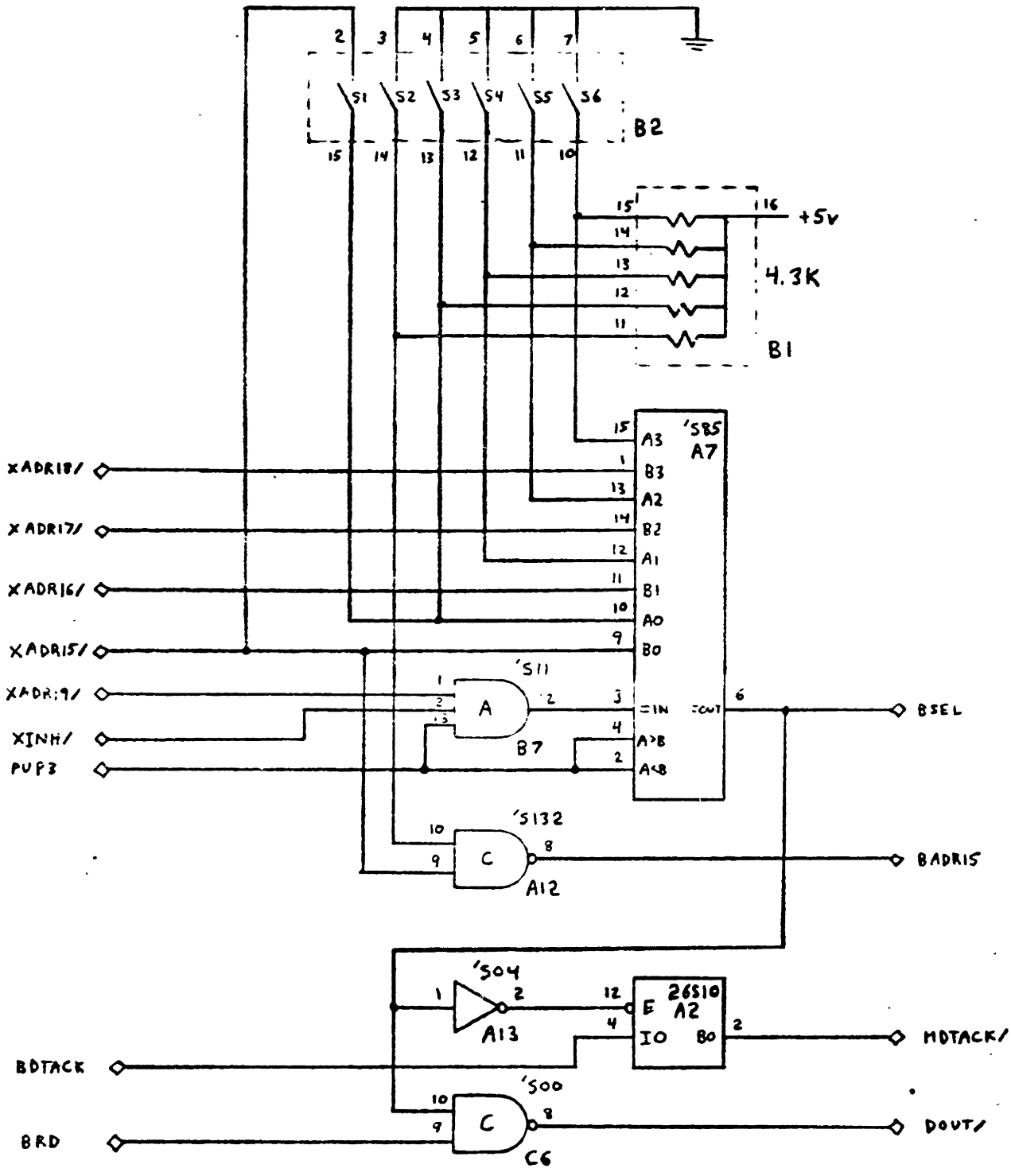

ADDRESS WINDOW SELECT
ADDRESS DECODE
DATA DRIVERS
ADDRESS DRIVERS
TIMING CIRCUITS
DATA STORAGE ARRAY
PARITY STORAGE ARRAY
LOCAL OR GLOBAL BUS


B2
A8 'LS14
XADR14/
BADR14
XADR13/
BADR13
XADR12/
BADR12
XADR11/
BADR11
'S132
XADR10/
BADR10
A10
XADR9/
BADR9
XADR8/
BADR8
XADR7/
BADR7
XADR6/
BADR6
A11
XADR5/
BADR5
XADR4/
BADR4
XADR3/
BADR3
PUP2
XADR2/
BADR2
A12
PUP1
XADR1/
BADR1

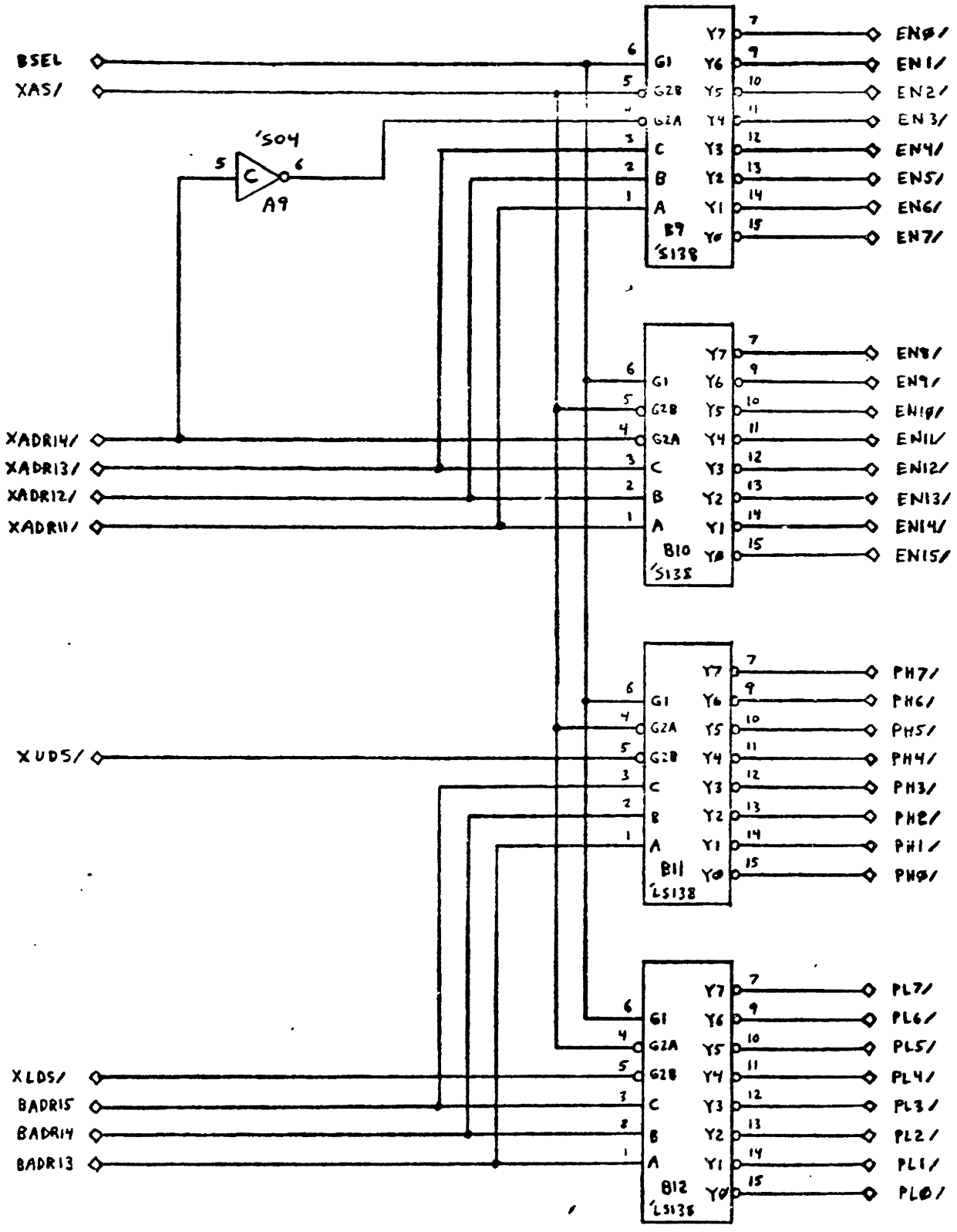

B3
DOUT/
26S10 A3
26S10 A4
26S10 A5
E
I3
I2
I1
I0
B3
B2
B1
B0
DATA15
DATA14
DATA13
DATA12
DATA11
DATA10
DATA9
DATA8
DATA7
DATA6
DATA5
DATA4
DIN/
1Y
2Y
3Y
4Y
5Y
6Y
G2
G1
A
1A
2A
3A
4A
5A
6A
B4
B5
LS368
XDAT15/
XDAT14/
XDAT13/
XDAT12/
XDAT11/
XDAT10/
XDAT9/
XDAT8/
XDAT7/
XDAT6/
XDAT5/
XDAT4/


B4
PUP4
MCLK/
'LS14
A
A8
IN
BB
TTLDM-25
'S11
B
B7
BCLK
'S04
A
A9
BCLK/
MRD/
F
A9
BRD
B
A9
BOE/
MUDS/
D
A9
BUDS
MLDS/
E
A9
BLDS
DOUT/
26S10
A6
E
I3
I2
I1
I0
B3
B2
B1
B0
26S10
A2
PAROUTH
PAROUTL
DATA3
DATA2
DATA1
DATA0
PARINH
PARINL
DIN/
1Y
2Y
3Y
4Y
5Y
6Y
G2
G1
1A
2A
3A
4A
5A
6A
B6
LS368
XDAT3/
XDAT2/
XDAT1/
XDAT0/
XDATPH/
XDATPL/

# B5


2 3 4 5 6 7
S1 S2 S3 S4 S5 S6
B2
15 14 13 12 11 10
15 16
+5v
4.3K
B1
'585
A7
A3
B3
A2
B2
A1
B1
A0
B0
XADR18/
XADR17/
XADR16/
XADR15/
XADR:9/
XINH/
PUP3
'S11
A
B7
=IN
=OUT
A>B
A<B
BSEL
'S132
C
A12
BADR15
'S04
A13
'26S10
E
A2
IO
B0
BDTACK
MDTACK/
'S00
C
C6
BRD
DOUT/

# B6


BSEL
XAS/
'S04
A9
B9
'S138
B10
'S138
B11
'LS138
B12
'LS138
XADR14/
XADR13/
XADR12/
XADR11/
XUDS/
XLDS/
BADR15
BADR14
BADR13
EN0/
EN1/
EN2/
EN3/
EN4/
EN5/
EN6/
EN7/
EN8/
EN9/
EN10/
EN11/
EN12/
EN13/
EN14/
EN15/
PH7/
PH6/
PH5/
PH4/
PH3/
PH2/
PH1/
PH0/
PL7/
PL6/
PL5/
PL4/
PL3/
PL2/
PL1/
PL0/

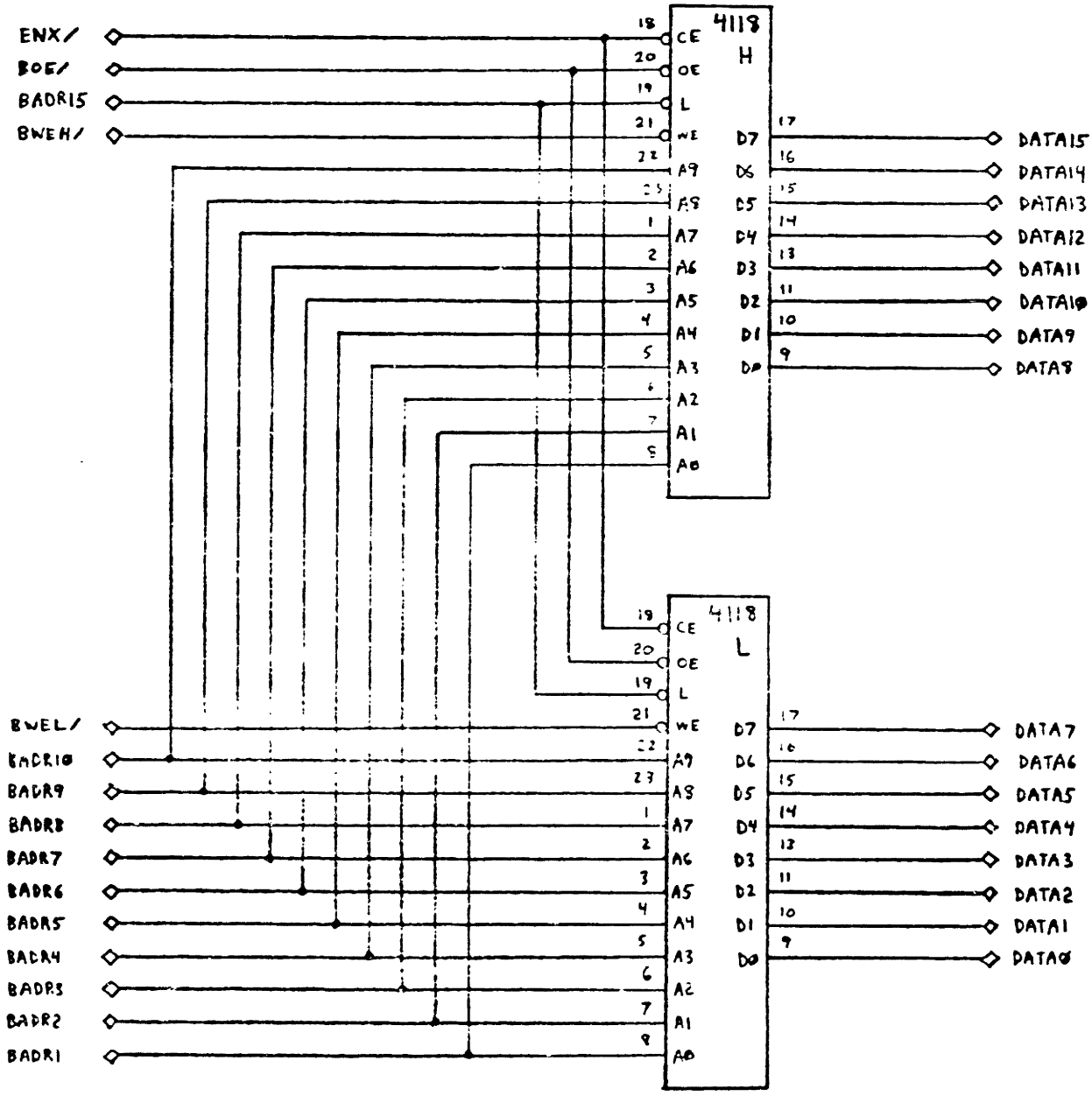
# B7

| | H | L |
|---|---|---|
| EN0 | E1 | D13 |
| EN1 | E4 | E7 |
| EN2 | E4 | E7 |
| EN3 | E6 | E10 |

| | H | L |
|---|---|---|
| EN4 | E7 | E12 |
| EN5 | E9 | E15 |
| EN6 | F3 | E15 |
| EN7 | F4 | F9 |

| | H | L |
|---|---|---|
| EN8 | F6 | F10 |
| EN9 | F8 | F13 |
| EN10 | G1 | F13 |
| EN11 | G3 | F15 |

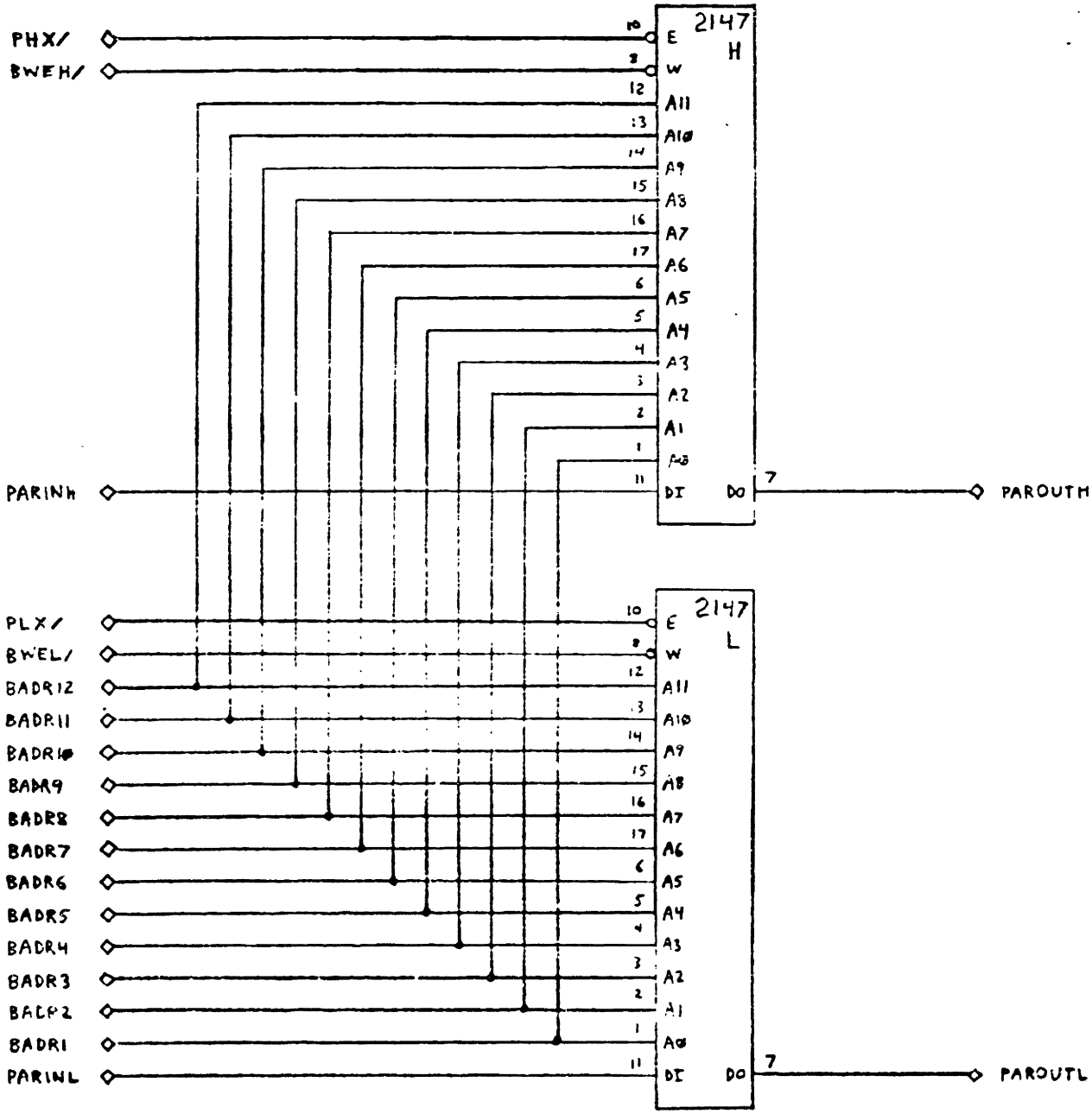
| | H | L |
|---|---|---|
| EN12 | G4 | G10 |
| EN13 | G6 | G12 |
| EN14 | G7 | G13 |
| EN15 | G9 | G15 |



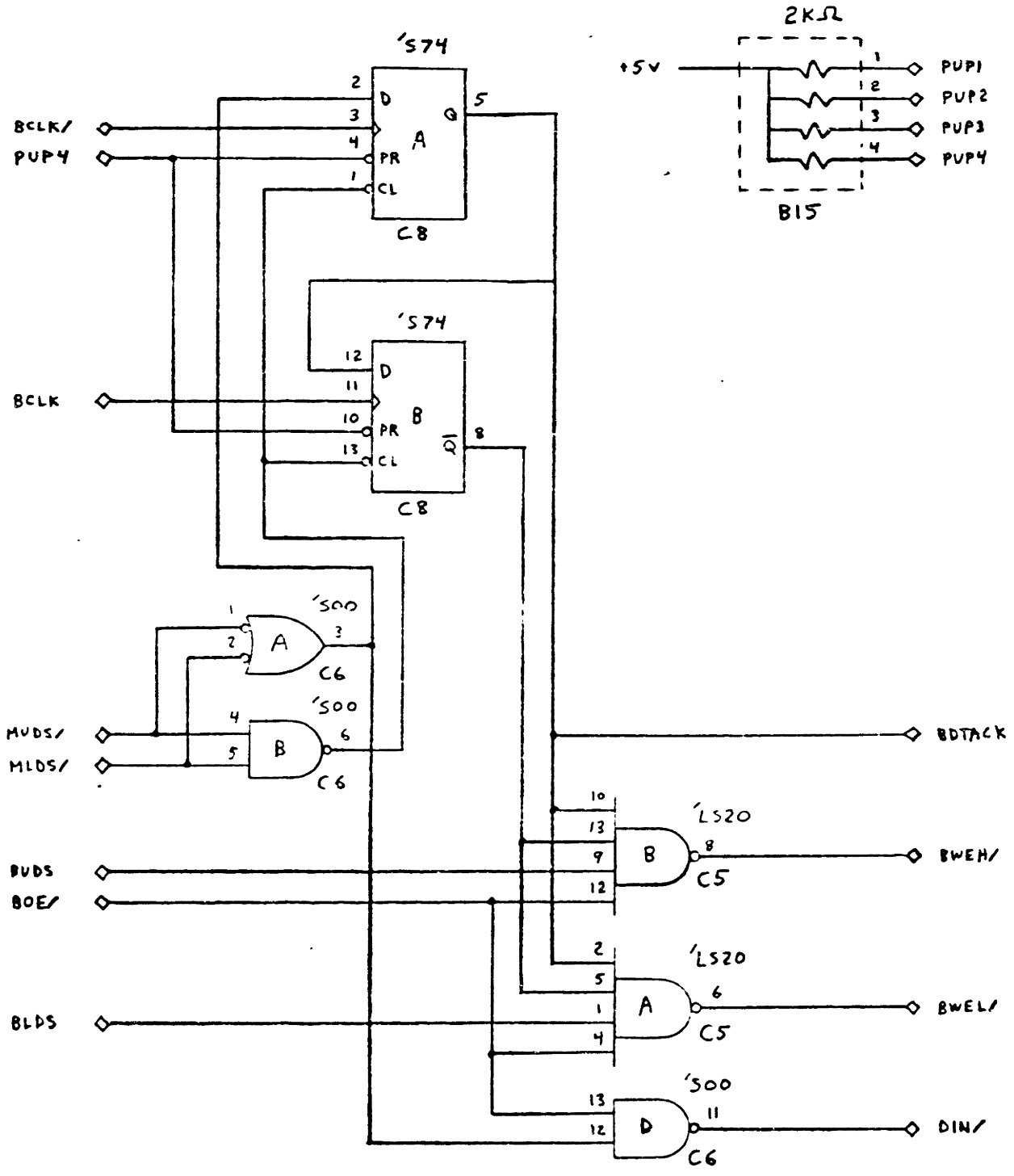
# B8

| | H | L |
|---|---|---|
| P0 | C0 | D1 |
| P1 | C10 | D2 |
| P2 | C11 | D3 |
| P3 | C12 | D4 |

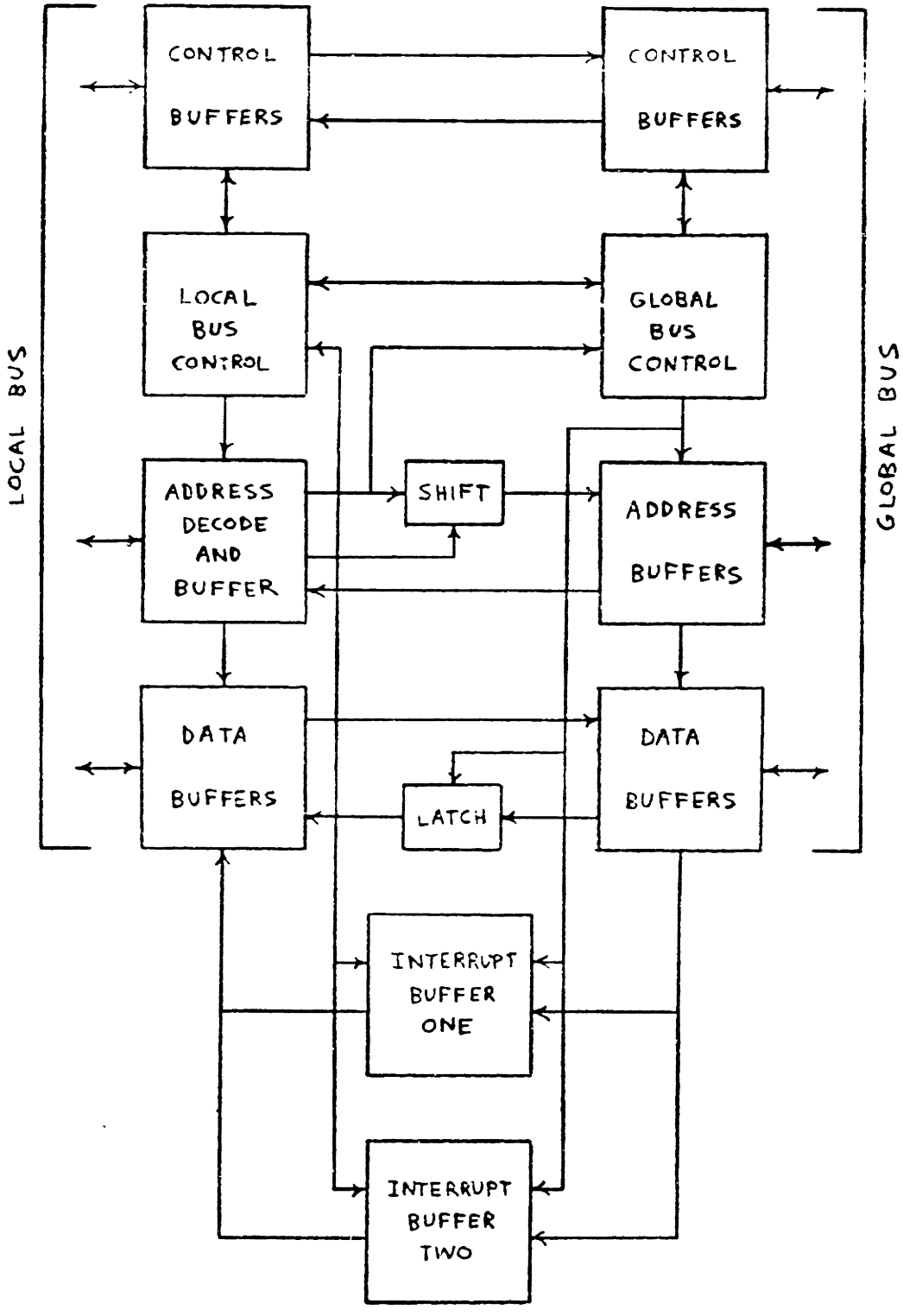
| | H | L |
|---|---|---|
| P4 | D9 | D5 |
| P5 | | |
| P6 | D11 | D7 |
| P7 | D12 | D8 |

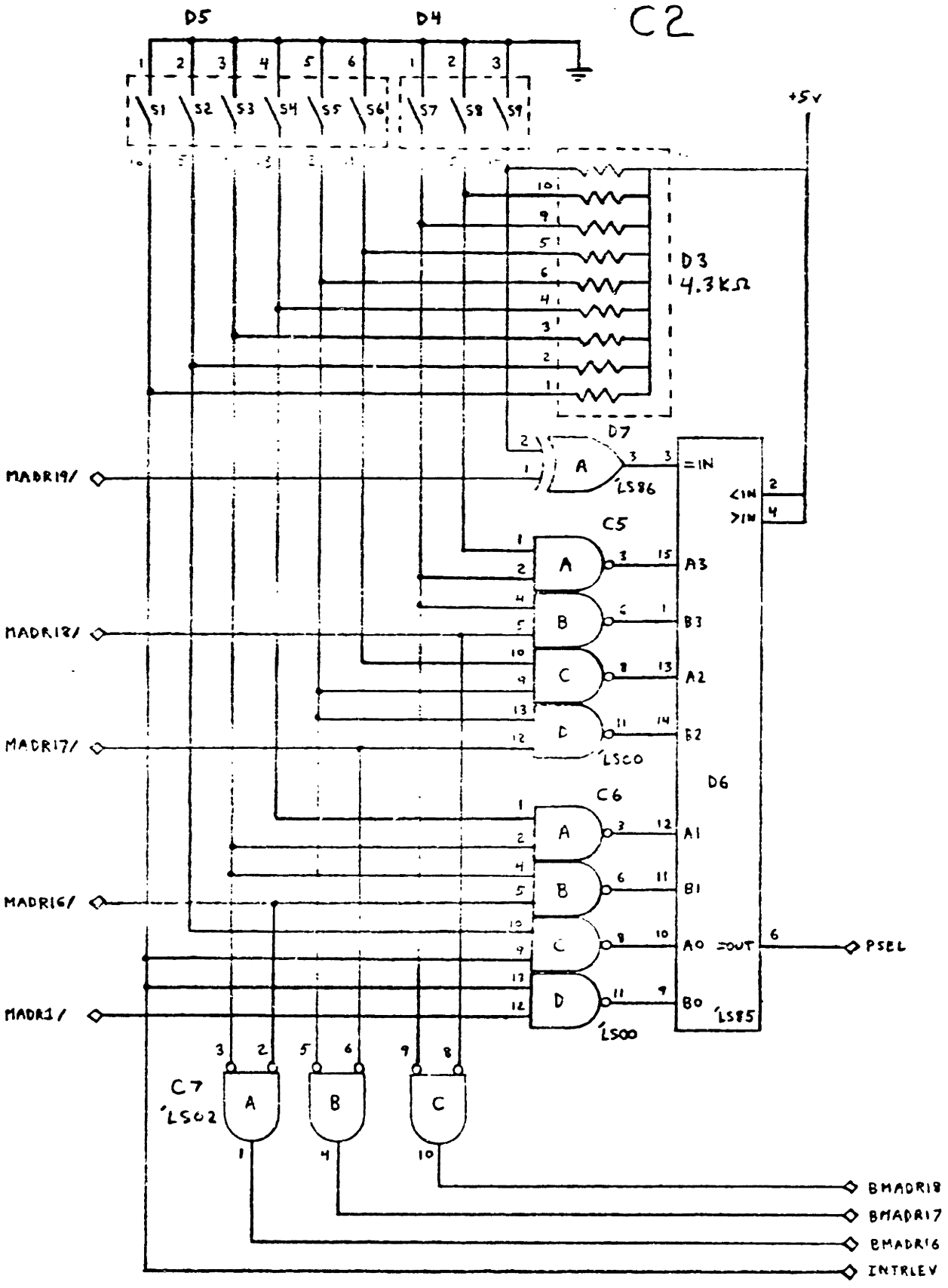


# B9

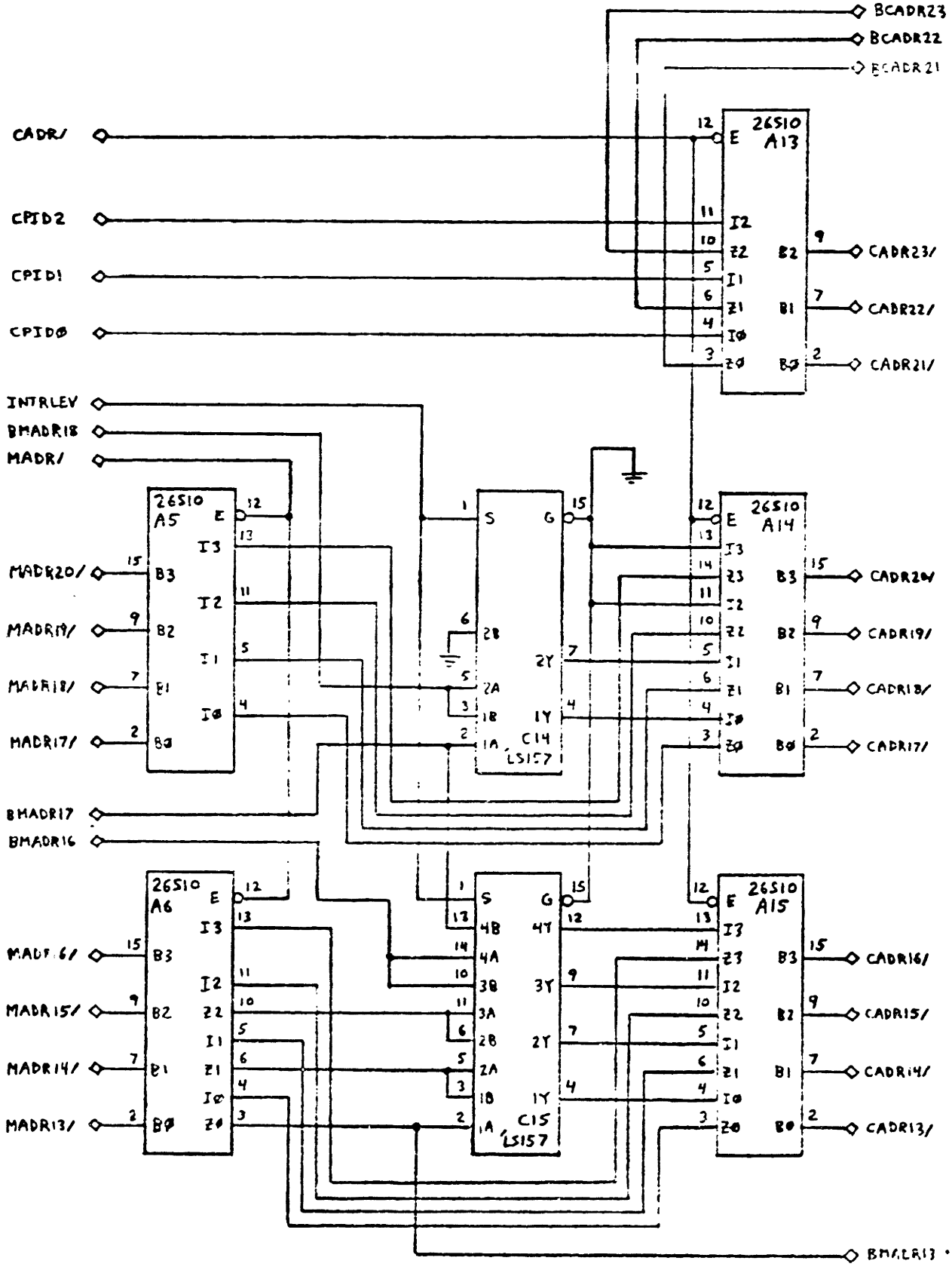

2KΩ
+5V
PUP1
PUP2
PUP3
PUP4
B15
'S74
BCLK/
PUP4
A
C8
'S74
BCLK
B
C8
'S00
A
C6
'S00
MUDS/
MLDS/
B
C6
BDTACK
'LS20
BUDS
BOE/
B
C5
BWEH/
'LS20
BLDS
A
C5
BWEL/
'S00
D
C6
DIN/

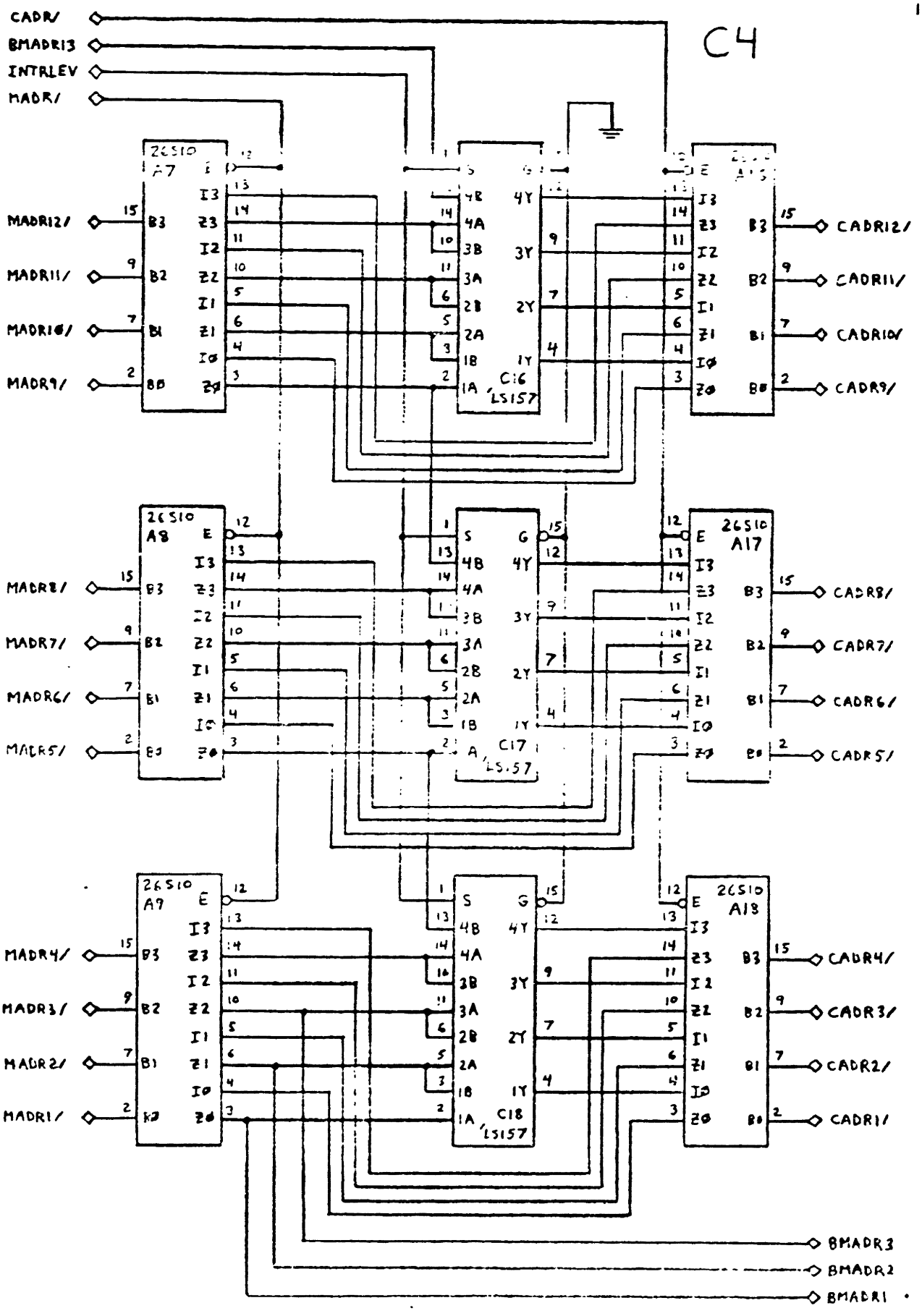
# CI - SWITCH


CONTROL BUFFERS
CONTROL BUFFERS
LOCAL BUS CONTROL
GLOBAL BUS CONTROL
LOCAL BUS
GLOBAL BUS
ADDRESS DECODE AND BUFFER
SHIFT
ADDRESS BUFFERS
DATA BUFFERS
LATCH
DATA BUFFERS
INTERRUPT BUFFER ONE
INTERRUPT BUFFER TWO

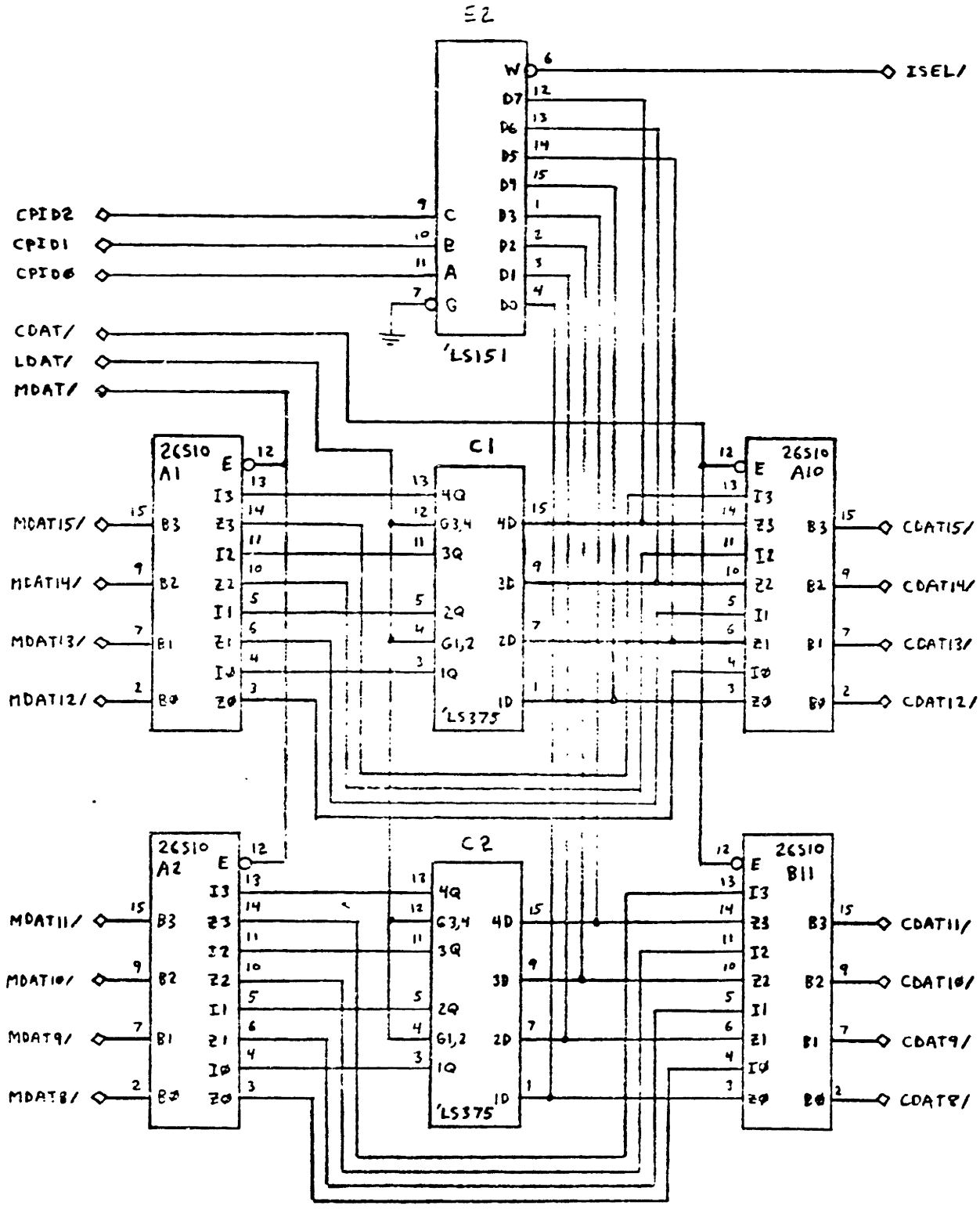

C2
D5
D4
S1
S2
S3
S4
S5
S6
S7
S8
S9
+5V
D3
4.3kΩ
D7
A
'LS86
=IN
<IN
>IN
MADR19/
MADR18/
MADR17/
MADR16/
MADR1/
C5
B
C
D
'LS00
D6
A3
B3
A2
B2
C6
A1
B1
A0
B0
=OUT
'LS85
PSEL
C7
'LS02
BMADR18
BMADR17
BMADR16
INTRLEV

# C3

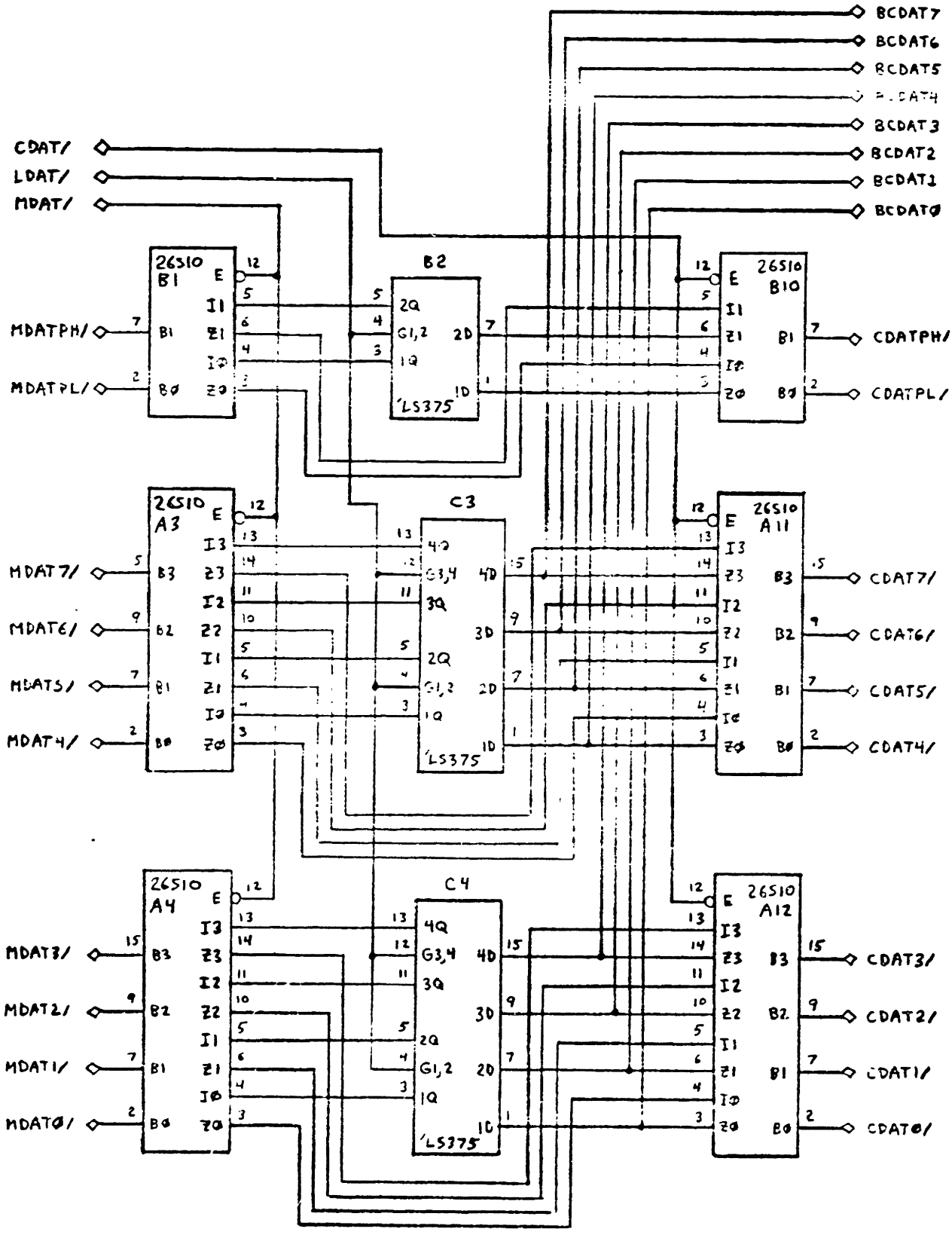

BCADR23
BCADR22
BCADR21
CADR/
CPID2
CPID1
CPID0
INTRLEV
BMADR18
MADR/
26S10 A13
26S10 A14
26S10 A15
26S10 A5
26S10 A6
C14 LS157
C15 LS157
CADR23/
CADR22/
CADR21/
CADR20/
CADR19/
CADR18/
CADR17/
CADR16/
CADR15/
CADR14/
CADR13/
MADR20/
MADR19/
MADR18/
MADR17/
MADR16/
MADR15/
MADR14/
MADR13/
BMADR17
BMADR16
BMADR13


C4
CADR/
BMADR13
INTRLEV
MADR/
26S10
A7
A8
A9
A16
A17
A18
MADR12/
MADR11/
MADR10/
MADR9/
MADR8/
MADR7/
MADR6/
MADR5/
MADR4/
MADR3/
MADR2/
MADR1/
CADR12/
CADR11/
CADR10/
CADR9/
CADR8/
CADR7/
CADR6/
CADR5/
CADR4/
CADR3/
CADR2/
CADR1/
C16
C17
C18
LS157
BMADR3
BMADR2
BMADR1

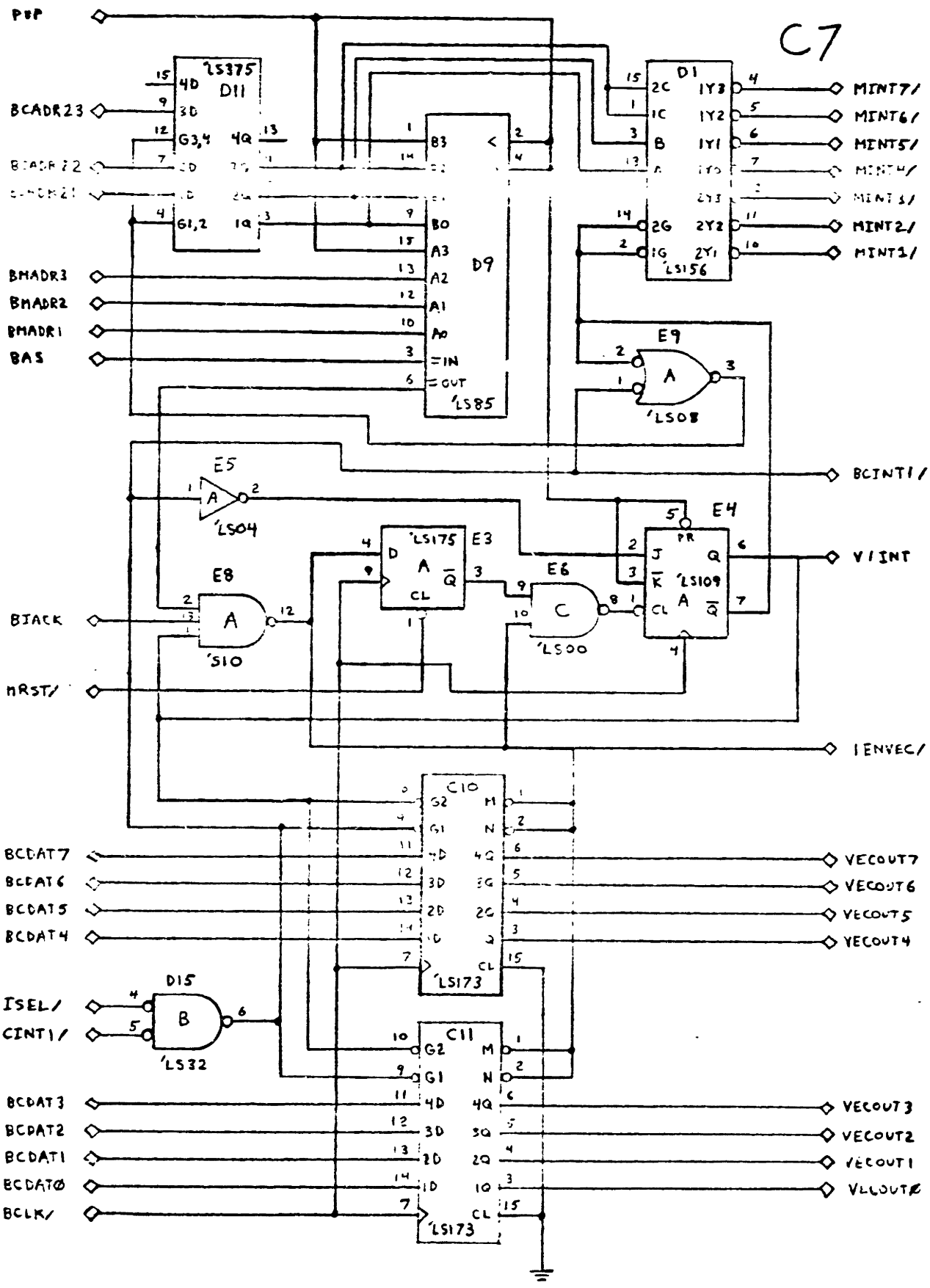
# C5


E2
'LS151
W
D7
D6
D5
D4
D3
D2
D1
D0
C
B
A
G
ISEL/
CPID2
CPID1
CPID0
CDAT/
LDAT/
MDAT/
C1
'LS375
4Q
3Q
2Q
1Q
4D
3D
2D
1D
G3,4
G1,2
26S10
A1
A10
A2
B11
E
I3
I2
I1
I0
Z3
Z2
Z1
Z0
B3
B2
B1
B0
MDAT15/
MDAT14/
MDAT13/
MDAT12/
MDAT11/
MDAT10/
MDAT9/
MDAT8/
CDAT15/
CDAT14/
CDAT13/
CDAT12/
CDAT11/
CDAT10/
CDAT9/
CDAT8/

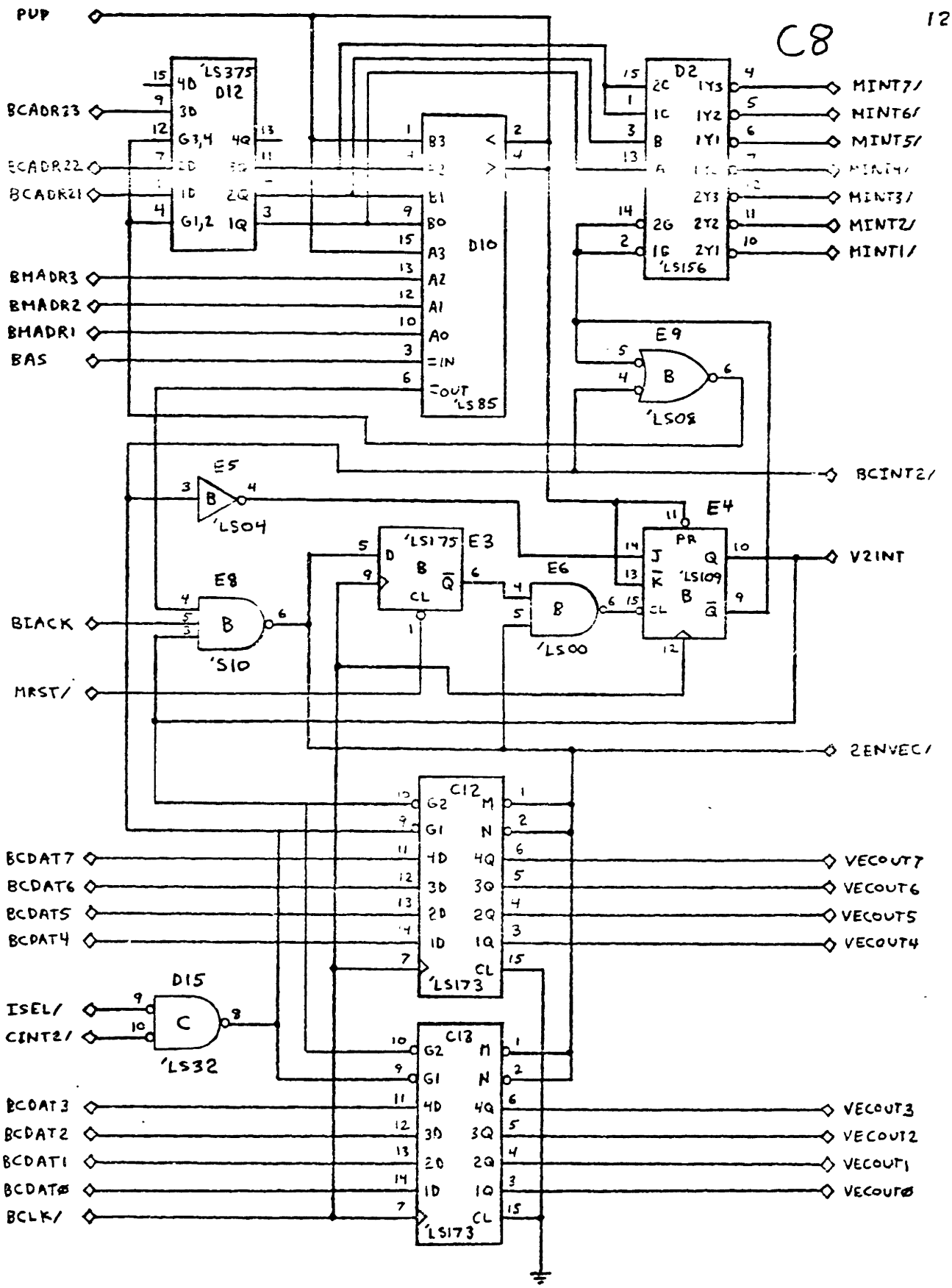
# C6


BCDAT7
BCDAT6
BCDAT5
BCDAT4
BCDAT3
BCDAT2
BCDAT1
BCDAT0
CDAT/
LDAT/
MDAT/
26S10 B1
B2
26S10 B10
MDATPH/
MDATPL/
CDATPH/
CDATPL/
'LS375
26S10 A3
C3
26S10 A11
MDAT7/
MDAT6/
MDAT5/
MDAT4/
CDAT7/
CDAT6/
CDAT5/
CDAT4/
26S10 A4
C4
26S10 A12
MDAT3/
MDAT2/
MDAT1/
MDAT0/
CDAT3/
CDAT2/
CDAT1/
CDAT0/

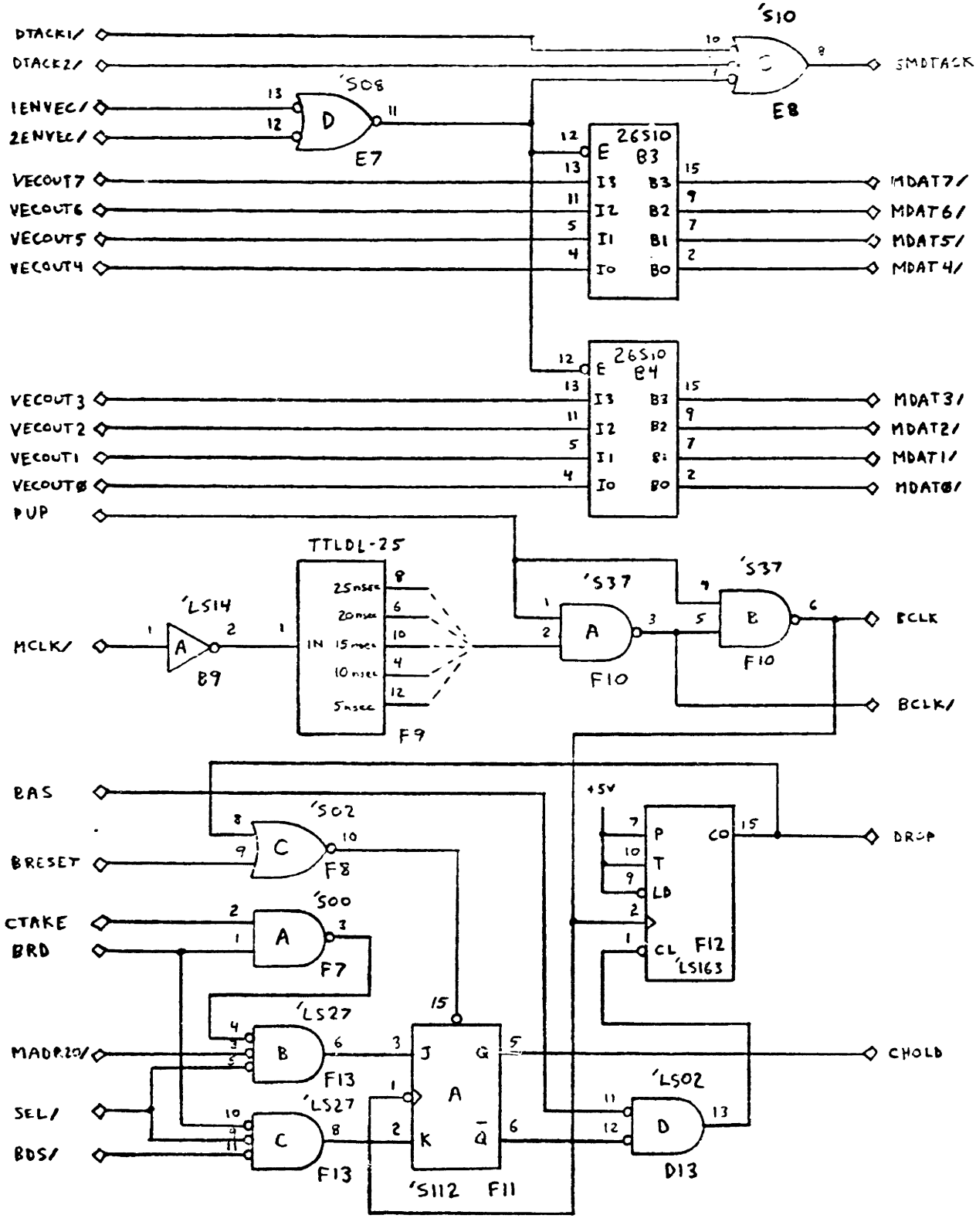
C7


PSP
BCADR23
BCADR22
BCADR21
BMADR3
BMADR2
BMADR1
BAS
'LS375
D11
4D
3D
G3,4
4Q
G1,2
1Q
D9
B3
B0
A3
A2
A1
A0
=IN
=OUT
'LS85
D1
2C
1C
B
A
1Y3
1Y2
1Y1
2Y3
2Y2
2Y1
2G
1G
'LS156
MINT7/
MINT6/
MINT5/
MINT4/
MINT3/
MINT2/
MINT1/
E9
A
'LS08
BCINT1/
E5
A
'LS04
E4
PR
J
K
Q
CL
'LS109
V1INT
E3
'LS175
D
CL
E8
A
'S10
E6
C
'LS00
BIACK
MRST/
IENVEC/
C10
G2
G1
M
N
4D
3D
2D
1D
4Q
3Q
2Q
1Q
'LS173
BCDAT7
BCDAT6
BCDAT5
BCDAT4
VECOUT7
VECOUT6
VECOUT5
VECOUT4
D15
B
'LS32
ISEL/
CINT1/
C11
BCDAT3
BCDAT2
BCDAT1
BCDAT0
BCLK/
VECOUT3
VECOUT2
VECOUT1
VECOUT0

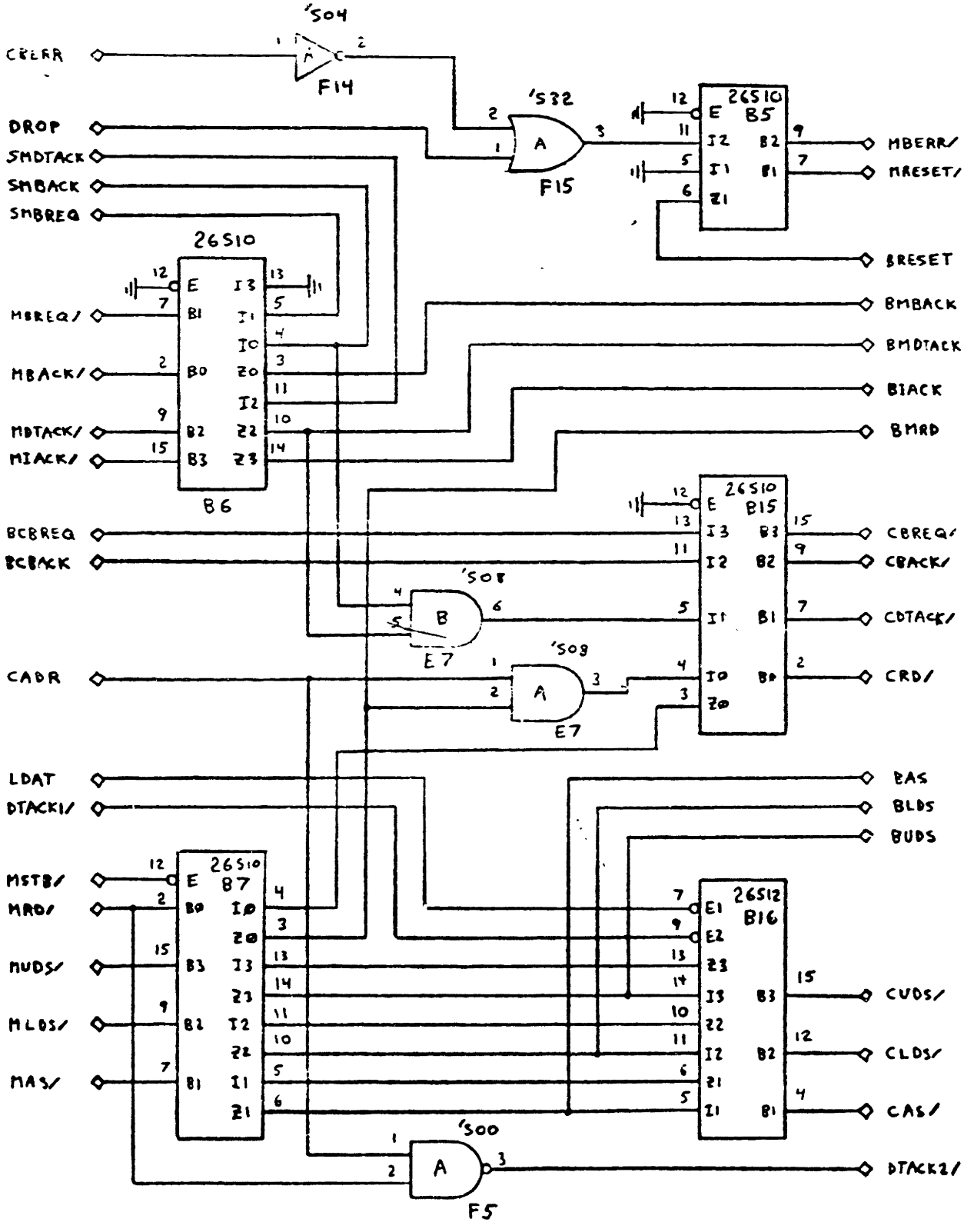
C8

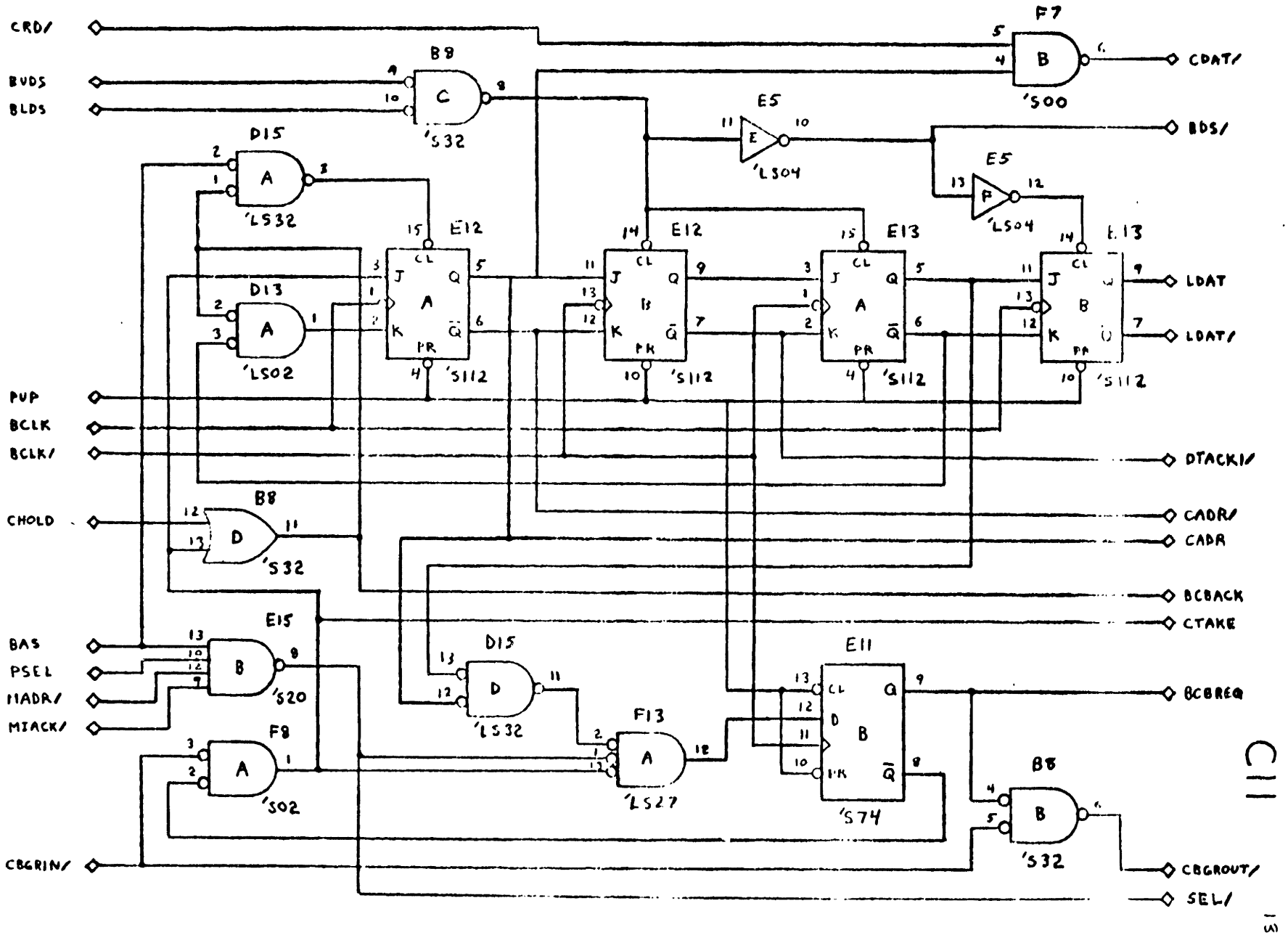

PUP
'LS375 D12
BCADR23
BCADR22
BCADR21
BMADR3
BMADR2
BMADR1
BAS
D10
'LS85
D2
'LS156
MINT7/
MINT6/
MINT5/
MINT4/
MINT3/
MINT2/
MINT1/
E9
'LS08
BCINT2/
E5
'LS04
E4
'LS109
V2INT
'LS175 E3
E8
'S10
BIACK
MRST/
E6
'LS00
2ENVEC/
C12
'LS173
BCDAT7
BCDAT6
BCDAT5
BCDAT4
VECOUT7
VECOUT6
VECOUT5
VECOUT4
D15
'LS32
ISEL/
CINT2/
C18
'LS173
BCDAT3
BCDAT2
BCDAT1
BCDAT0
BCLK/
VECOUT3
VECOUT2
VECOUT1
VECOUT0

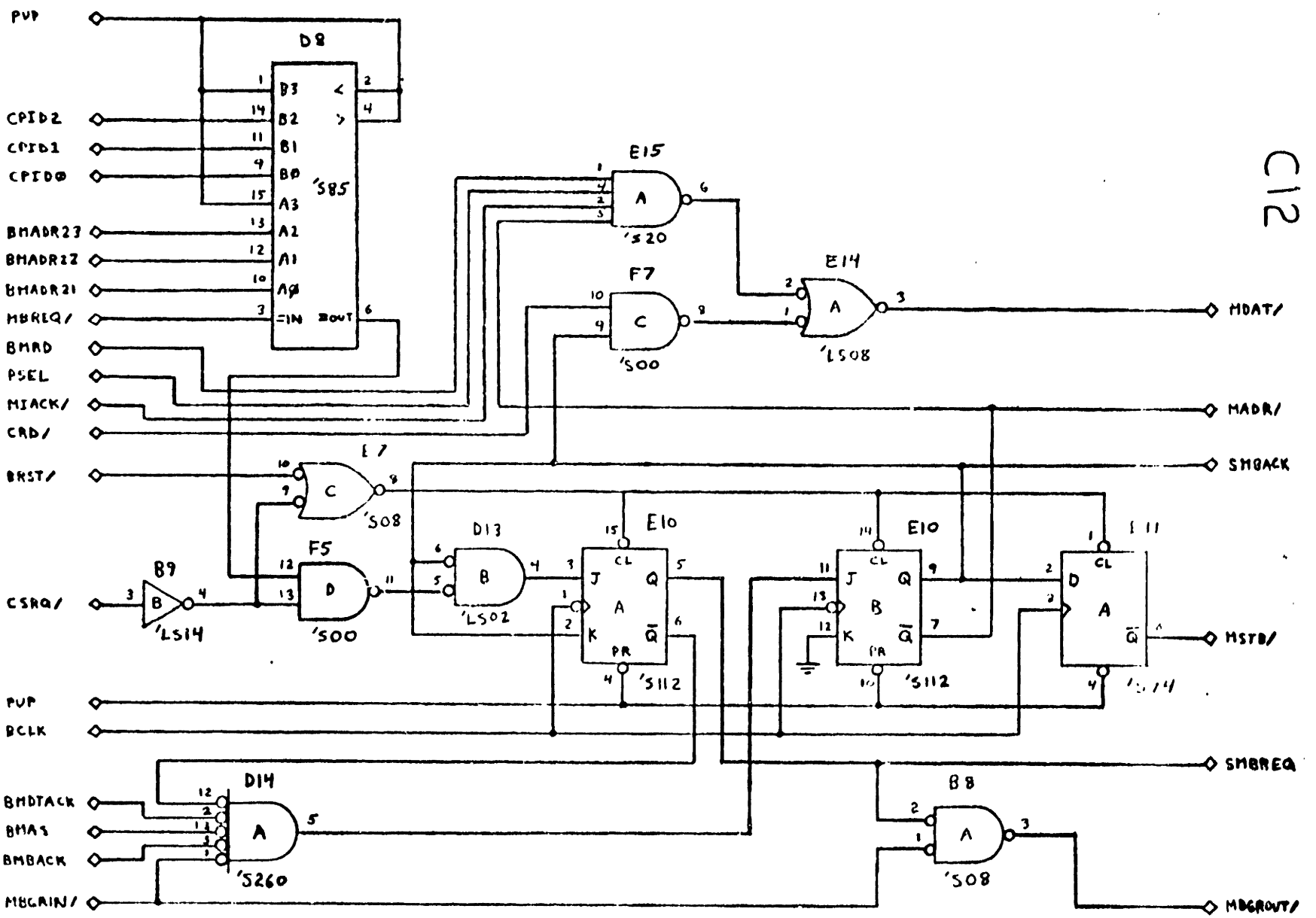
# C9


DTACK1/
DTACK2/
'S10
SMDTACK
E8
1ENVEC/
2ENVEC/
'S08
E7
VECOUT7
VECOUT6
VECOUT5
VECOUT4
26S10
B3
MDAT7/
MDAT6/
MDAT5/
MDAT4/
VECOUT3
VECOUT2
VECOUT1
VECOUT0
26S10
B4
MDAT3/
MDAT2/
MDAT1/
MDAT0/
PUP
TTLDL-25
25nsec
20nsec
15nsec
10nsec
5nsec
F9
MCLK/
'LS14
B9
'S37
F10
BCLK
BCLK/
BAS
BRESET
'S02
F8
CTAKE
BRD
'S00
F7
+5V
DROP
'LS163
F12
'LS27
MADR20/
F13
SEL/
BDS/
'S112
F11
CHOLD
'LS02
D13

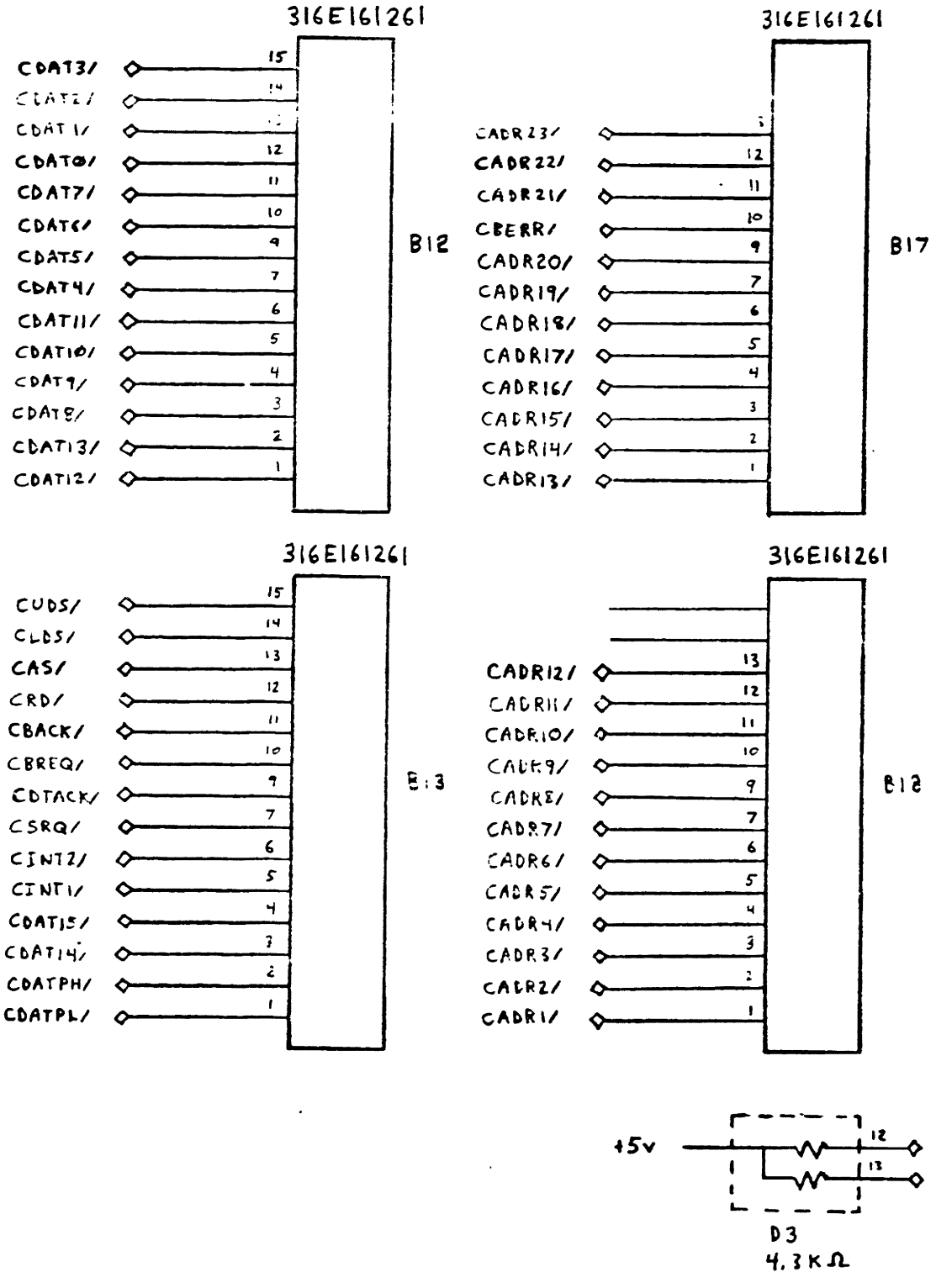
# C10


'S04
CBERR
F14
'S32
DROP
SMDTACK
SMBACK
SMBREQ
F15
26S10
B5
MBERR/
MRESET/
BRESET
26S10
B6
MBREQ/
MBACK/
MDTACK/
MIACK/
BMBACK
BMDTACK
BIACK
BMRD
26S10
B15
BCBREQ
BCBACK
CBREQ/
CBACK/
CDTACK/
CRD/
'S08
E7
'S08
E7
CADR
LDAT
DTACK1/
BAS
BLDS
BUDS
26S10
B7
MSTB/
MRD/
MUDS/
MLDS/
MAS/
26S12
B16
CUDS/
CLDS/
CAS/
'S00
F5
DTACK2/

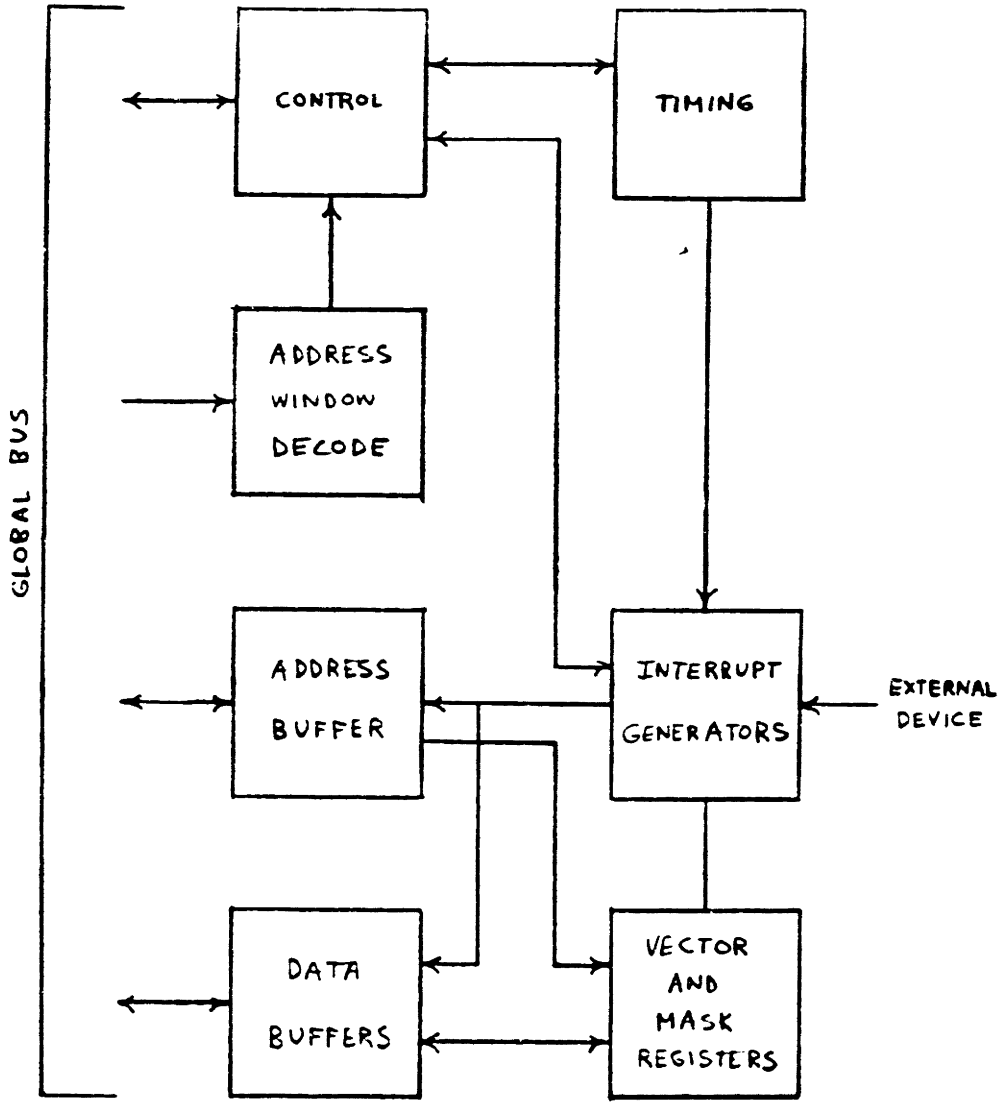

CRD/
BUDS
BLDS
PUP
BCLK
BCLK/
CHOLD
BAS
PSEL
MADR/
MIACK/
CBGRIN/
B8
'S32
F7
'S00
E5
'LS04
D15
'LS32
D13
'LS02
E12
'S112
E13
'S112
B.13
'S112
CDAT/
BDS/
LDAT
LDAT/
DTACK1/
CADR/
CADR
BCBACK
CTAKE
E15
'S20
F8
'S02
D15
'LS32
F13
'LS27
E11
'S74
BCBREQ
B8
'S32
CBGROUT/
SEL/

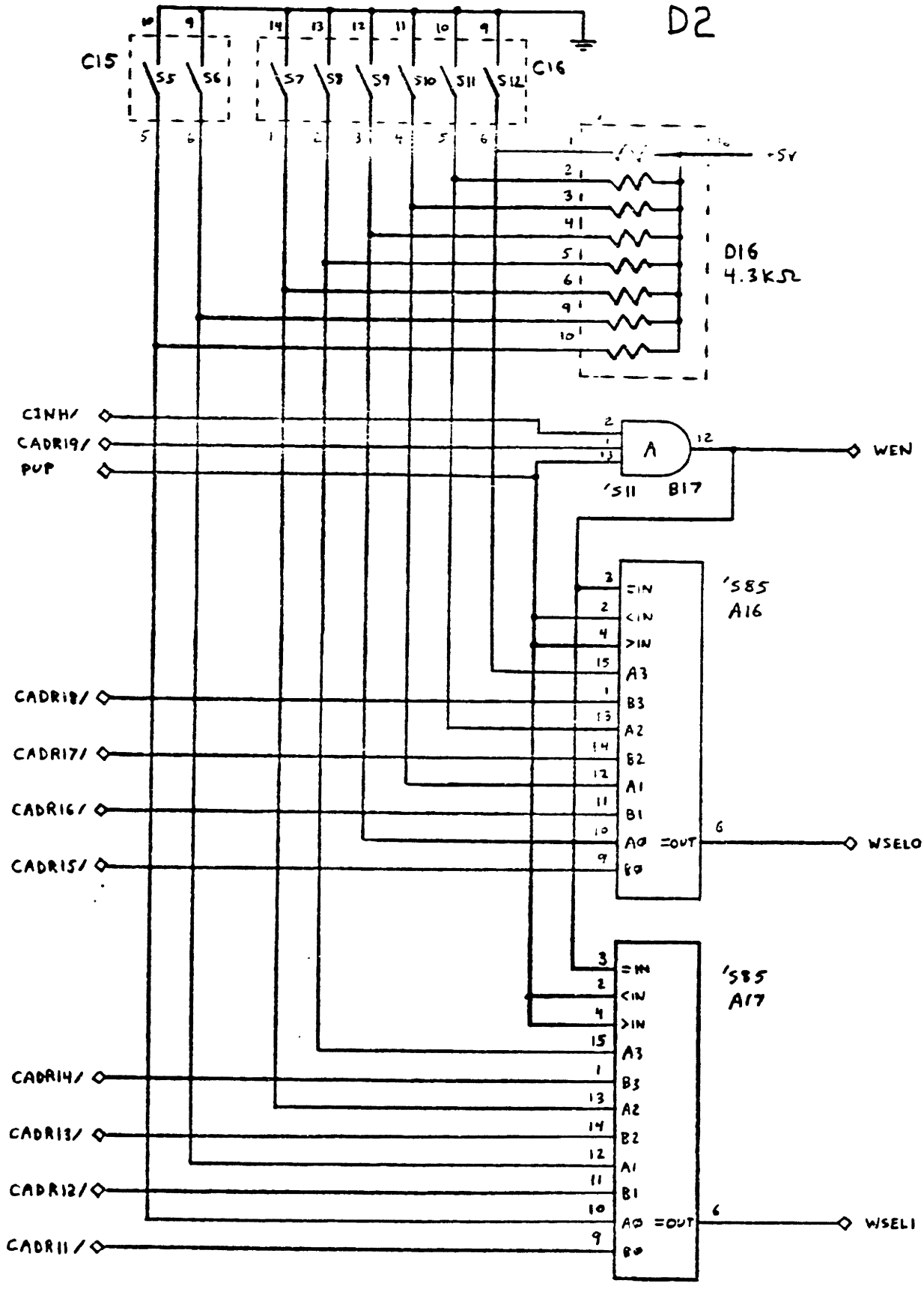

PUP
CPID2
CPID1
CPID0
BMADR23
BMADR22
BMADR21
MBREQ/
BMRD
PSEL
MIACK/
CRD/
BRST/
CSRQ/
PUP
BCLK
BMDTACK
BMAS
BMBACK
MBGRIN/
D8
'S85
E15
'S20
F7
'S00
E14
'LS08
E7
'S08
B9
'LS14
F5
'S00
D13
'LS02
E10
'S112
E10
'S112
E11
'S74
D14
'S260
B8
'S08
MDAT/
MADR/
SMBACK
MSTB/
SMBREQ
MBGROUT/

# C13

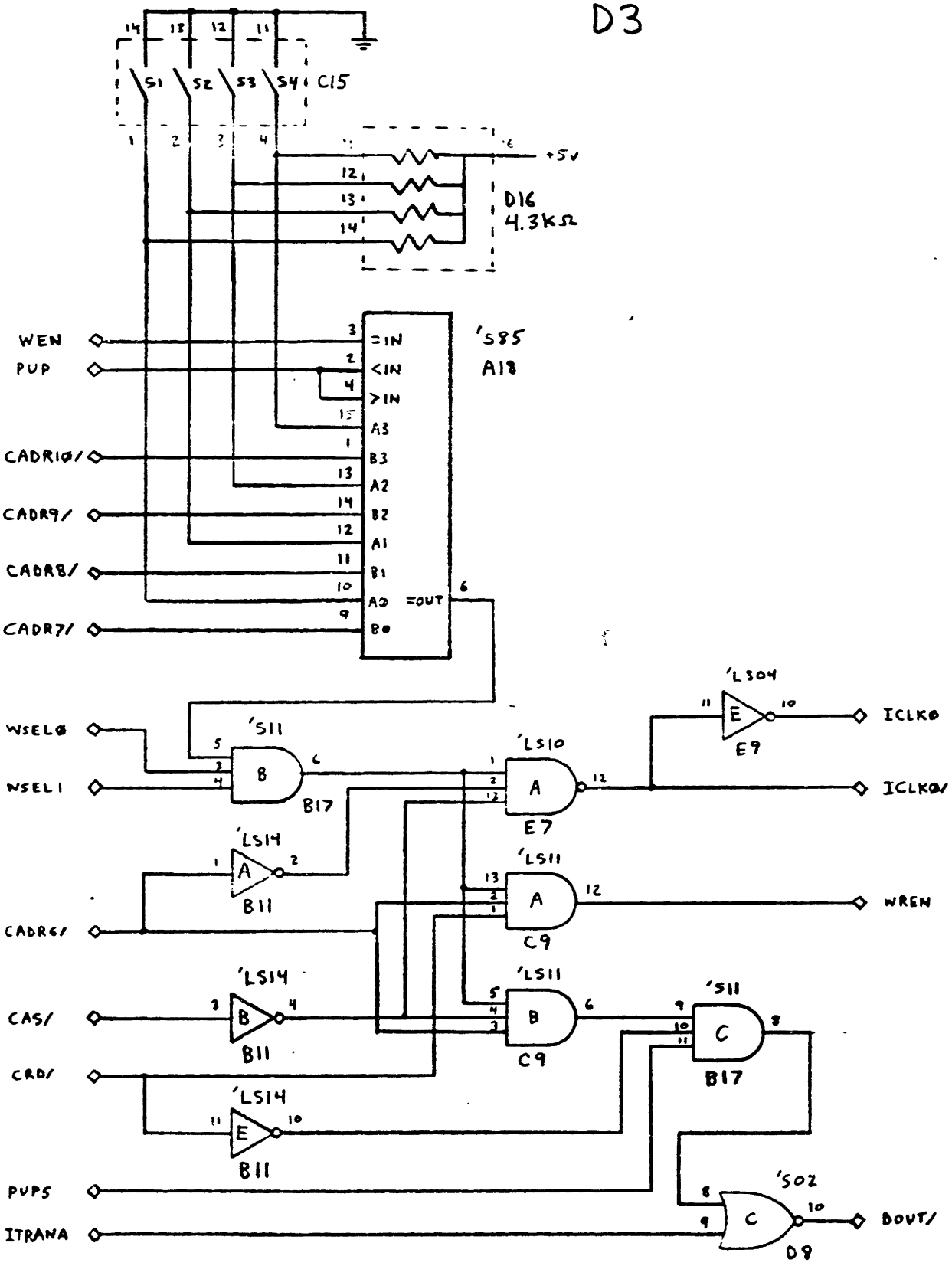

316E161261
CDAT3/
CDAT2/
CDAT1/
CDAT0/
CDAT7/
CDAT6/
CDAT5/
CDAT4/
CDAT11/
CDAT10/
CDAT9/
CDAT8/
CDAT13/
CDAT12/
B12
316E161261
CADR23/
CADR22/
CADR21/
CBERR/
CADR20/
CADR19/
CADR18/
CADR17/
CADR16/
CADR15/
CADR14/
CADR13/
B17
316E161261
CUDS/
CLDS/
CAS/
CRD/
CBACK/
CBREQ/
CDTACK/
CSRQ/
CINT2/
CINT1/
CDAT15/
CDAT14/
CDATPH/
CDATPL/
B13
316E161261
CADR12/
CADR11/
CADR10/
CADR9/
CADR8/
CADR7/
CADR6/
CADR5/
CADR4/
CADR3/
CADR2/
CADR1/
B12
+5v
PUP1
PUP2
D3
4.3KΩ

# DI - IMOD

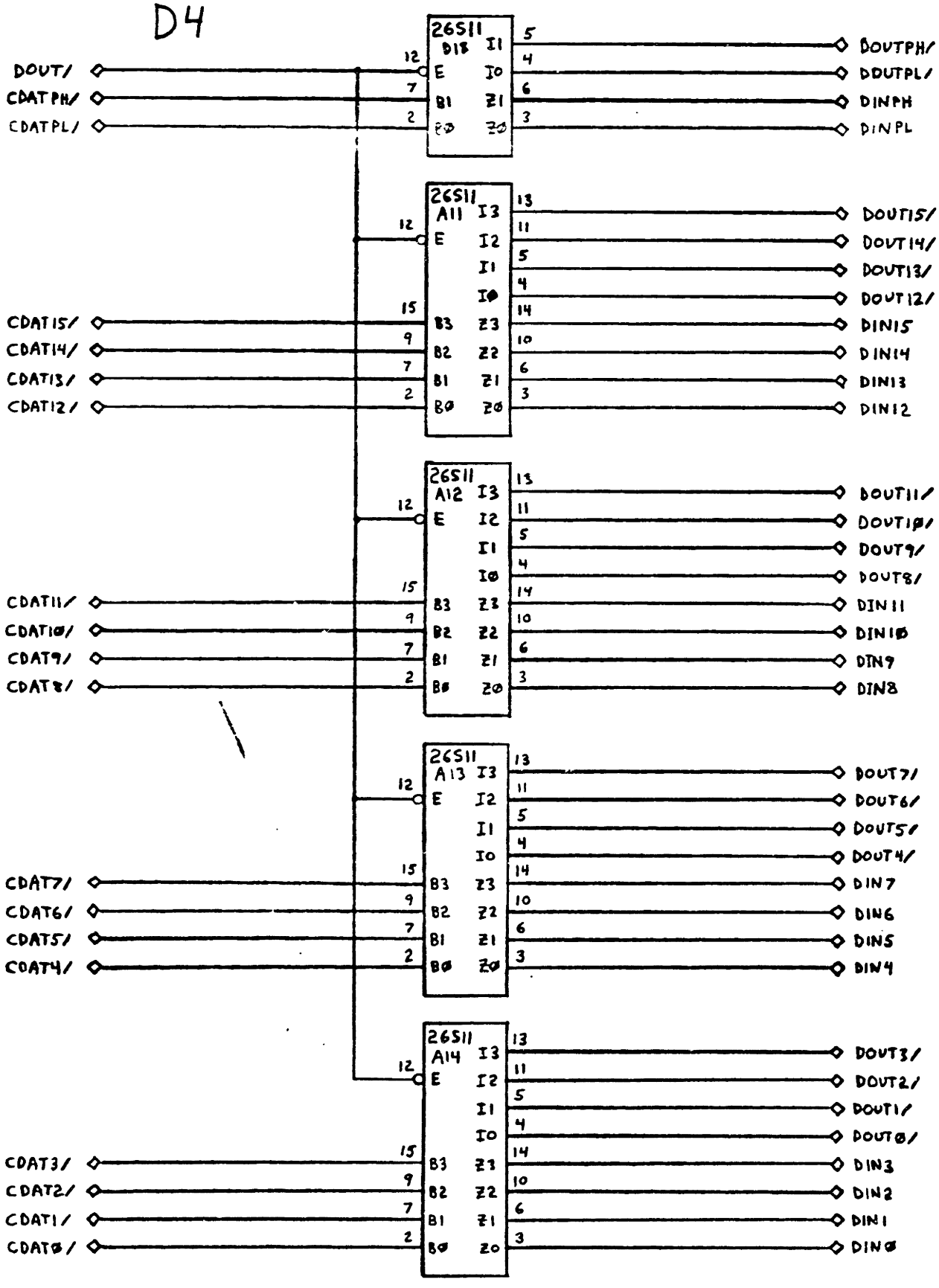

CONTROL
TIMING
ADDRESS WINDOW DECODE
GLOBAL BUS
ADDRESS BUFFER
INTERRUPT GENERATORS
EXTERNAL DEVICE
DATA BUFFERS
VECTOR AND MASK REGISTERS


D2
C15
C16
S5
S6
S7
S8
S9
S10
S11
S12
+5V
D16
4.3KΩ
CINH/
CADR19/
PUP
A
'S11
B17
WEN
'S85
A16
CADR18/
CADR17/
CADR16/
CADR15/
=IN
<IN
>IN
A3
B3
A2
B2
A1
B1
A0
B0
=OUT
WSEL0
'S85
A17
CADR14/
CADR13/
CADR12/
CADR11/
WSEL1

# D3

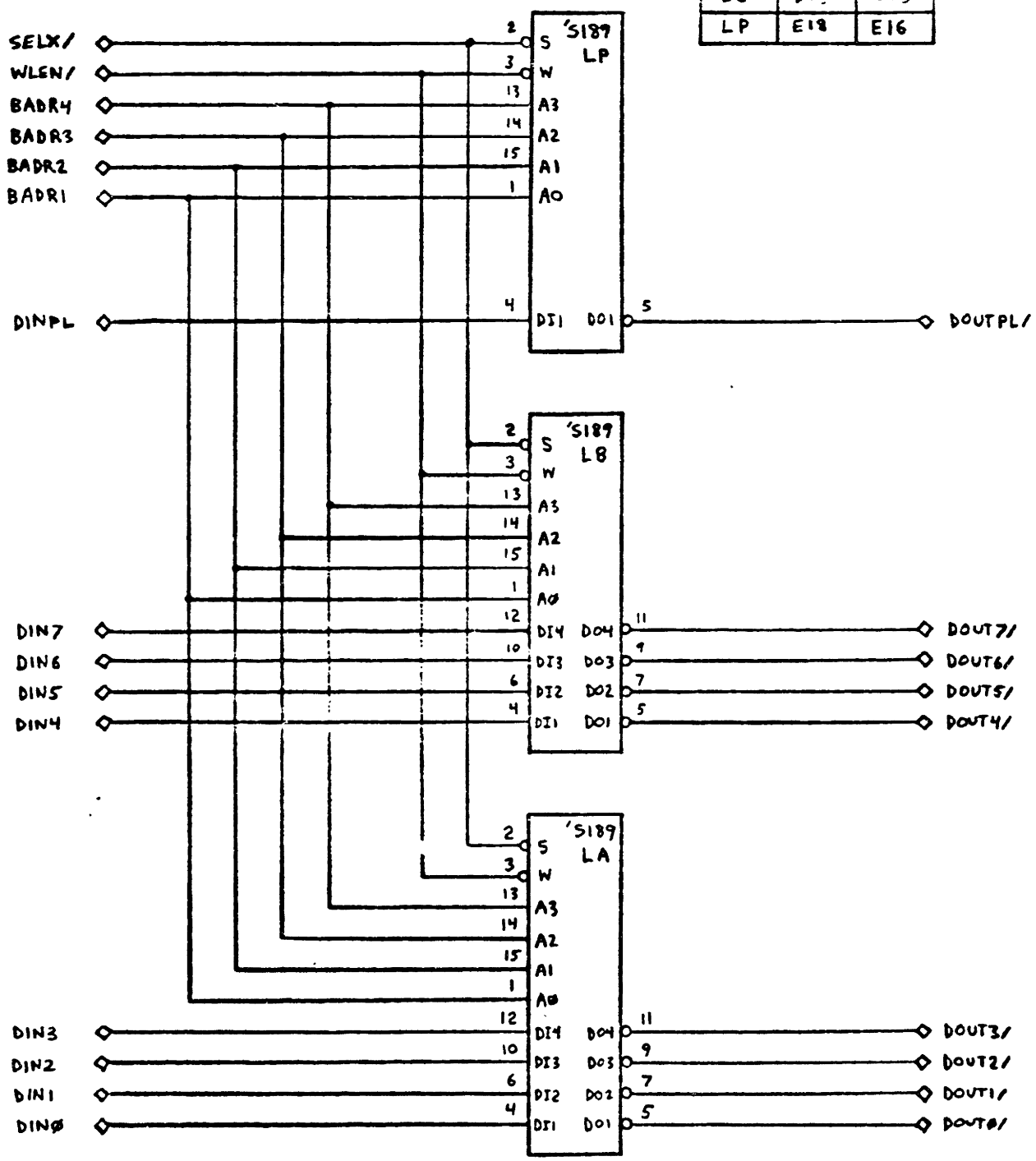

S1
S2
S3
S4
C15
+5V
D16
4.3kΩ
WEN
PUP
CADR10/
CADR9/
CADR8/
CADR7/
'S85
A18
=IN
<IN
>IN
A3
B3
A2
B2
A1
B1
A0
B0
=OUT
WSEL0
WSEL1
'S11
B17
'LS10
E7
'LS04
E9
ICLK0
ICLK0/
'LS14
B11
CADR6/
'LS11
C9
WREN
CAS/
CR0/
PUPS
ITRANA
'S02
D9
DOUT/

# D4


26S11 DIS A11
DOUT/
CDATPH/
CDATPL/
DOUTPH/
DOUTPL/
DINPH
DINPL
26S11 A11
CDAT15/
CDAT14/
CDAT13/
CDAT12/
DOUT15/
DOUT14/
DOUT13/
DOUT12/
DIN15
DIN14
DIN13
DIN12
26S11 A12
CDAT11/
CDAT10/
CDAT9/
CDAT8/
DOUT11/
DOUT10/
DOUT9/
DOUT8/
DIN11
DIN10
DIN9
DIN8
26S11 A13
CDAT7/
CDAT6/
CDAT5/
CDAT4/
DOUT7/
DOUT6/
DOUT5/
DOUT4/
DIN7
DIN6
DIN5
DIN4
26S11 A14
CDAT3/
CDAT2/
CDAT1/
CDAT0/
DOUT3/
DOUT2/
DOUT1/
DOUT0/
DIN3
DIN2
DIN1
DIN0

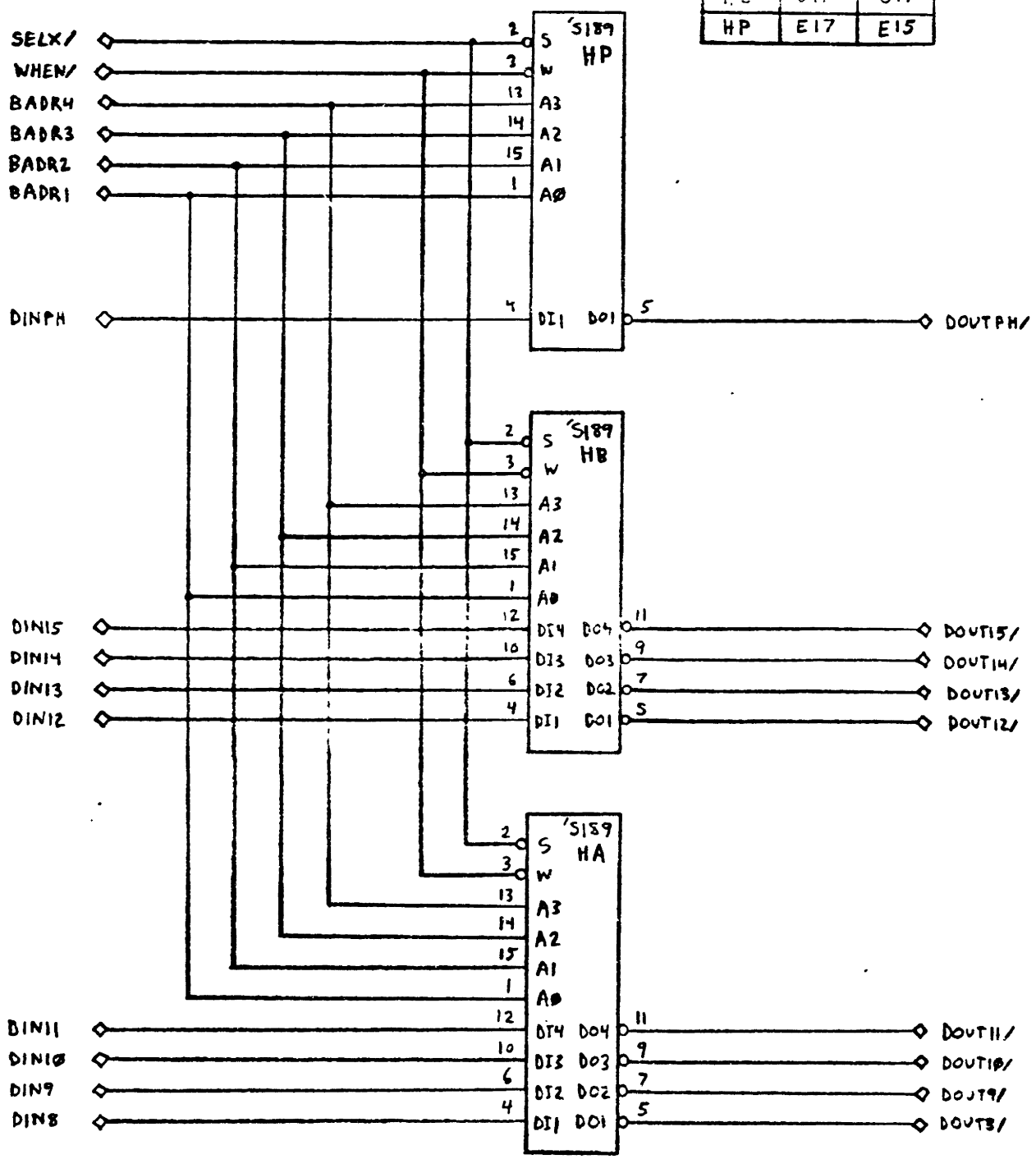
# D5

| | SEL0 | SEL1 |
|---|---|---|
| LA | D14 | C14 |
| LB | D13 | C13 |
| LP | E18 | E16 |

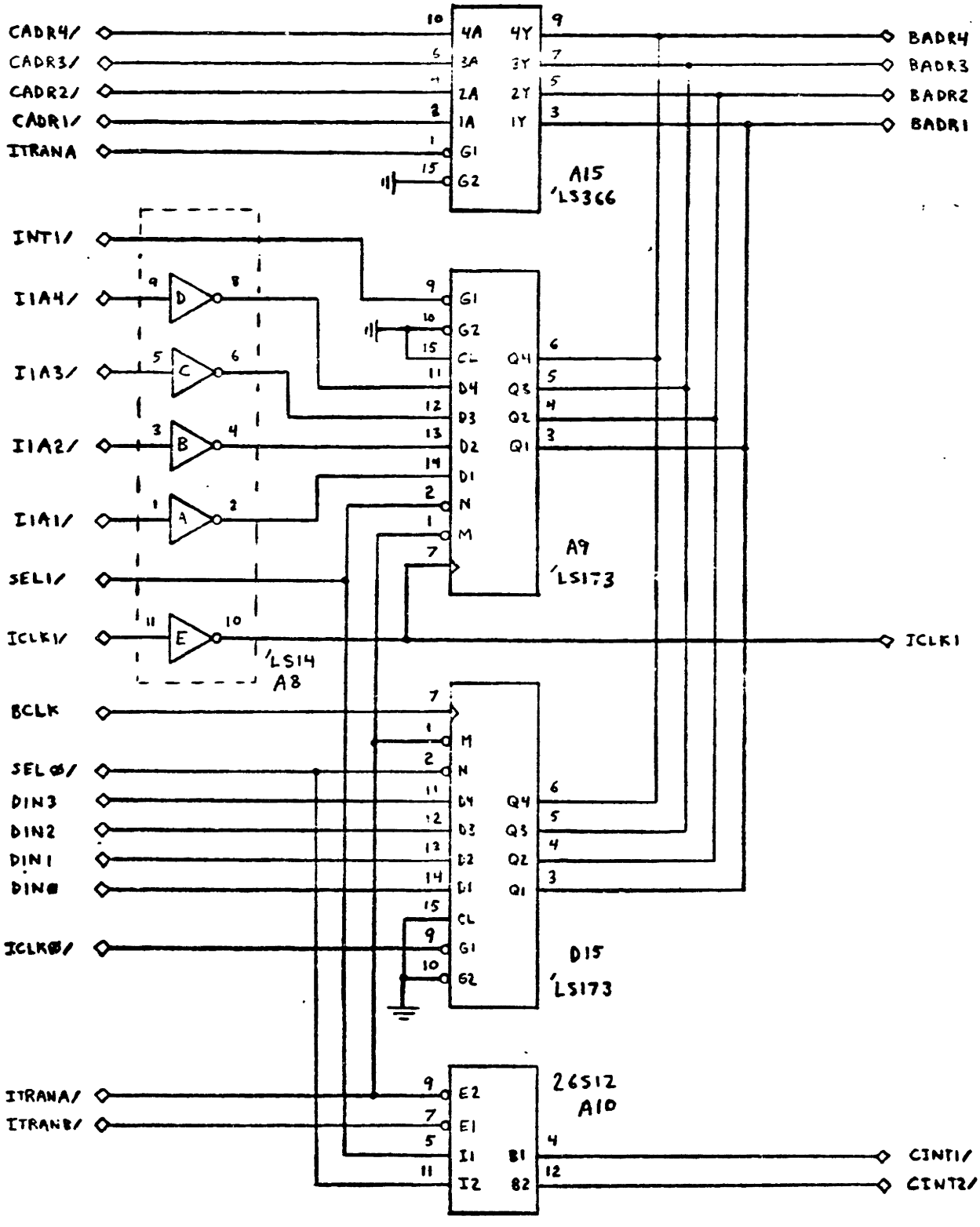


# D6

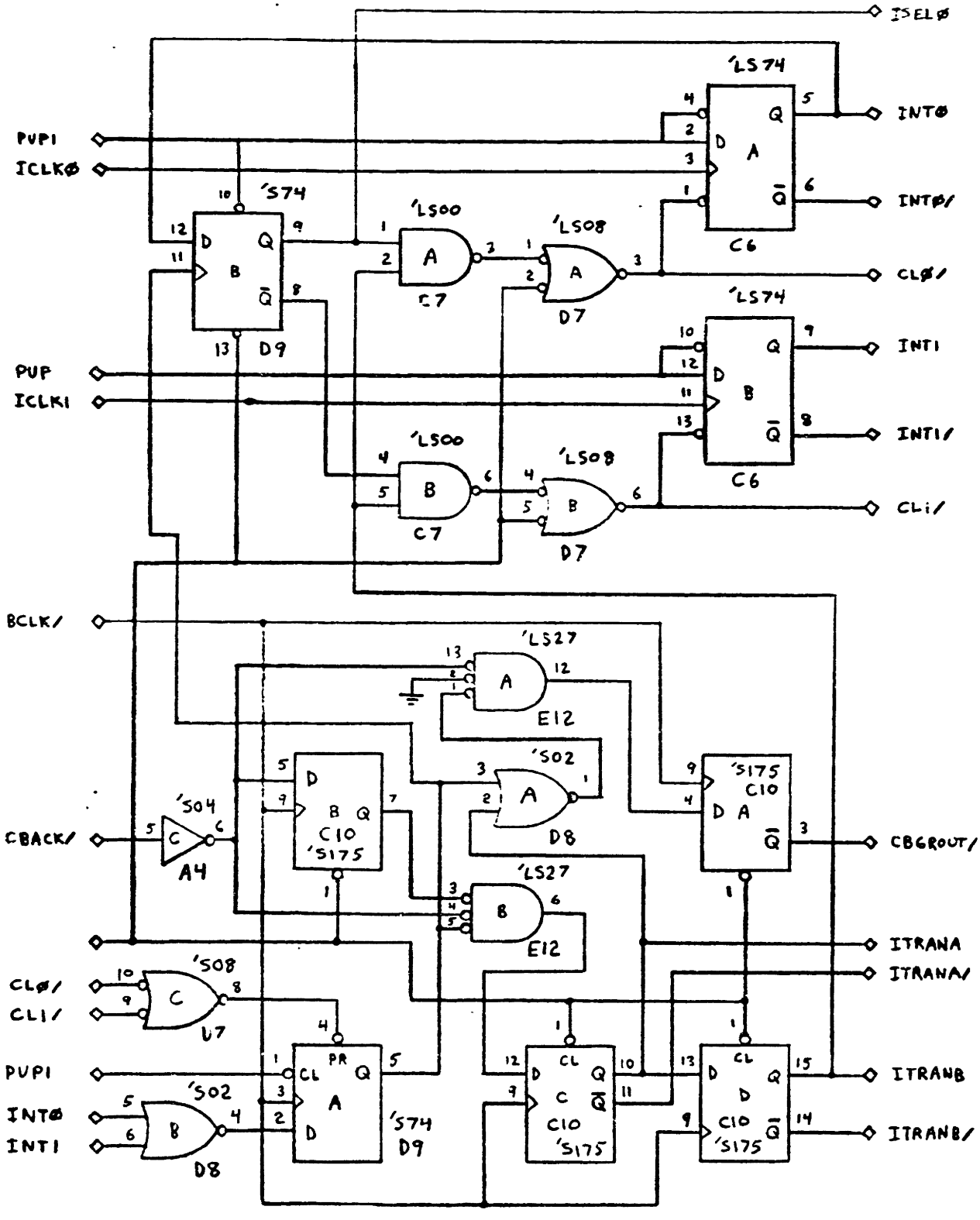
| | SEL0 | SEL1 |
|---|---|---|
| HA | D12 | C12 |
| HB | D11 | C11 |
| HP | E17 | E15 |



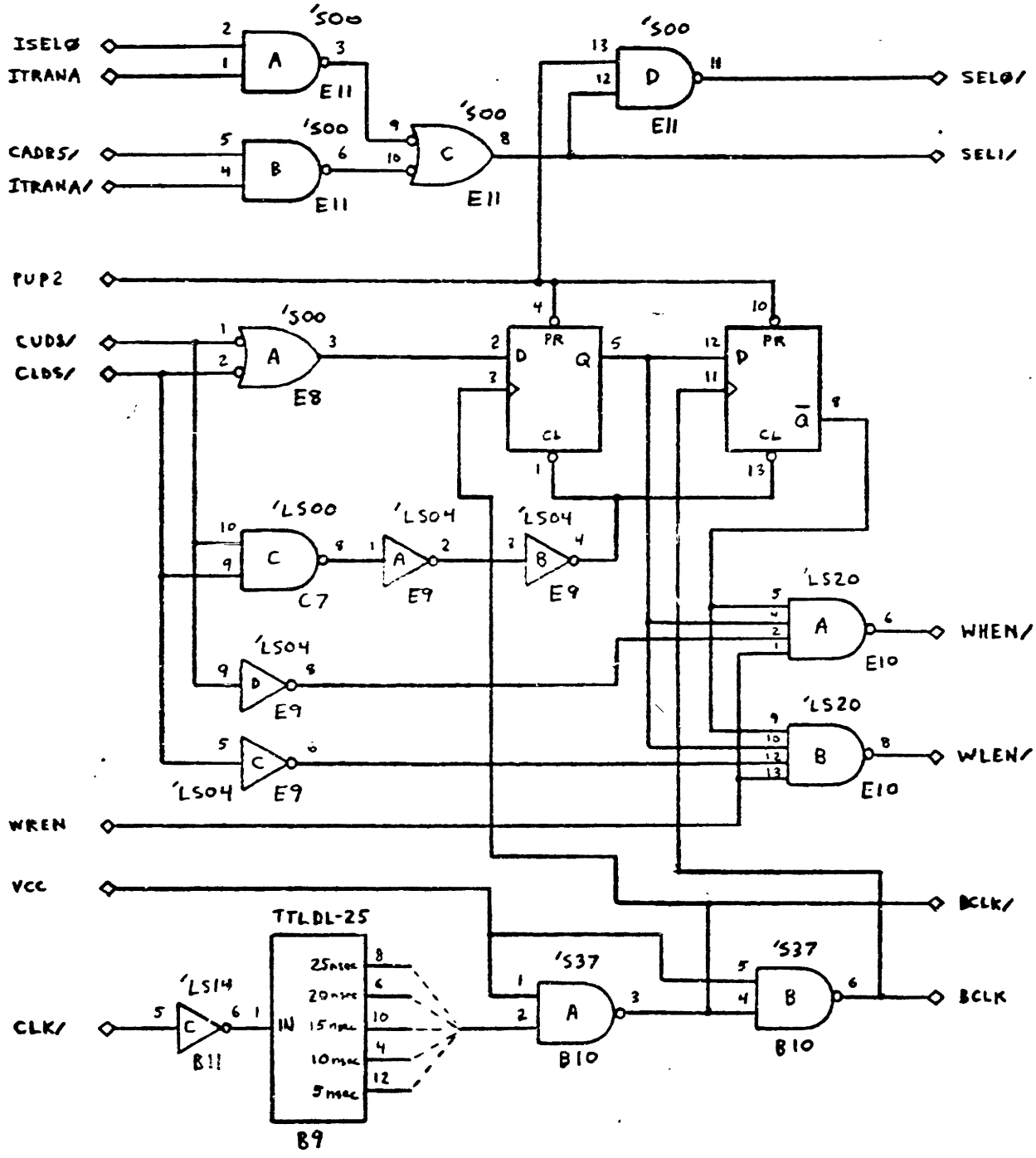
# D7


CADR4/
CADR3/
CADR2/
CADR1/
ITRANA
4A
3A
2A
1A
G1
G2
4Y
3Y
2Y
1Y
BADR4
BADR3
BADR2
BADR1
A15
'LS366
INT1/
I1A4/
I1A3/
I1A2/
I1A1/
SEL1/
ICLK1/
D
C
B
A
E
'LS14
A8
G1
G2
CL
D4
D3
D2
D1
N
M
Q4
Q3
Q2
Q1
A9
'LS173
ICLK1
BCLK
SEL0/
DIN3
DIN2
DIN1
DIN0
ICLK0/
D15
'LS173
ITRANA/
ITRANB/
E2
E1
I1
I2
B1
B2
26S12
A10
CINT1/
CINT2/

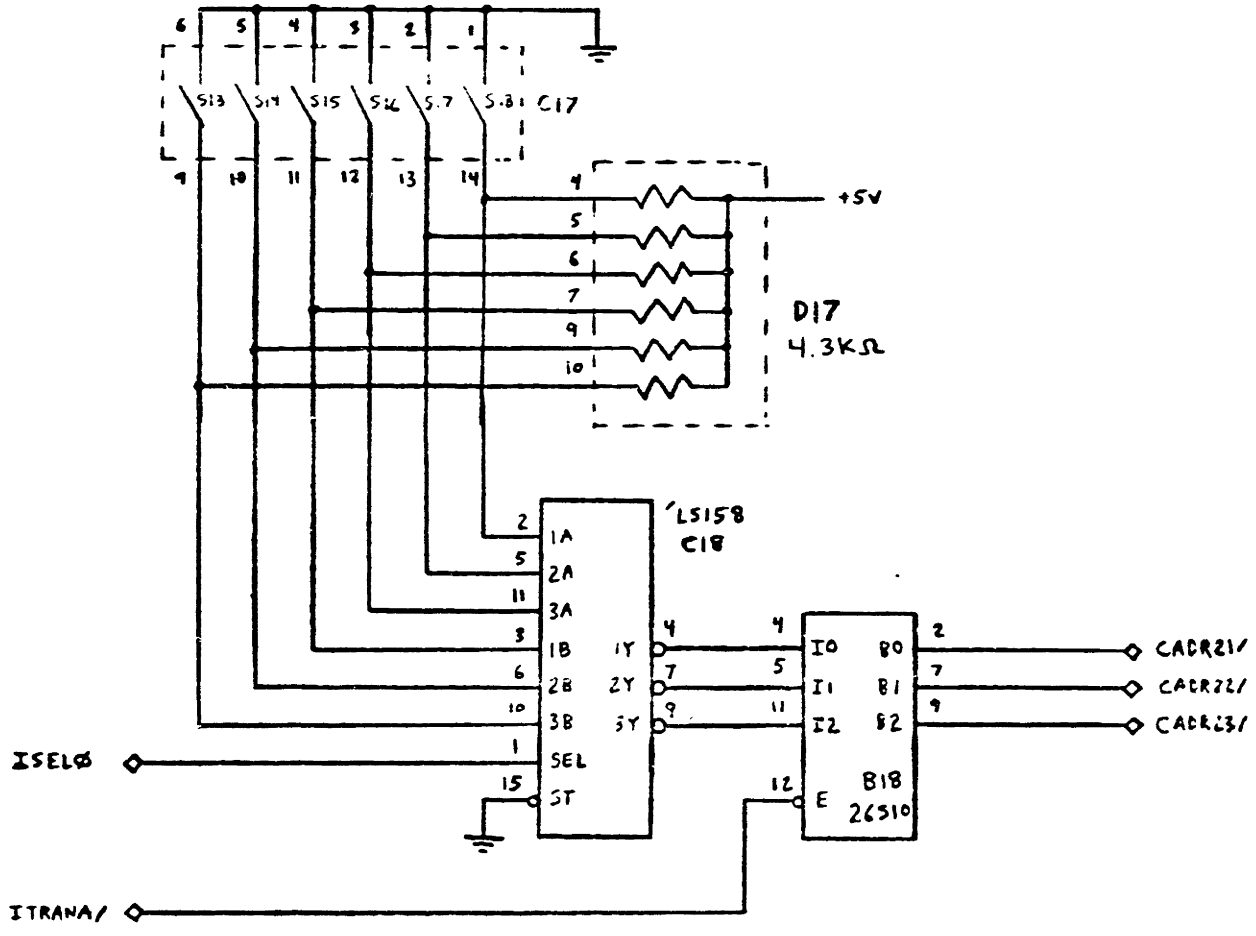
# D8


ISELØ
'LS74
INTØ
INTØ/
CLØ/
C6
PUPI
ICLKØ
'S74
'LS00
'LS08
C7
D7
D9
'LS74
INTI
INTI/
CLI/
C6
PUP
ICLKI
'LS00
'LS08
C7
D7
BCLK/
'LS27
E12
'S02
D8
'S175
C10
CBACK/
'S04
A4
CBGROUT/
'LS27
E12
ITRANA
ITRANA/
CLØ/
CLI/
'S08
U7
PUPI
INTØ
INTI
'S02
D8
'S74
D9
'S175
C10
ITRANB
ITRANB/

# D9

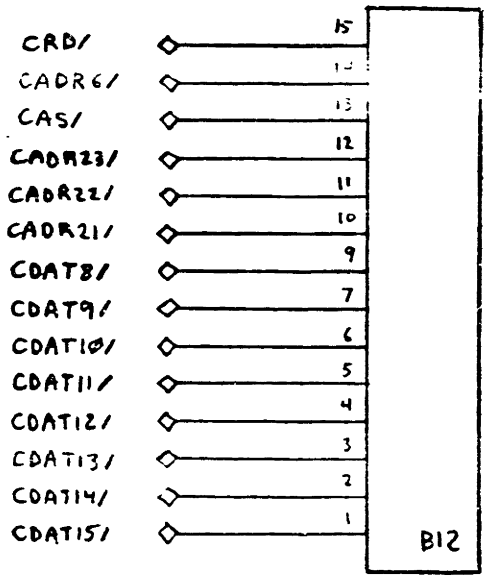

ISELØ
ITRANA
CADBS/
ITRANA/
'S00
A
E11
B
E11
C
E11
D
E11
SELØ/
SELI/
PUP2
CUDS/
CLDS/
E8
PR
D
Q
CL
$\overline{Q}$
'LS00
C
C7
'LS04
A
E9
B
E9
D
E9
C
E9
'LS20
A
E10
B
E10
WHEN/
WLEN/
WREN
VCC
BCLK/
BCLK
TTLDL-25
25nsec
20nsec
15nsec
10nsec
5nsec
IN
'LS14
B11
B9
'S37
A
B10
B
B10
CLK/

# D10

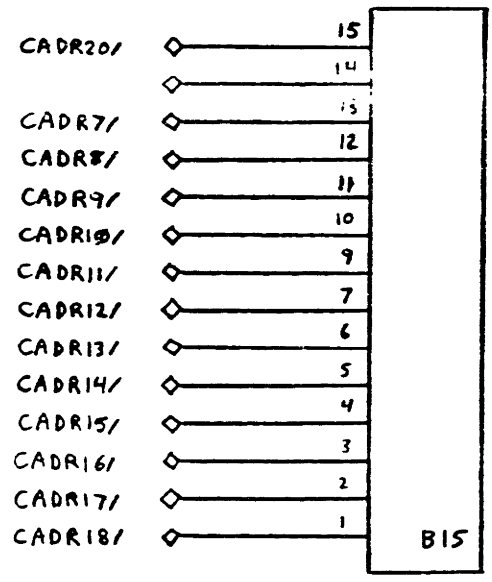

C17
+5V
D17
4.3KΩ
'LS158
C18
ISELØ
ITRANA/
CADR21/
CADR22/
CADR23/
B18
26S10

# D11

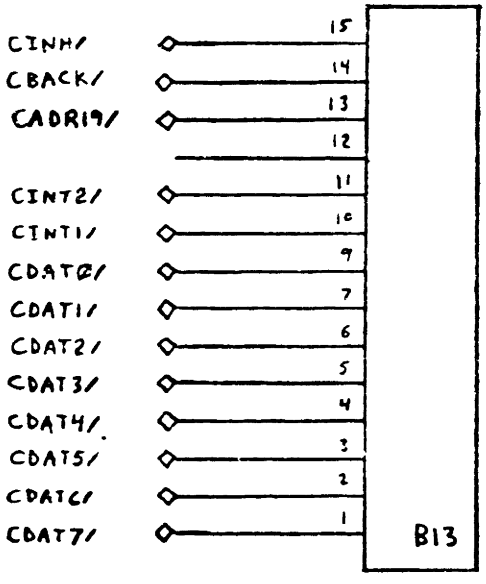
316E161261



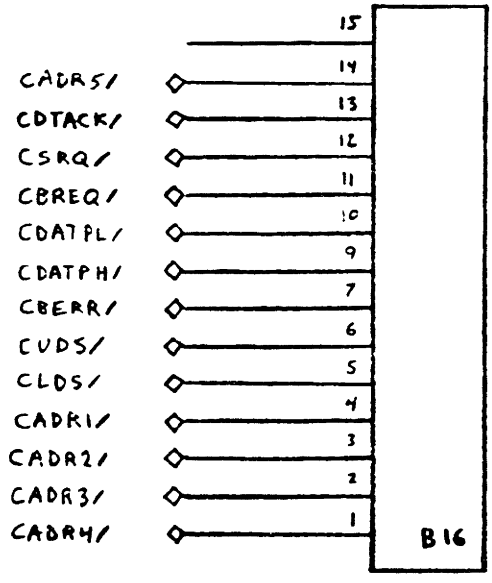

316E161261

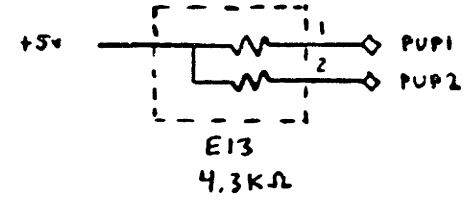



316E161261

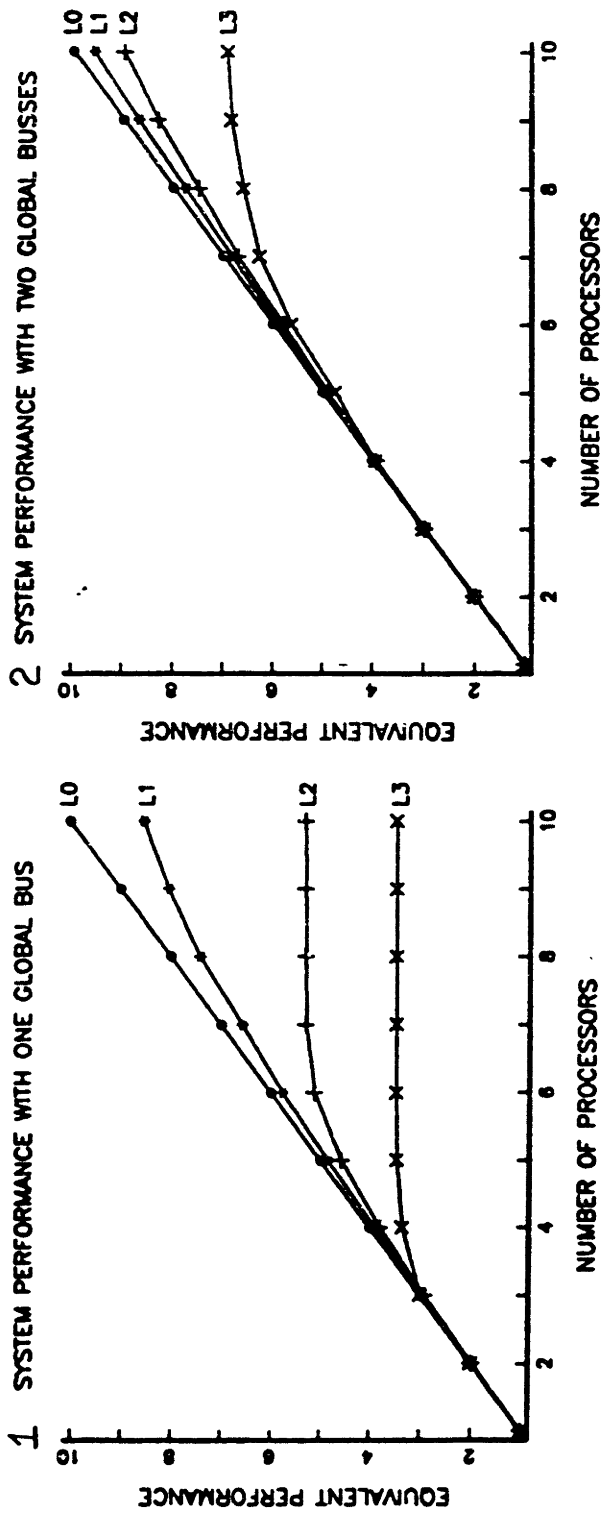



316E161261

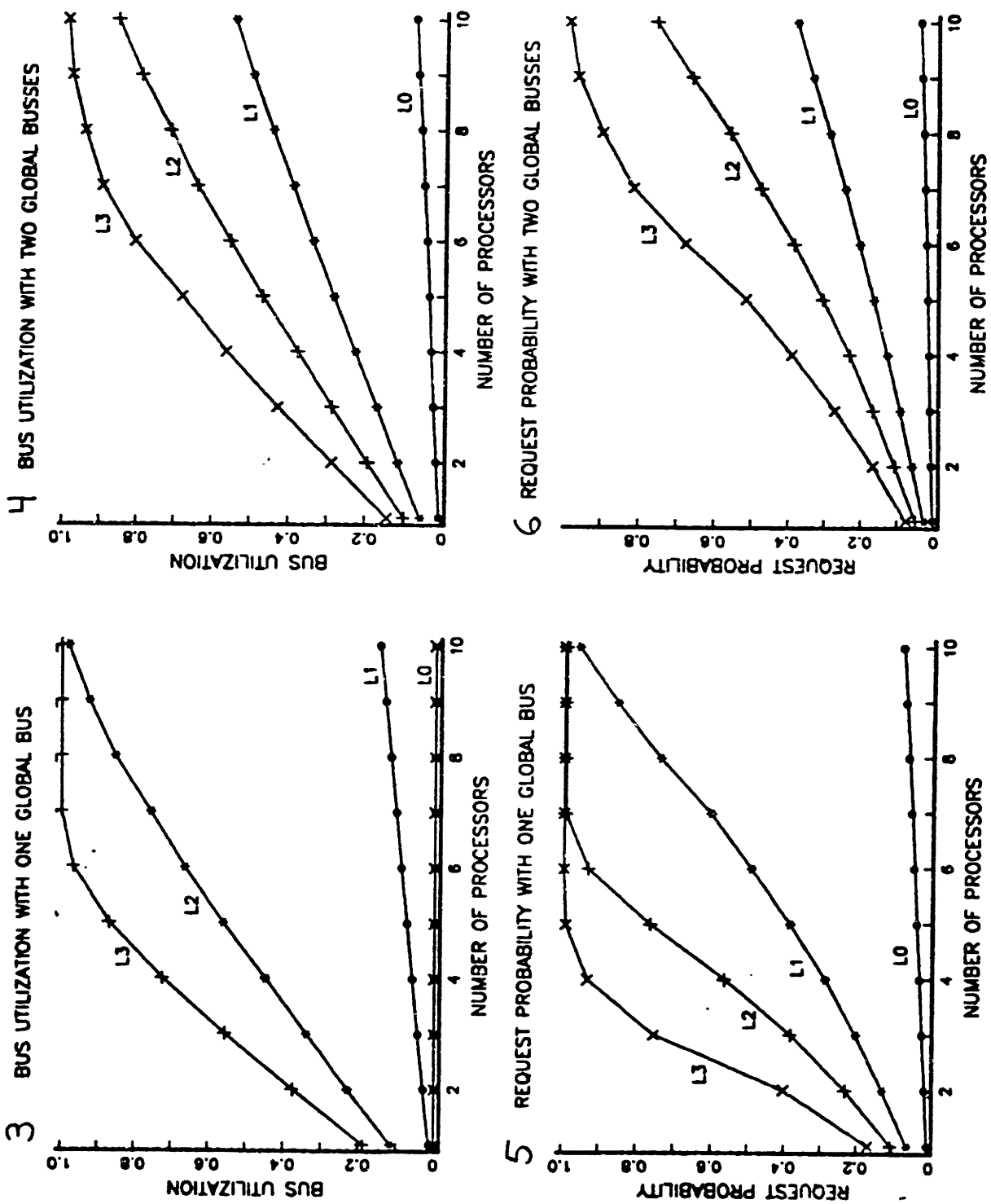


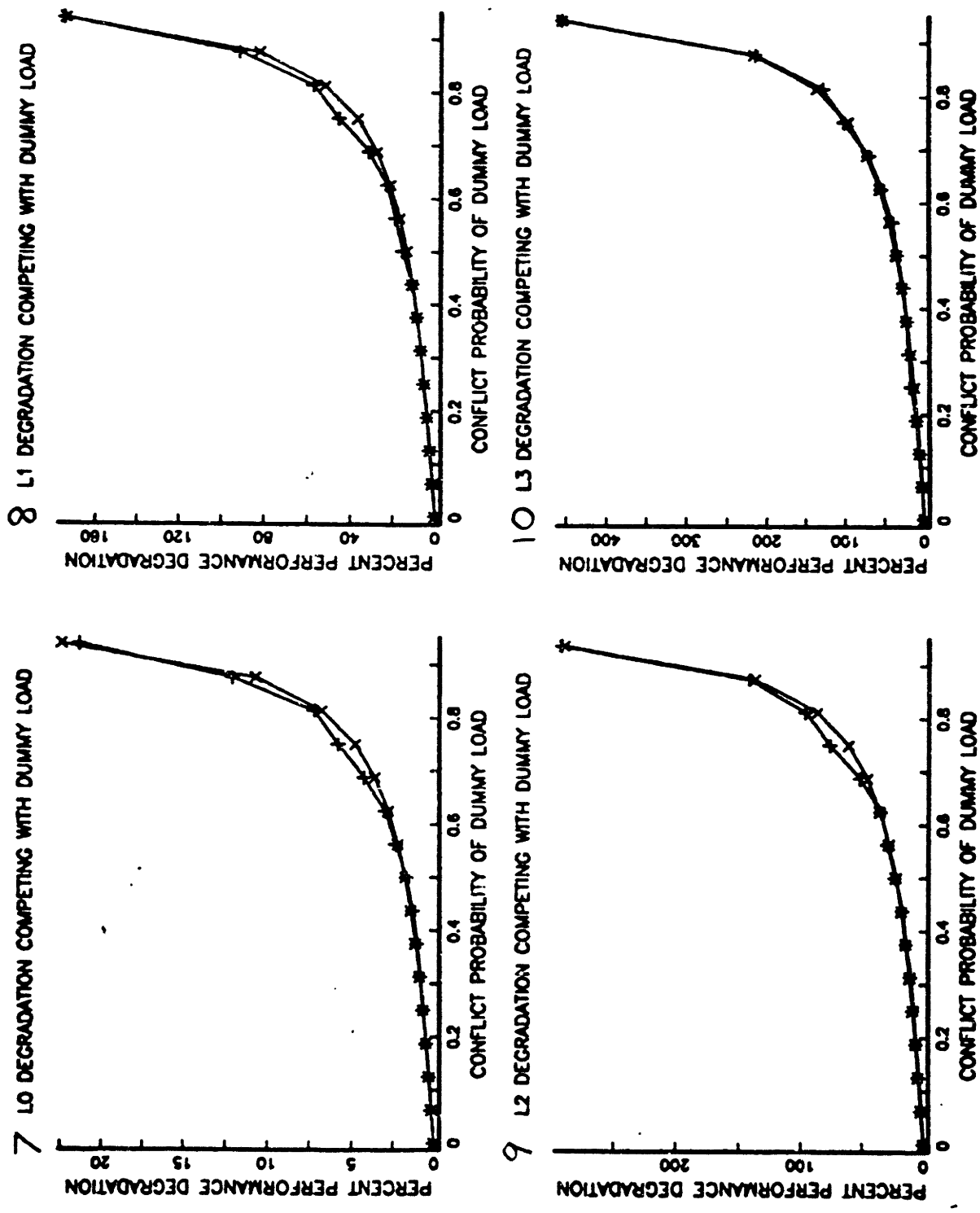
E1


1 SYSTEM PERFORMANCE WITH ONE GLOBAL BUS
2 SYSTEM PERFORMANCE WITH TWO GLOBAL BUSSES
L0
L1
L2
L3
EQUIVALENT PERFORMANCE
NUMBER OF PROCESSORS
2
4
6
8
10

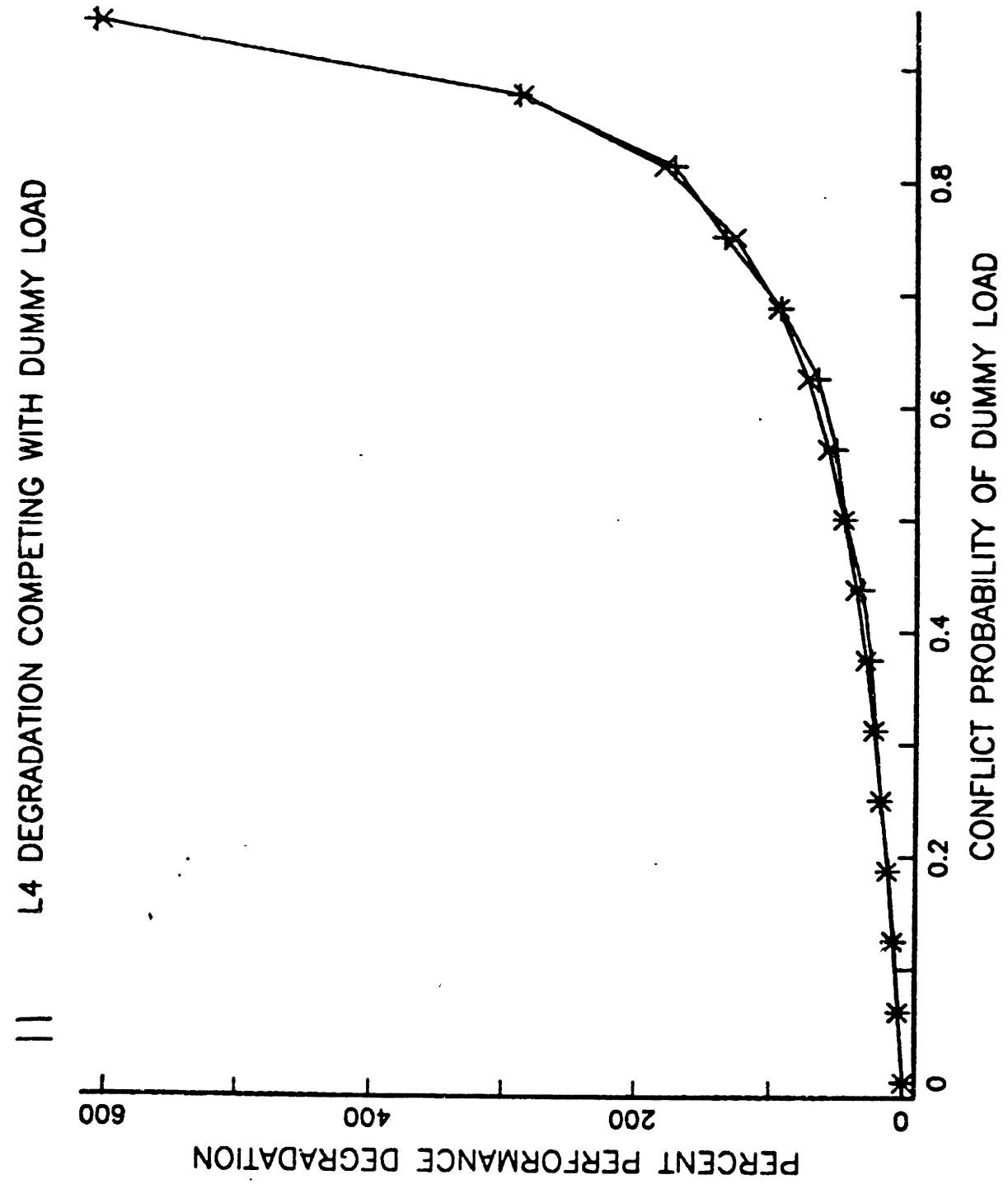
E2


3 BUS UTILIZATION WITH ONE GLOBAL BUS
BUS UTILIZATION
NUMBER OF PROCESSORS
L3
L2
L1
L0
4 BUS UTILIZATION WITH TWO GLOBAL BUSSES
BUS UTILIZATION
NUMBER OF PROCESSORS
L3
L2
L1
L0
5 REQUEST PROBABILITY WITH ONE GLOBAL BUS
REQUEST PROBABILITY
NUMBER OF PROCESSORS
L3
L2
L1
L0
6 REQUEST PROBABILITY WITH TWO GLOBAL BUSSES
REQUEST PROBABILITY
NUMBER OF PROCESSORS
L3
L2
L1
L0

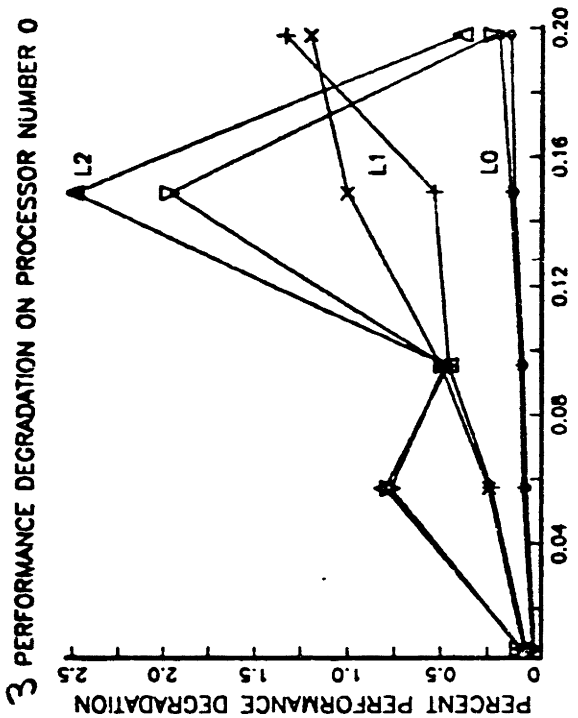
E 3

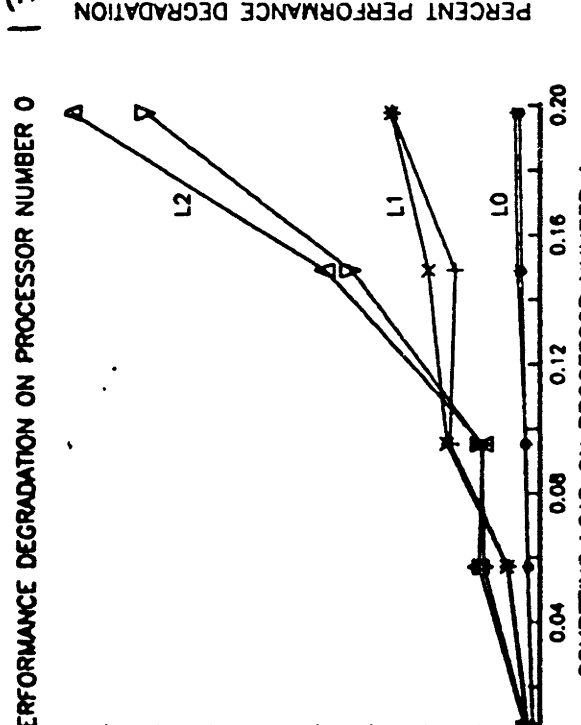

7 L0 DEGRADATION COMPETING WITH DUMMY LOAD
PERCENT PERFORMANCE DEGRADATION
0
5
10
15
20
CONFLICT PROBABILITY OF DUMMY LOAD
0
0.2
0.4
0.6
0.8
8 L1 DEGRADATION COMPETING WITH DUMMY LOAD
PERCENT PERFORMANCE DEGRADATION
0
40
80
120
160
CONFLICT PROBABILITY OF DUMMY LOAD
0
0.2
0.4
0.6
0.8
9 L2 DEGRADATION COMPETING WITH DUMMY LOAD
PERCENT PERFORMANCE DEGRADATION
0
100
200
CONFLICT PROBABILITY OF DUMMY LOAD
0
0.2
0.4
0.6
0.8
10 L3 DEGRADATION COMPETING WITH DUMMY LOAD
PERCENT PERFORMANCE DEGRADATION
0
100
200
300
400
CONFLICT PROBABILITY OF DUMMY LOAD
0
0.2
0.4
0.6
0.8

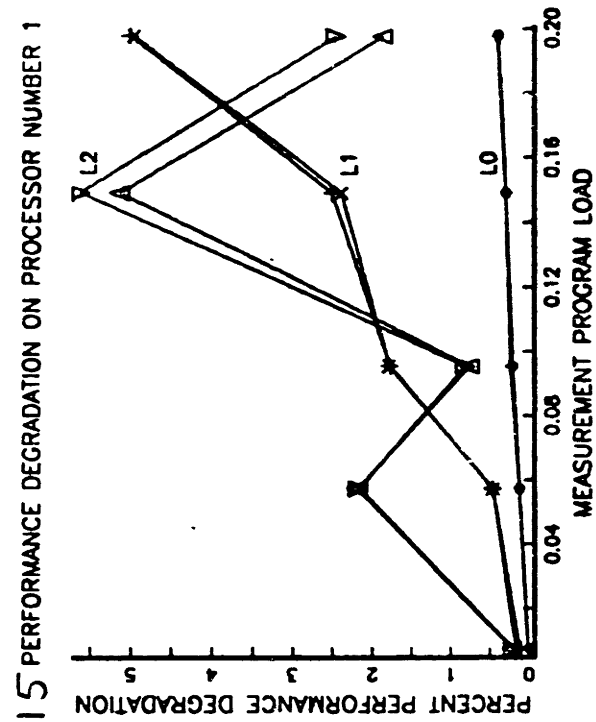
E4

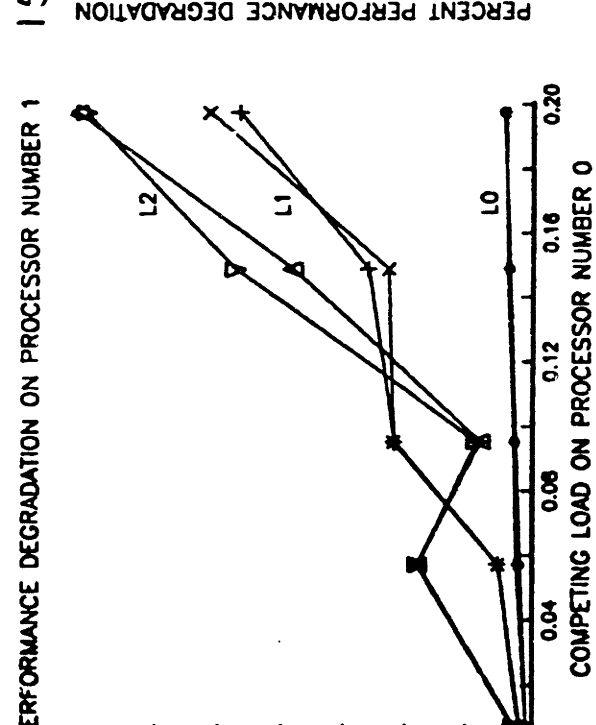

L4 DEGRADATION COMPETING WITH DUMMY LOAD
PERCENT PERFORMANCE DEGRADATION
0
200
400
600
CONFLICT PROBABILITY OF DUMMY LOAD
0
0.2
0.4
0.6
0.8

E5

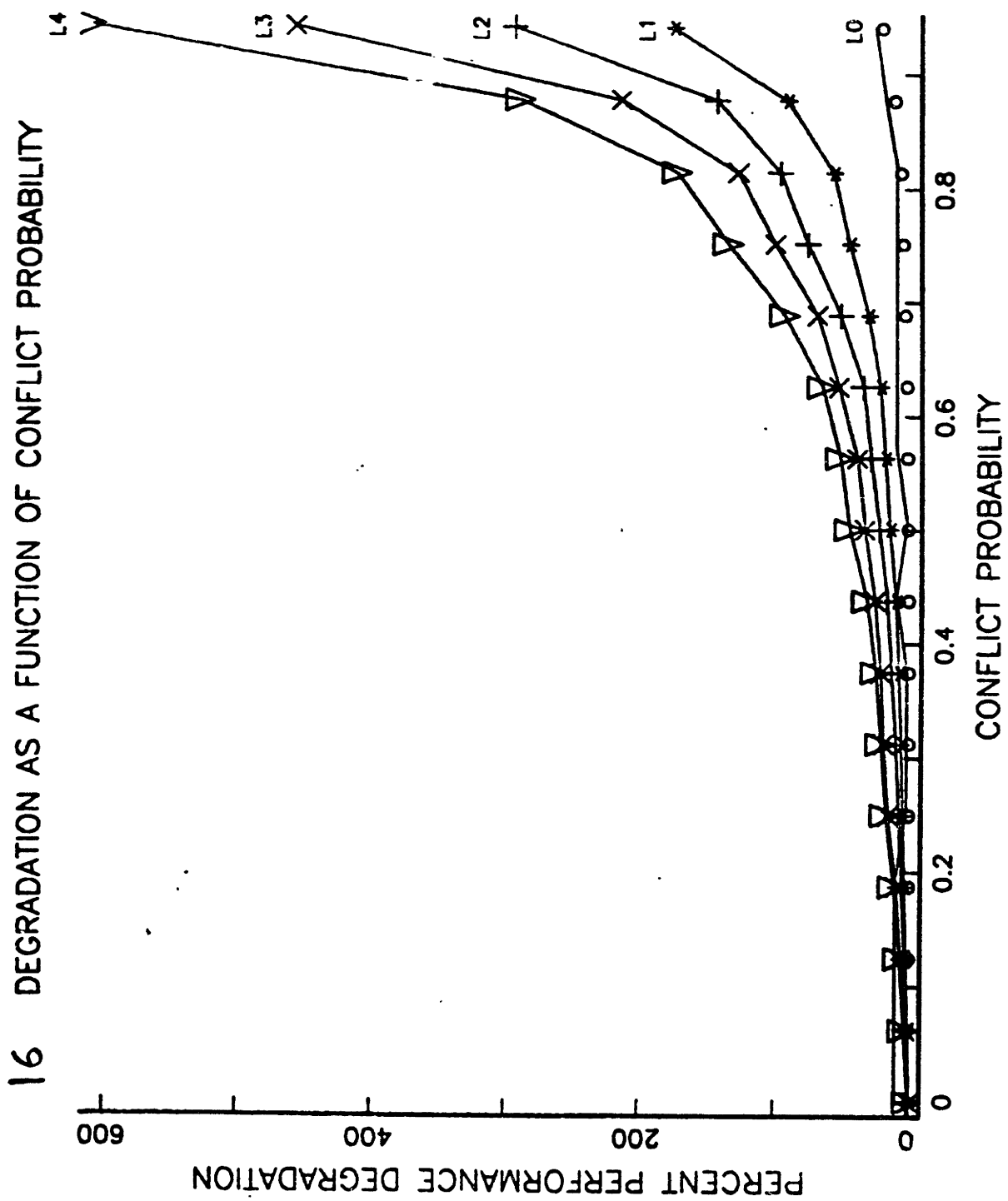

12 PERFORMANCE DEGRADATION ON PROCESSOR NUMBER 0
PERCENT PERFORMANCE DEGRADATION
COMPETING LOAD ON PROCESSOR NUMBER 1
L2
L1
L0


13 PERFORMANCE DEGRADATION ON PROCESSOR NUMBER 0
PERCENT PERFORMANCE DEGRADATION
MEASUREMENT PROGRAM LOAD
L2
L1
L0


14 PERFORMANCE DEGRADATION ON PROCESSOR NUMBER 1
PERCENT PERFORMANCE DEGRADATION
COMPETING LOAD ON PROCESSOR NUMBER 0
L2
L1
L0

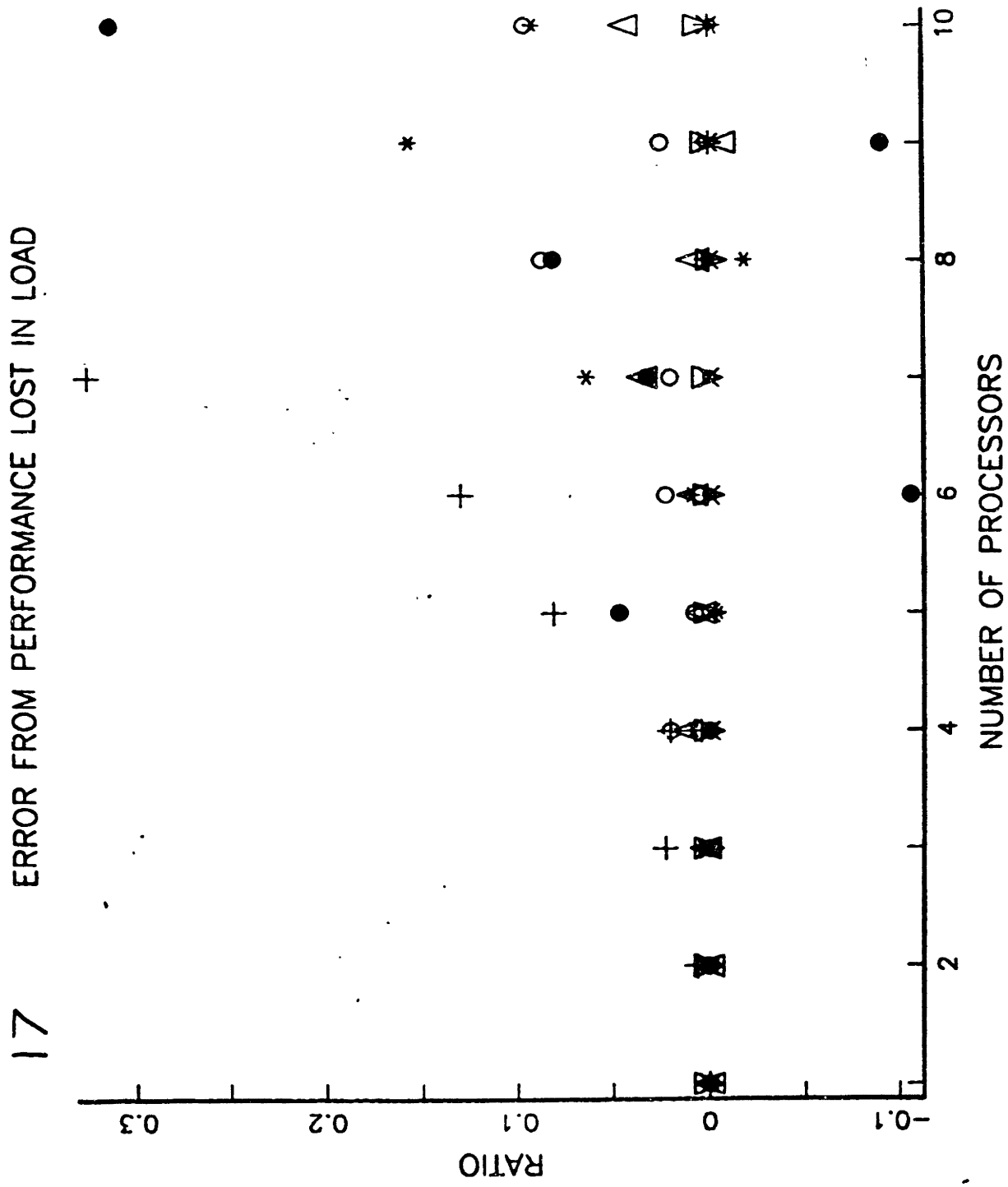

15 PERFORMANCE DEGRADATION ON PROCESSOR NUMBER 1
PERCENT PERFORMANCE DEGRADATION
MEASUREMENT PROGRAM LOAD
L2
L1
L0

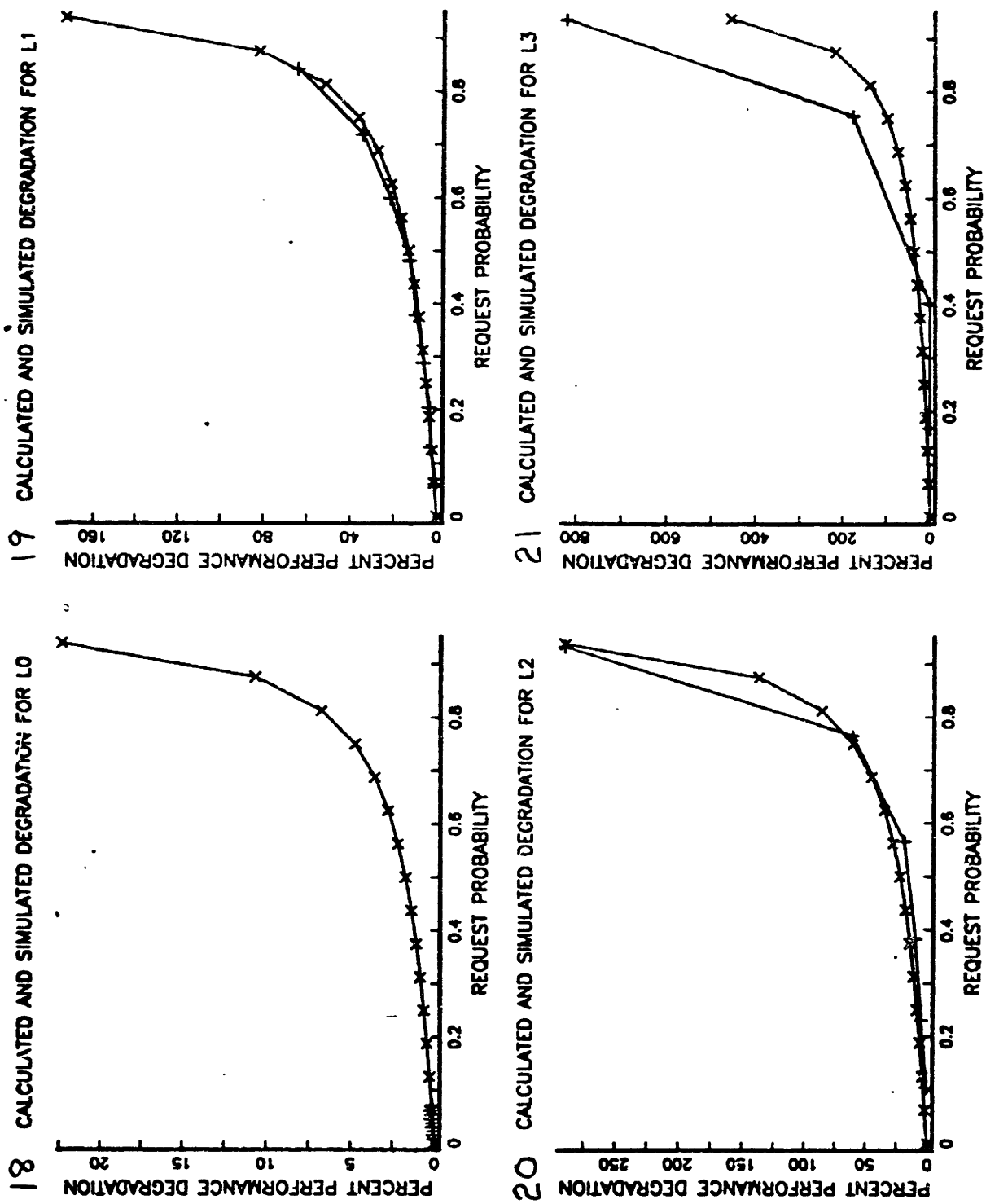
E6

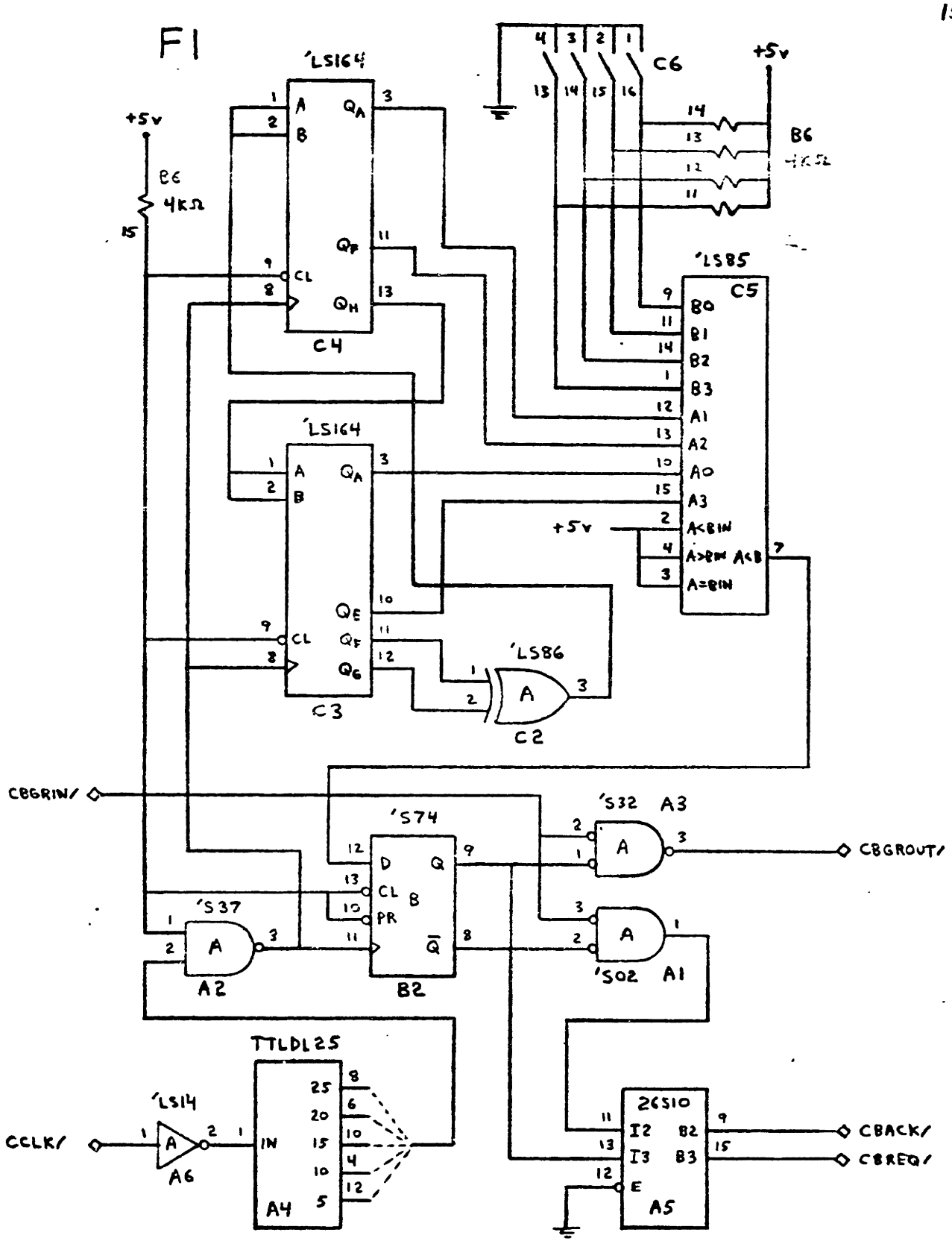



16 DEGRADATION AS A FUNCTION OF CONFLICT PROBABILITY

E7


17
ERROR FROM PERFORMANCE LOST IN LOAD
RATIO
-0.1
0
0.1
0.2
0.3
2
4
6
8
10
NUMBER OF PROCESSORS


18 CALCULATED AND SIMULATED DEGRADATION FOR L0
PERCENT PERFORMANCE DEGRADATION
0
5
10
15
20
REQUEST PROBABILITY
0
0.2
0.4
0.6
0.8
19 CALCULATED AND SIMULATED DEGRADATION FOR L1
PERCENT PERFORMANCE DEGRADATION
0
40
80
120
160
REQUEST PROBABILITY
0
0.2
0.4
0.6
0.8
20 CALCULATED AND SIMULATED DEGRADATION FOR L2
PERCENT PERFORMANCE DEGRADATION
0
50
100
150
200
250
REQUEST PROBABILITY
0
0.2
0.4
0.6
0.8
21 CALCULATED AND SIMULATED DEGRADATION FOR L3
PERCENT PERFORMANCE DEGRADATION
0
200
400
600
800
REQUEST PROBABILITY
0
0.2
0.4
0.6
0.8


F1
'LS164
C4
+5v
B6
4KΩ
C6
B5
4KΩ
'LS85
C5
B0
B1
B2
B3
A1
A2
A0
A3
A<BIN
A>BIN
A=BIN
A<B
'LS164
C3
'LS86
C2
CBGRIN/
'S74
B2
'S37
A2
'S32
A3
CBGROUT/
'S02
A1
TTLDL25
A4
'LS14
A6
CCLK/
2CS10
A5
CBACK/
CBREQ/

# APPENDIX F2

## Dummy Load Circuit Description

The Dummy Load circuit pictured on the previous page is placed on a global bus to simulate various levels of bus traffic. The operation of the circuit will be described here briefly.

Gate A of chip A6 receives the system clock from the global bus and chip A4 synchronizes it with all other system modules. Gate A of chip A2 then distributes the clock throughout the dummy load board.

Bus driver A5 asserts the global bus signals used in bus arbitration. An internal bus request is latched by flip-flop B2, which generates a bus request signal sent to the driver. The A gates in chips A1 and A3 control the global bus daisy-chain, asserting control when a request is granted.

Shift registers C3 and C4 are used, along with gate A of chip C2, as a fifteen bit pseudo-random number generator. Four arbitrary bits in the register are used to create a four bit random number sent to the A input of comparator C5. The pattern of four bit random numbers does not repeat often enough to synchronize significantly with the test programs. The B input to the comparator is generated by switches so that the probability of the comparator output being asserted during a given clock cycle can be varied from zero to 15/16 in increments of 1/16. This comparator output is used to trigger independent random bus requests through the flip-flop in chip B2.

## References


1. Childs, R.E., "Multiple Microprocessor Systems: Goals, Limitations, and Alternatives," Exploding Technology, Responsible Growth- Digest of Papers- COMPCON Spring 1979, San Francisco, Calif., Mar. 1979, pp.94-97.

2. Wulf, W. A. and C. G. Bell, "C.mmp- a Multi-mini- processor," AFIPS Conf. Proc., Vol. 41, 1972 FJCC, pp.765-777.

3. Hearst, F.E. et al., "A New Minicomputer/Multiprocessor for the ARPA Network," AFIPS Conf. Proc., Vol. 42, 1973, pp.529-537.
4. Jensen, Douglas E., "A Distributed Function Computer for Real-Time Control, "Conference Proc. 2nd Ann. Symp. Computer Architecture, 1975, pp.176-182.

5. Baum, A. and D. Senzig, "Hardware Considerations in a Microcomputer Multiprocessing System," Computer Technology to Reach the People- Digest of Papers- COMPCON Spring 75, San Francisco, Calif., Feb. 1975, pp.27-30.

6. Widdoes, L.C., "The Minerva Multi-Microprocessor," Conference Proc. 3rd Ann. Symp. Computer Architecture, Clearwater, Fla., Jan. 1976, pp.34-39.

7. Swan, R. J. et al., "Cm*- A Modular, Multi-microprocessor," AFIPS Conf. Proc., 1977, pp.637-655.

8. Pollard, L.H., "Multiprocessing with the TI9900," Conference Record- 11th Asilomar Conf. circuits, Systems and Computers, Pacific Grove, Calif., Nov. 1977, pp.461-465.

9. Maekawa, Mamoru et al., "Experimental Polyprocessor System (EPOS)- Architecture," Conference Proc. 6th Ann. Symp. Computer Architecture, 1979, pp.188-195.

10. Davidson, Edward S., "A Multiple Stream Microprocessor Prototype System: AMP-1," Conference Proc. 7th Ann. Symp. Computer Architecture, 1980, pp.9-16.

11. Tobias, Jeffrey M., "A Single User Multiprocessor Incorporating Processor Manipulation Facilities," Conference Proc. 7th Ann. Symp. Computer Architecture, 1980, pp.131-138.

12. "Introduction to the iAPX432 Architecture," Intel Corp., 1981.

13. "MC68000 Microprocessor Users Manual," Motorola Corp., 1980.

14. Bain, W. L. et al., "Performance Analysis of High-Speed Digital Buses for Multiprocessing Systems," Conference Proc. 8th Ann. Symp. Computer Architecture, 1981, pp.107-133.

15. Hoogendoorn, C. H., "Reduction of Memory Interference in Multiprocessor Systems," Conference Proc. 4th Ann. Symp. Computer Architecture, 1977, pp.179-183.

16. Bhandarkar, D.P., "Analysis of Memory Interference in Multiprocessors," IEEE Trans. on Computers, C-24, 9, Sept., 1975, pp.897-908.

17. Sastry, K. V. and Kain, R. Y., "On the Performance of Certain Multiprocessor Computer Organizations," IEEE Trans. on Computers, C-24, 11, November 1975, pp.1066-1074.