



Additional copies of this manual or other Intel literature may be obtained from:

Literature Department  
Intel Corporation  
3065 Bowers Avenue  
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP  
CREDIT  
i  
ICE  
iCS  
im  
Insite  
Intel

intel  
Intelevision  
intelec  
iRMX  
iSBC  
iSBX  
Library Manager  
MCS

Megachassis  
Micromap  
Multibus  
Multimodule  
PROMPT  
Promware  
RMX/80  
System 2000  
UPI  
 $\mu$ Scope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or RMX and a numerical suffix.

## LIST OF ILLUSTRATIONS

### Chapter 1. Introduction to the iAPX Component User's Guide

#### Figure

- 1-1 An iAPX 432 System
- 1-2 432 I/O Model

### Chapter 2. Introduction to iAPX GDP Architecture

#### Figure

- 2-1 Operations Defined on a Data Type
- 2-2 Some operations Defined on Character and Integer
- 2-3 Operations Defined
- 2-4 Different Access Paths and Rights to the Same Object
- 2-5 Network of Objects as an Object
- 2-6 Access Segment Symbology

### Chapter 3. Introduction to the iAPX IP Architecture

#### Figure

- 3-1 Main System and Peripheral Subsystem
- 3-2 Basic I/O Service Cycle
- 3-3 Peripheral Subsystem Interface
- 3-4 Peripheral Subsystem Hardware Interface
- 3-5 Interface Processor Window

### Chapter 4. iAPX Processor Environment Definition

#### Figure

- 4-1 Basic iAPX Data Lengths
- 4-2 Hardware Error Detection
- 4-3 State Diagram for Processor Packetbus
- 4-4 Nominal Write Cycle Timing
- 4-5 Stretched Write Cycle Timing
- 4-6 Minimum Write Cycle Timing
- 4-7 Minimum Read Cycle (Buffered System)
- 4-8 Minimum Read Cycle (Not Buffered)
- 4-9 Minimum Faulted Access Cycle

## Chapter 5. An iAPX 432 Multiprocessor System Implementation

### Figure

- 5-1 Block Diagram (Two Processor 432 System)
- 5-2 Physical Partitioning (Two Processor 432 system)
- 5-3 Address/Specification Demultiplexing
- 5-4 Local Access Operation
- 5-5 Cancelled Access Detection
- 5-6 Cancelled Access (With Subsequent Access)
- 5-7 Static Memory System
- 5-8 Byte Swapping for 32-bit Operand
- 5-9 Parallel Memory Banks
- 5-10 Interface to IP
- 5-11 GDP Schematics
- 5-12 IP Schematics

## APPENDIX

### iAPX 43201/43202 GDP Data Sheets

#### Figure

- 1. 43201 Pin Assignment
- 2. 43202 Pin Assignment
- 3. 43201 Block Diagram
- 4a. 43202 Block Diagram
- 4b. GDP Logic Symbol
- 5. Hardware Error Detection
- 6. QUIP Layout
- 7. Modes of Selector Generation
- 8. Modes of Displacement Generation.
- 9. GDP Packetbus State Diagram
- 10. 43201 Output Timing Specification
- 11. 43201 Input Timing Specification
- 12. Clock Input Specification
- 13. 43201 Hardware Error Checking Timing
- 14. 43201 Initialization Timing
- 15. 43201 Interrogation Timing
- 16. 43202 Output Timing Specification
- 17. 43202 Input Timing Specification

### iAPX 43203 IP Data Sheets

#### Figure

- 1. 43203 Pinout
- 2. 43203 Logic Symbol
- 3. 43203 Functional Block Diagram
- 4. 43203 Bus State Diagram
- 5. Timing Diagram for ACD Parameters
- 6. Timing Diagram for Local Processor Bus Timing
- 7. Multibus Interface Timing
- 8. Two Styles of ALE
- 9. XACK Interface Programmable Timing
- 10. Maximum Mode Interface
- 11. MULTIBUS System Interface

### QUIP Specification Figures

## LIST OF TABLES

### Chapter 4. 432 Processor Environment Definition

#### Table

- 4-1 ACD Specification Encoding
- 4-2 ICS Interpretation
- 4-3 PRQ Interpretation
- 4-4 BOUT Interpretation
- 4-5 iAPX 432 Component Signalling Scheme

### Chapter 5. An iAPX 432 Multiprocessor System Implementation

#### Table

- 5-1 Miscellaneous Pin Connection
- 5-2 IPC Register Bit Designation
- 5-3 Peripheral Subsystem Dedicated Ports

## APPENDIX

### iAPX 43201/43202 GDP Data Sheets

#### Table

- 1. GDP Operator Instruction Set
- 2. 43201 DC Characteristics
- 3. 43201 AC Characteristics
- 4. 43201 Capacitance Ratings
- 5. 43202 DC Characteristics
- 6. 43202 AC Characteristics
- 7. 43202 Capacitance Ratings
- 8. GDP Absolute Maximum Ratings

### iAPX 43203 IP Data Sheet

#### Table

- 1. 43203 Interface Pin Summary
- 2. Packetbus signals
- 3. IP Absolute Maximum Ratings
- 4. DC Characteristics
- 5. AC Characteristics
- 6. XACK Timing Parameters
- 7. 8-bit Processor Interface
- 8. 16-bit Processor Interface
- 9. Maximum Mode 8086 System

# iAPX 432 COMPONENT USER'S GUIDE

## CHAPTER 1

### INTRODUCTION

Welcome to Intel's new iAPX 432 family of components. The 432 product line was designed to solve the [existing; growing?] industrial problem of software development. Today's state-of-the-art technology has produced large-scale microprocessors capable of handling tremendous hardware tasks, but has failed to relieve the ever-growing needs of software design. Complex devices, conceptually designed to control this problem have in the end only added more weight to the burden.

#### WHAT IS THE PROBLEM?

Today's system applications have evolved into highly complex configurations. Retail business systems, telephone switching systems, laboratory and industrial control systems, communication systems, word processors, and many other large-scale systems have forced the industry to consider alternative

methods of handling the extensive requirements of multiple programs, multiple users, and multiple processors. The demands of existing high-level languages, operating system software, and the increased number of peripheral controllers and memory options have produced overwhelming problems for the system designer.

To further compound the problem, tomorrow's applications will increase system complexity by requiring fault-tolerant computing, transparent multiprocessing, distributed operation, networking, and multiple families of programming languages. These system requirements are beyond the scope of today's mini- and mainframe computers. This Component User's Guide presents a hardware discussion of the two components, the 43201 and 43202, that combine to form the General Data Processor (GDP), and the 43203 Interface Processor (IP). Also included in this book are instructions concerning the hardware implementation of the iAPX 432 system concepts. References to additional source documents are given as needed.

#### **WHAT IS THE SOLUTION?**

As you become familiar with the iAPX 432 family members, you will discover an innovative approach to the software development problem. It is not enough to say the 432 family processors have an enhanced architecture with the integra-

tion of over 100,000 silicon gates into 64-pin packages, nor is it enough to promise an increased throughput. What must be understood is that all phases of hardware and software design will be oriented toward safe and efficient control of information with favorable results in development costs.

The solution to complex and growing software demands is the development of the object-oriented machine. Such a machine can provide the important qualities of abstraction and decomposition. Abstraction is one of the inherent qualities of the iAPX432 family, which hides the irrelevant detail of predefined high-level languages, e.g., different number and data types. Decomposition is the quality of breaking up complex problems into manageable units and allows for modular programming. The iAPX 432 hardware provides an architectural structure that defines access limitations to prevent the damage of any established data base. The software designer need only be concerned with the construction of individual, easy-to-understand program modules that are more manageable and capable of working together as a system unit. Refer to Chapter 2 for a discussion of 432 Architecture and its importance to the hardware designer.

#### HOW IS IT DONE?

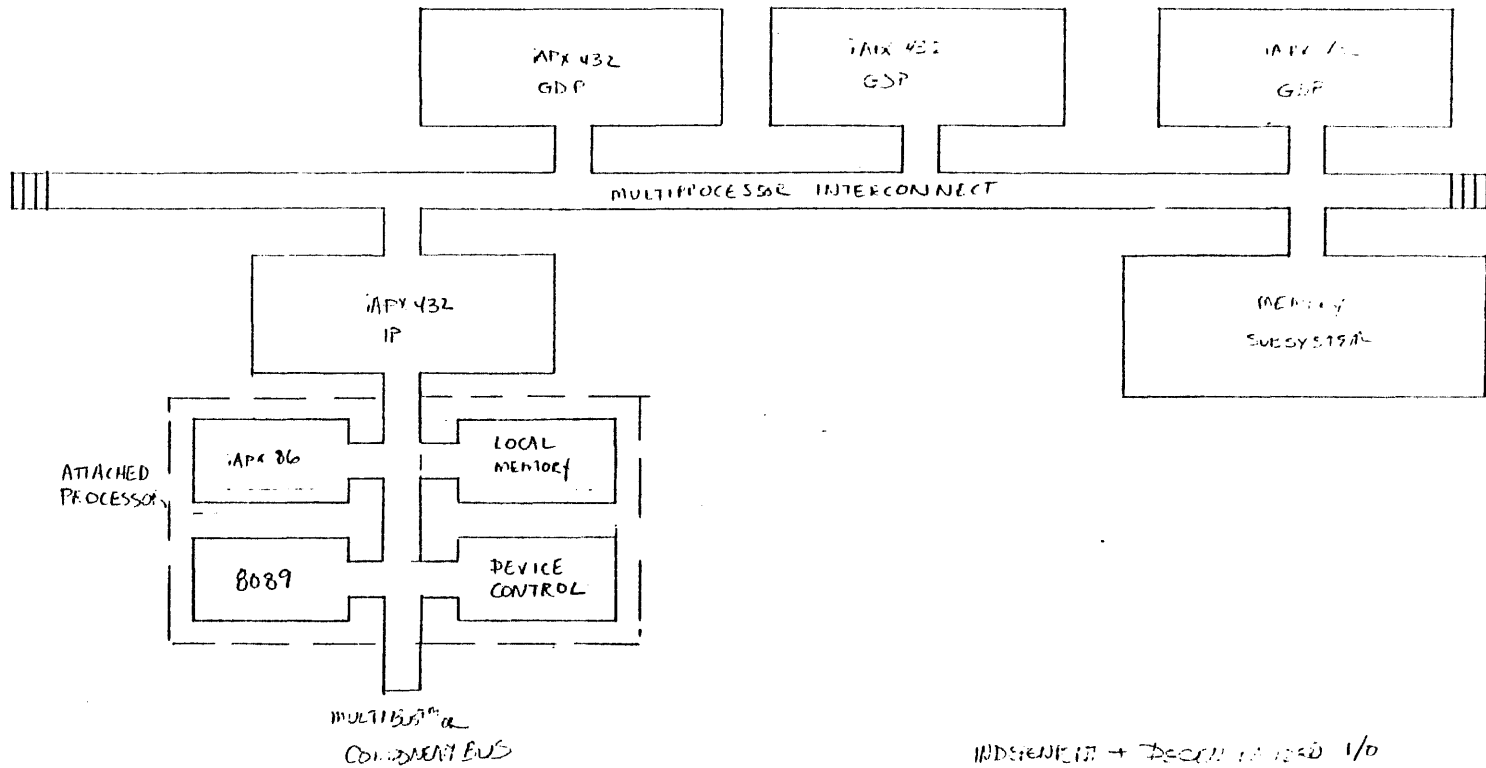
A basic iAPX 432 system consists of one or more Intel



general data processors (GDP's), one or more interface processors (IP's), several independent I/O systems, and a common memory system. Each external I/O system may consist of several I/O devices and microprocessors that may be capable of providing their own complex arbitration schemes, memory interleaving, and dynamic memory configuration. These external processors view memory as a single and continuous byte-addressable sequence, and are not aware of any system interconnection (although that capability does exist). Figure 1-1 illustrates a basic 432 system.

Figure 1-1 432 System Block Diagram

Programs executing in iAPX 432 processors view the memory and the interconnect address spaces quite differently from their attached processors. Programs are hardware-defined as processes, which are controlled by access descriptors; i.e., each process is limited to specific objects that may only be accessed if predefined conditions have been met. After verifying access rights, the hardware automatically computes physical addresses as they are required. The external processors define processes; the interface processor provides access to general data processors and memory resources to run those processes.



INDEPENDENT + LOCAL I/O  
Through attached processing

Figure 1-1 An iAPX 432 System.

Processor management methodology becomes more important as more subsystems are added. In the iAPX 432 system, processes are scheduled to be executed according to policies, implemented in software, that determine how processes are to share processors. As each process becomes available, i.e., reaches the head of its queue, it is dispatched to the next available processor that can execute the process. Conversely, an idle processor will wait at a dispatching port for a process to appear. Therefore, systems may become as complex as the mind can imagine. Software Engineers are now pleased to discover that the need to develop or redesign new software to handle the complexity of increasing system requirements does not exist! Hardware engineers are pleased to save valuable design effort in increasing the hardware capabilities by simply plugging additional GDP or IP units into their printed circuit boards. Chapters 5 and 6 provide a detailed discussion of the GDP and the IP and their respective roles in implementing these concepts.

Figure 1-2 represents an iAPX 432 system model complete with three GDP processors (43201 and 43202), three IP processors (43203), and three I/O subsystems. If the external processors were identical, it would be conceivable to use only one IP and several GDP's. The figure illustrates the ease with which information is allocated to available processors. Also shown is the independence of each subsystem as separate tasks are defined without the need for under-

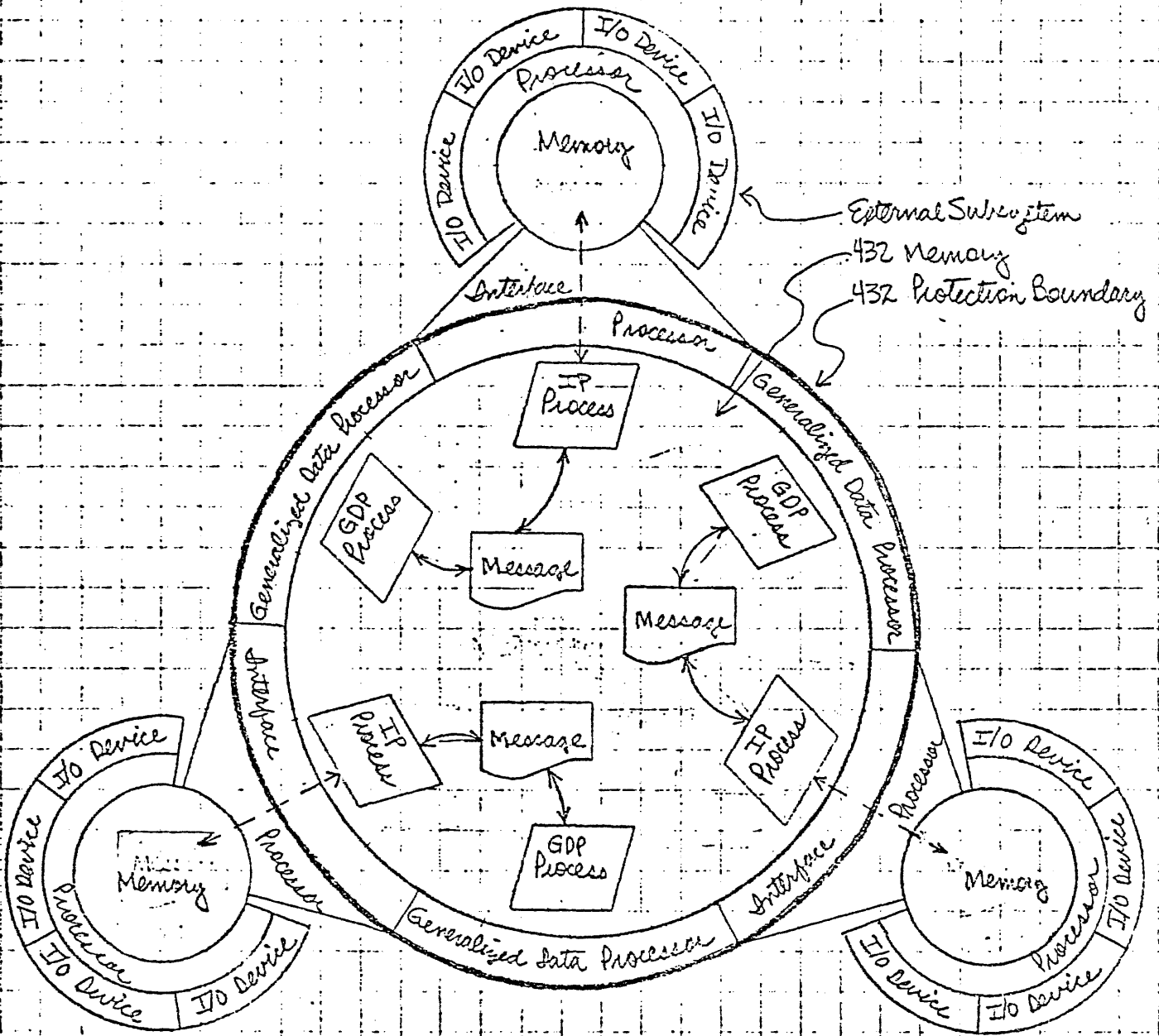


Figure 1-2 432 I/O Model

standing tasks for other subsystems. Refer to Chapter 4 for an actual 432 system configuration.

#### **OBJECT-ORIENTED MACHINE**

An object is a data structure that contains information in an organized manner. An object-oriented machine, such as the iAPX 432, is a device that handles data in terms of objects rather than the specific elements within the object. Certain primitive data types (Reals, Integers, Ordinals, etc.) each have a specific structure that may vary from one type to the next. A given binary number might represent a different meaning to each of these data types. In the iAPX 432 architecture, these data structures may be found in a contiguous set of memory locations or possibly a combination of several segments.

A 432 object not only contains organized information but also a basic set of operations defined to directly manipulate the data structure. The 432 hardware inherently ensures manipulation of the data structure by these specific operations only. In this manner, violations to the data structure are prevented. Also, each object can be referenced as one thing; i.e., there is no need to address any of the parts. Therefore, each object has a label to identify it from among the various types.

Notice that objects are manipulated not only by the hardware. Some objects are manipulated by a combination of hardware and software, and some only by software. Frequently used operations have been placed in the 432 structure (hardware implemented) while less frequent operations have been left to the software. Refer to Chapter 2 for a discussion of the following objects: Processor objects, Process objects, Context objects, Instruction objects, and data objects. The coordination of these objects represents the completed picture of the 432 system.

#### **A FINAL NOTE**

The Intel iAPX 432 family of components has been designed specifically to provide a solution to the growing software needs of the 80's. The 432 system should be viewed as a time-saving system, a cost-saving system, and a system to solve design development problems. Welcome to the Object-Oriented world of Intel's iAPX 432 family.

## CHAPTER 2

### INTRODUCTION TO iAPX 432 ARCHITECTURE

The purpose of this chapter is to familiarize the iAPX 432 hardware designer with the basic object-oriented concepts provided by the 432 system. A solid understanding of the following concepts is essential to the construction of 432 systems (Refer to the iAPX 432 General Data Processor Architectural manual, Order no. 171860, and the iAPX 432 Interface Processor Architectural manual, Order No.171862, for more detailed information on these concepts).

The iAPX 432 system is an object-oriented, capability-based architecture that supports software-transparent multiprocessing and adaptive virtual memory. By including operating-system and high-level language functions in hardware, it provides mainframe functionality and performance in a VLSI microcomputer form factor. I/O processing is fully independent and decentralized, allowing other iAPX processors (e.g., 86, 186) to perform I/O as attached processors. The IEEE floating point standard is fully supported in hardware.

The iAPX 432 combines VLSI technology with an architectural

and software design methodology to produce a new computer architecture that will significantly reduce the cost of large-scale software systems and enhance their reliability and security.

Several key concepts are fundamental to an understanding of the iAPX 432 architecture. They are:

**Objects** -- data structures having known types that can be either system- or user-defined; moreover, objects are acted upon by a restricted set of operations, which can be individual machine instructions, software package, or hidden hardware functions. In the 432, objects are represented by segments, subsets of segments called refinements, or arbitrarily complex networks of segments and/or refinements.

**Access descriptors** -- sometimes called "object references" or "capabilities," these are how objects refer to one another. Each object confers upon its holder access to an object, along with certain rights associated with the object, for instance, read/write or send/receive.

**Segments** -- segments form the basis for the physical representation of objects. Each segment is a contiguous block (that is, no gaps) of address space, up to 64K (65,536) bytes long. Each segment has a hardware-



recognized base type, which must be either data segment or access segment.

**Data Segments** -- data segments have no inherent structure, but are hardware-recognized. They are variously used to hold instructions and scalar data items, but not object references.

**Access Segments** -- each access segment is a hardware-recognized array of object references, protected by the hardware to ensure that inadvertent or malicious alteration of access rights or addressing information does not occur.

**System Management Objects** --in addition to a base type (data segment or access segment), each segment is assigned a system type so that objects can be built to represent processors, processes, and storage resources. The hardware-recognized typing of these objects, together with a restricted set of operations defined on each type, facilitates efficient system management as processor management, process management, and storage management.

**Program Environment Objects** -- by building "packages" or modules of arbitrarily complex objects, a protected program environment is established wherein access to

instructions, scalar data items, and objects can not only be controlled, but also hidden. Programming abstractions such as "subroutine activation records" and "static access environment" are realized in the hardware as contexts and domains, respectively.

**User-defined Objects** -- also called "extended-type" objects, these are objects whose structure and behavior are defined by the user. This extends the concept of hardware typing to include arbitrary objects (and networks of objects) that can only be acted on by a restricted set of operations, which may also be defined by the user.

**Type Management Objects** -- in order to manage the arbitrary user-defined types, the objects that are instances of those types, and the operations that act upon them, the 432 architecture provides a hardware-recognized type definition object that can be selectively made available to other users, even though its inner workings are hidden from them.

**Object Locking** -- objects can be locked by either hardware or software to ensure object integrity in a multiple processor environment.

## **OBJECTS**

## Operations Defined on Data Types

The concept of an object grows out of the natural concept of different data types having different operations that act upon them. Schematically, we depict the situation as shown in Figure 2.1.

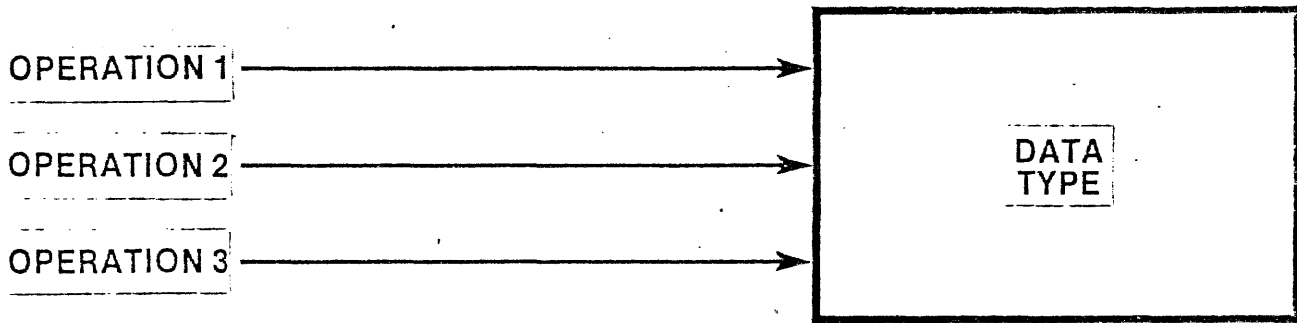
Figure 2-1. Operations Defined on a Data Type


### Operations Defined on CHARACTER and INTEGER types

Data types such as CHARACTER and INTEGER are familiar examples of data types. The data items 'A' and 65 are instances of these respective types, which are conceptually different, even though they have the same internal representation, assuming 'A' is represented in ASCII and 65 is represented as a pure binary number in a byte.

These data types may have different operations defined on them by:

- o Machine instructions (such as an editing instruction for characters, or an arithmetic instruction for integers),



 **FIGURE 1-1: OPERATIONS DEFINED ON A DATA TYPE**

2-1

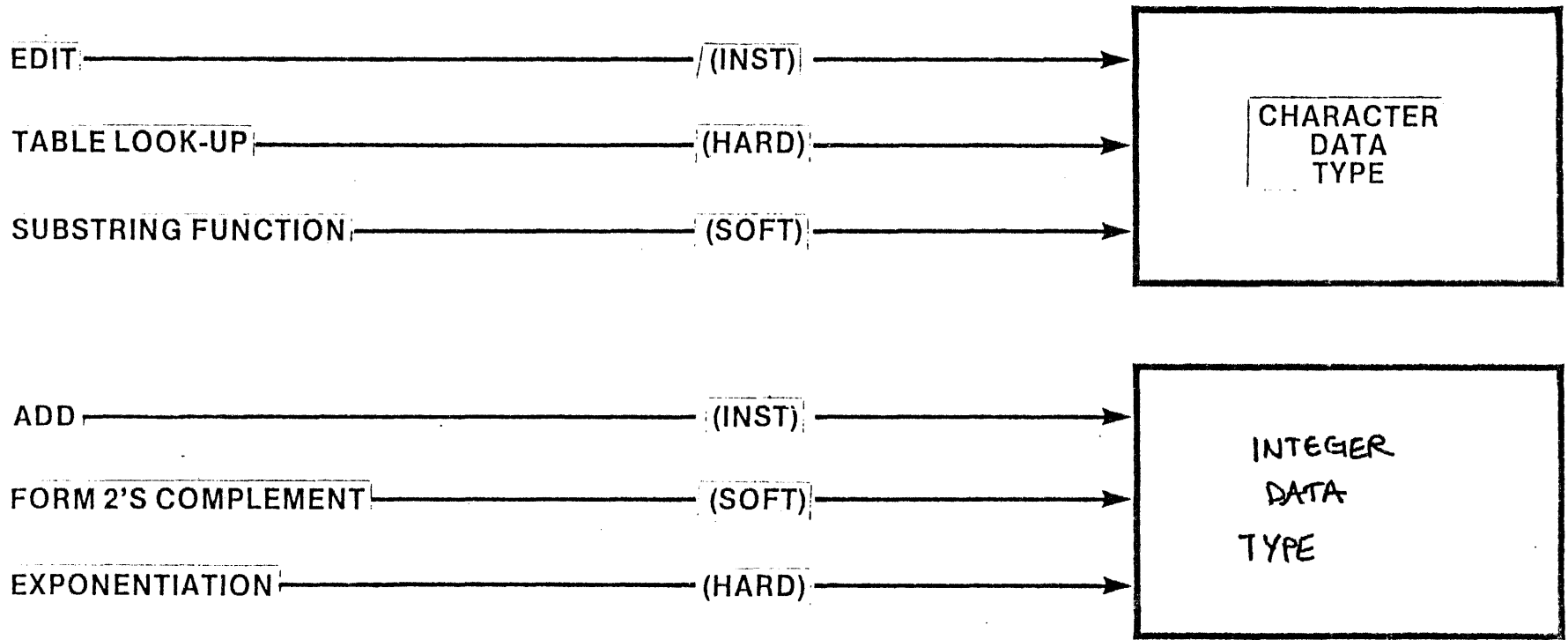
- o Hidden hardware functions (such as a table look-up for a character, or forming the 2's complement of an integer),
- o Software-implemented functions (such as character substring functions, or exponentiation of integers).

Schematically, we can depict this situation as shown in Figure 2-2.

Figure 2-2. Some Operations Defined on CHARACTER and INTEGER Types.

Characters and integers are not objects, but looking at them this way may help to understand the rationale of objects. Before CHARACTER and INTEGER data types were developed, they existed as abstractions. Programmers coding in octal or binary coded the character 'A' the same as if they were coding the integer 65, and the discipline of choosing the appropriate operations (instructions or subroutines) for the data types they had in mind were left to them.

Programming languages and their translators (assemblers, compilers, and interpreters) introduced formalisms of



13A

2-2  
 FIGURE 13A SOME OPERATIONS DEFINED ON CHARACTER AND INTEGER TYPES

subroutines, functions, and procedures into the "art" of programming code: by enforcing declarations of these formerly abstract concepts in programs, language translators could at least ensure that, for instance, a function call was being used as a function call. (An interesting counter-example of this occurred in FORTRAN, in which the statement:

$$A = F(X)$$

could be the assignment to A of either:

the value of the function F evaluated using X as an argument, or -

the Xth element of the array F (depending on the declaration of F).

### Software Data Types and Associated Operations

As code abstractions (procedures, functions, and subroutines) demonstrated their merit in enforcing discipline in programming, data abstractions became realized as arrays, vectors, records, and other formal structures. Each essentially different kind of data structure was realized as a data type, and sets of operations were introduced to act upon only certain data types.

For instance, the data type MATRIX was formalized in certain

languages, and the operations DETERMINANT and INVERT, usually implemented in software, could be performed only on a data structure of type MATRIX. If the type did not match the operation, the program would not compile properly.

Compile-time checking ensured that execution time need not be wasted on large programs that would not run properly because a data type did not match the operation applied to it.

### **Hardware-Recognized Data Structures and Associated Operations**

The software concepts of typed data structures and the operations that can be applied to them are realized in the machine architecture of the iAPX 432. In doing so, the 432 raises the level of the instruction interface from the traditional data computational types of bytes, double bytes, words, etc., to objects.

Just as languages defined operations on primitive data types, the architecture of the 432 (and any language capable of embracing this architecture) defines operations on objects, which are instances of typed data structures. Thus, a machine instruction on the 432 typically specifies as its operands objects that represent:



- o Processors in a multiprocessor environment
- o Processes (tasks) in a concurrent environment
- o Dispatching ports that bind processes to processors
- o Interprocess messages sent from one process to another, and synchronized using communication ports
- o Domains (also called type managers) as the static access environments of programs
- o Contexts (subprograms)

And many more. These objects are all system objects; they are instances of system types, that is, types built into the system.

### **Objects as high level, strongly typed operands**

Intuitively, a system object is a high level operand that raises the level of the instruction interface. "High-level" in this sense means that the operand, rather than simply being a byte or word of data, is an organized data structure in memory that is recognized by the hardware as being an instance of a type that can be manipulated only by a select class of operations.

Traditional computer systems implement the abstractions of access environment control, task management, process status, interprocess communication, etc. using a combination of software structures defined by operating systems, utilities, and compilers, and associated supervisor calls.

The 432 implements these concepts as machine-inherent data structures referenced by instructions as strongly typed operands that are validity-checked at run time:

- o Each has a type (base and system), and
- o Each can be used as an operand only as intended, i.e., only with a well defined subset of the operator (instruction) set.

Many 432 system objects are highly organized "control blocks" reminiscent of operating task control, message queueing, fault handling, and other mechanisms. These "control block" system object operands do not in and of themselves enforce operating system policy, but rather provide the basic mechanisms from which virtually any operating system can be designed and built. These mechanisms provide functionality needed to design and build clean, fast running software. Indeed, for large, high availability systems, the software development cycle can be shortened

using the 432.

### Extended types

The 432 architecture allows users to define arbitrary data structures as objects, and to define the restricted set of operations that are allowed to act on these objects. These user-defined object types are called extended types, since they extend the set of recognized object types and the operations allowed by each type.

### Example of a 432 object and its associated operations

The processor object represents a unit of processing power in the system and as such is an abstraction of the physical processing unit, reflecting its various states (running, queued, assigned, etc.). Processor objects in a given system are in one-to-one correspondence with physical processors in that system.

A processor object thus provides a means of assigning the processor it represents to a particular process set (work stream), and also provides a means of communicating with (sending messages to) a processor.

### Operations

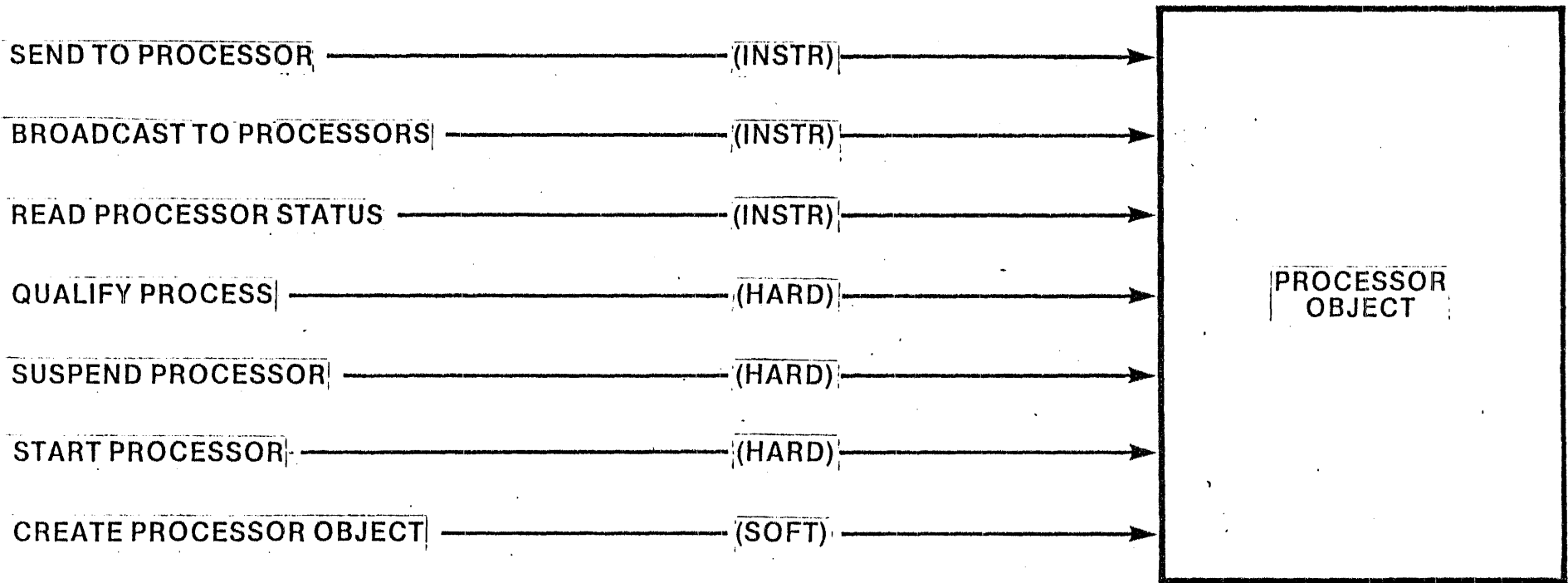
Figure 2-3 indicates a processor object and the operations that act upon it.

Figure 2-3 Operations defined on a processor object.

### Access descriptors-- "handles" for objects

When several users require access to the same object, at least two problems can arise:

- o **Need-to-Know Rights.** In a multi-user environment, not all users need to have the same rights to an object; for instance, one user may require only "read" rights to sensitive data, whereas another may require both "read" and "write" rights to that data structure (object); a mechanism is needed whereby different rights can be conferred on different instances of procedures. The solution to this problem is discussed below.
  
- o **Exclusive Access.** In a multiprocessor environment, one user of a data structure (object) may require exclusive access to that object for a particular operation, to ensure that no other processor alters the object while the operation proceeds. The solution



LEGEND: INSTR = OPERATION IS A MACHINE INSTRUCTION (OPERATOR)  
 HARD = OPERATION IS PERFORMED BY HARDWARE AS A RESULT OF CONDITIONS DETECTED BY THE PROCESSOR  
 SOFT = OPERATION IS IMPLEMENTED BY SOFTWARE (E.G. AN OPERATING SYSTEM)

FIGURE 1-3 OPERATIONS DEFINED ON A PROCESSOR OBJECT

to this problem is described at the end of this chapter under "Object Locking."

In the past, problems like the first have been solved by introducing "privileged" and "user" instructions or modes of operation, which have the disadvantage that each user is either "universally privileged" or "universally restricted." Systems that generalize this scheme to three or more levels typically encounter similar problems in their blanket or graduated privilege schemes. Such schemes characteristically do not take individual procedure access requirements into account.

In the object-oriented, capability-based environment of the 432, several privilege and protection abstractions are realized using access descriptors, which act as privilege and protection "handles" for system objects. The 432 approach is to grant a procedure read/write rights to explicit objects on an individual, need-to-know basis. By possessing an access descriptor to an object, a procedure is conferred the read/write rights specified by that access descriptor, and is thus capable of reading/writing the object. An access descriptor is thus sometimes called a capability.

A running procedure context cannot read from or write to an object unless that procedure has an access descriptor for

that object. Moreover, the procedure cannot read to or write from the object accessed by the access descriptor unless that read or write privilege is expressly granted, as indicated in the access descriptor. Furthermore, the procedure cannot delete the access descriptor unless that right is explicitly indicated within the access descriptor itself.

Thus, access descriptors provide:

- o Controlled access to a particular object; if a request to use an object matches the rights specified in the access descriptor for that object, the access descriptor maps a virtual address into a physical address, permitting access.
- o Read and/or write protection of the object by an instance of a procedure, independent of other objects, other procedures, and other instances of the same procedure.
- o Delete protection of the access descriptor itself.
- o Other object-specific information, such as system rights, an access descriptor validity check, and an indication of object storage allocation.

Since an object can be subsetted (using a refiner, as described later in this chapter) to obtain a smaller object, access descriptors provide a granularity of protection not found in "privilege-level" computer systems.

### **Access Paths**

Many access descriptors can exist for the same object. Each access descriptor belongs to some (but may be copied and/or passed to another) procedure (context), and defines the access rights of the procedure using it to access an object.

For instance, Figure 2-4 shows three different access descriptors to the same object. Context A can read from Object A, but cannot write into it; Context B can write into Object A, but cannot read from it; and Context C can both read from and write into Object A.

Figure 2-4. Different Access Paths and Rights to the Same Object.

### **Segments -- the Representations of Objects**

At the machine level, objects are represented by segments. A segment is characterized as being a contiguous block of



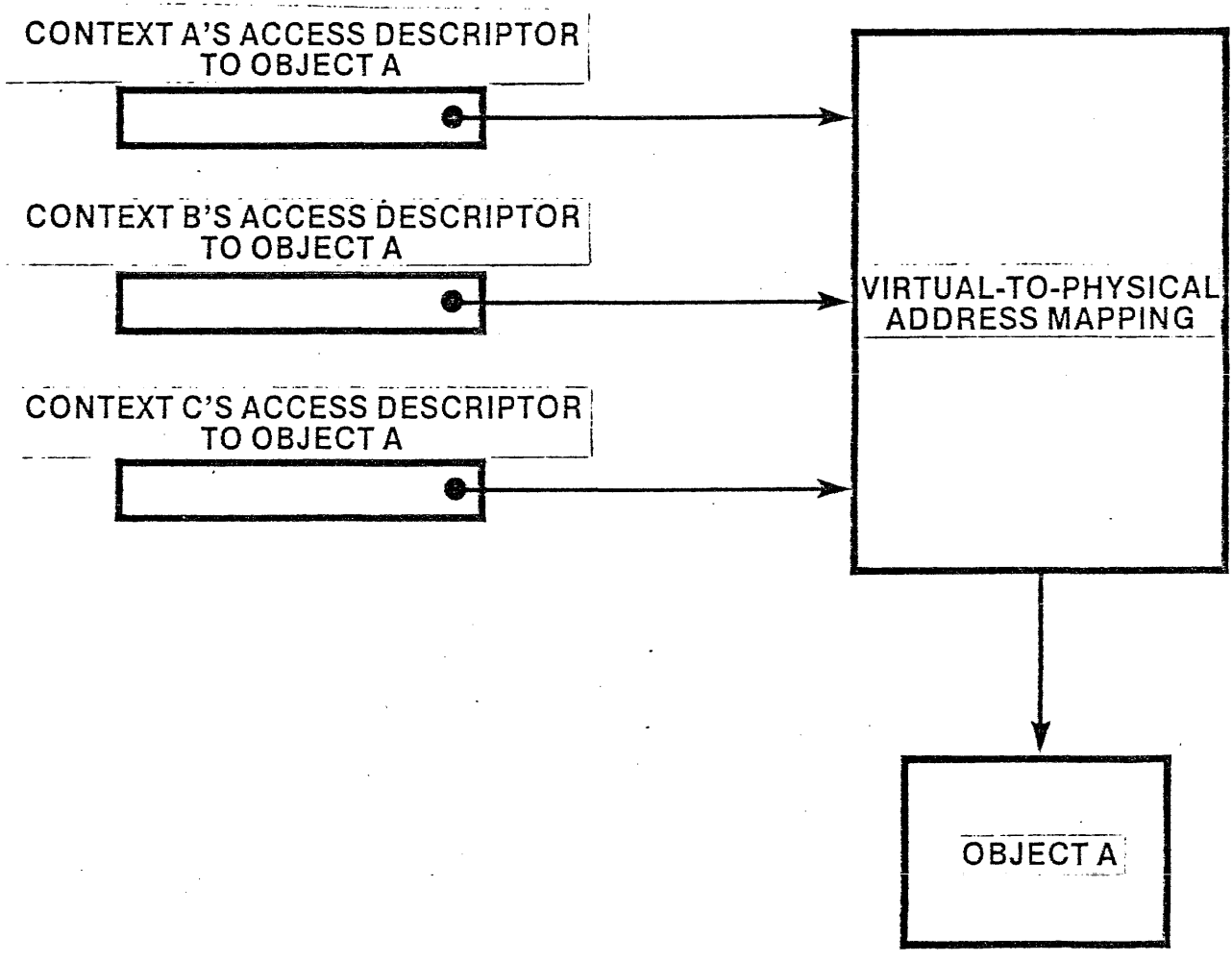


FIGURE 2-4 DIFFERENT ACCESS PATHS AND RIGHTS TO THE SAME OBJECT.

2-4

memory and having:

- o A base address.
- o A length, which can be up to 65536 bytes.
- o A base type and a system type, both of which depend on the intended use of the segment as an object.

### Relationship Between System Objects and Segments

Every segment is an object, but the converse does not hold.

Every object is either:

- o A segment, or
- o A subset of a segment (called a refinement), or
- o A collection of segments and/or refinements.

Both segments and system objects have these characteristics:

- o Each has a base address. If an object is a single segment, the base address is that of the segment. If an object is a group of segments, the base address is that of the "root" segment, i.e. the first in the access path.

- o Each has a base type, which must be either of type data segment or type access segment.
- o Each has a system type, which can be generic (data segment or access list), or can be specialized as a transformer, domain, dispatching port, context, operand stack, storage resource, etc.
- o Each is represented by, and accessed through, a segment descriptor in a segment table residing in memory. A segment descriptor encodes a segment's base address, length, base type, system type, processor class, and other information.

**Object Networks.** As an example of an object that is a collection of segments, consider the access segment (Object 0) together with the objects A1, A2, and A3 referenced by its access descriptors. This collection defines a multisegment object. Extending this concept, each of the objects A<sub>j</sub> (j = 1, 2, 3) could itself be an access list referencing objects B<sub>jk</sub> (k = 1, 2, 3). The resulting network of objects B<sub>11</sub>, B<sub>12</sub>, ..., B<sub>32</sub>, B<sub>33</sub>, could then be considered an object. Figure 2-5 depicts such an object.

Figure 2-5. A Network of Objects as an Object.

"ROOT" SEGMENT (ACCESS LIST):

MORE ACCESS LISTS:

DATA SEGMENTS:

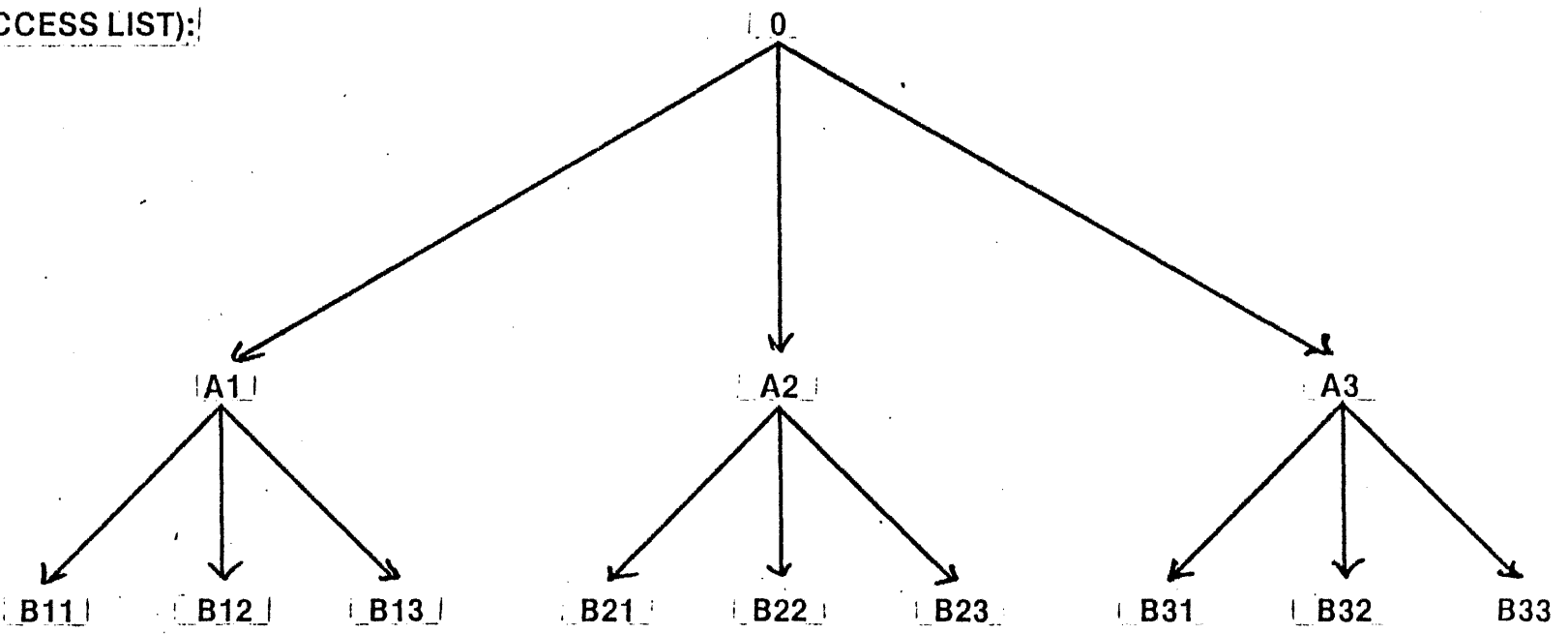


FIGURE 2-5 A NETWORK OF OBJECTS AS AN OBJECT

2-5

2-5

## Two-Level Object Typing

Objects defined by the 432 architecture have two levels of typing:

- o A base type, which is either data segment or access segment. Data segments cannot contain access descriptors; access segments contain only access descriptors.
- o A system type, of which there are several: processor, process, domain, context, and dispatching port are but a few system types.

## Data Segments

Data segments can contain anything but access descriptors. In fact, they can contain dummy copies of access descriptors for purposes of inspection, but the dummy copies will not "work" as access descriptors; if so referenced, the 432 hardware will not permit the operation to take place.

Data segments are used to hold instructions (as objects of system type instruction segment), scalar data types, and any type of data structure except an access descriptor. In order to access a data segment, a procedure must have an

access descriptor for the data segment.

## Access Segments

An access segment (sometimes called an "access list") is a hardware-recognized array of access descriptors. In order to access an access segment, a procedure must have an access descriptor for the access segment itself.

Symbology. In this manual, access segments are symbolized as shown in Figure 2-6.

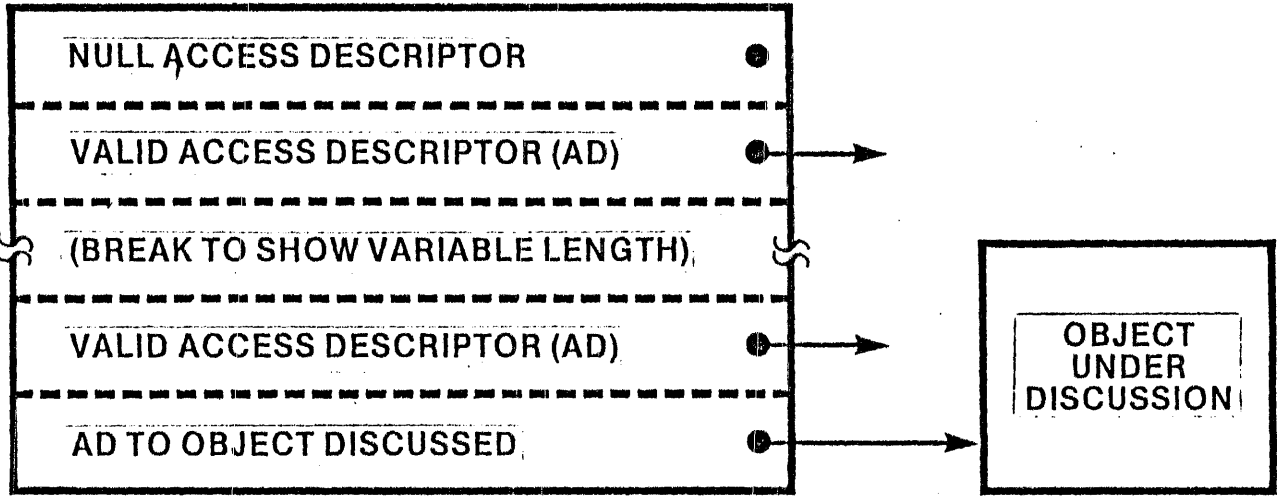
Figure 2-6. Access Segment Symbology.

## OBJECT LOCKING/EXCLUSION

### Object Locking

Either software or hardware can lock (obtain exclusive access to) an object when required by a sequence of instructions or microinstructions. Object can be locked in three ways:

- o Long-term software lock -- used when a software operation requires that an object be locked for a



27A

FIGURE 1-6 ACCESS SEGMENT SYMBOLOLOGY

2-6

period of time longer than the duration of one instruction. Accomplished using the Lock Object/Unlock Object instructions.

- o Short-term software lock -- used by a processor when executing an instruction that requires object locking for a period of time less than the duration of an instruction.
  
- o Short-term hardware lock -- used by a processor when performing an operation on its own behalf that requires an object to be locked.



## CHAPTER 3

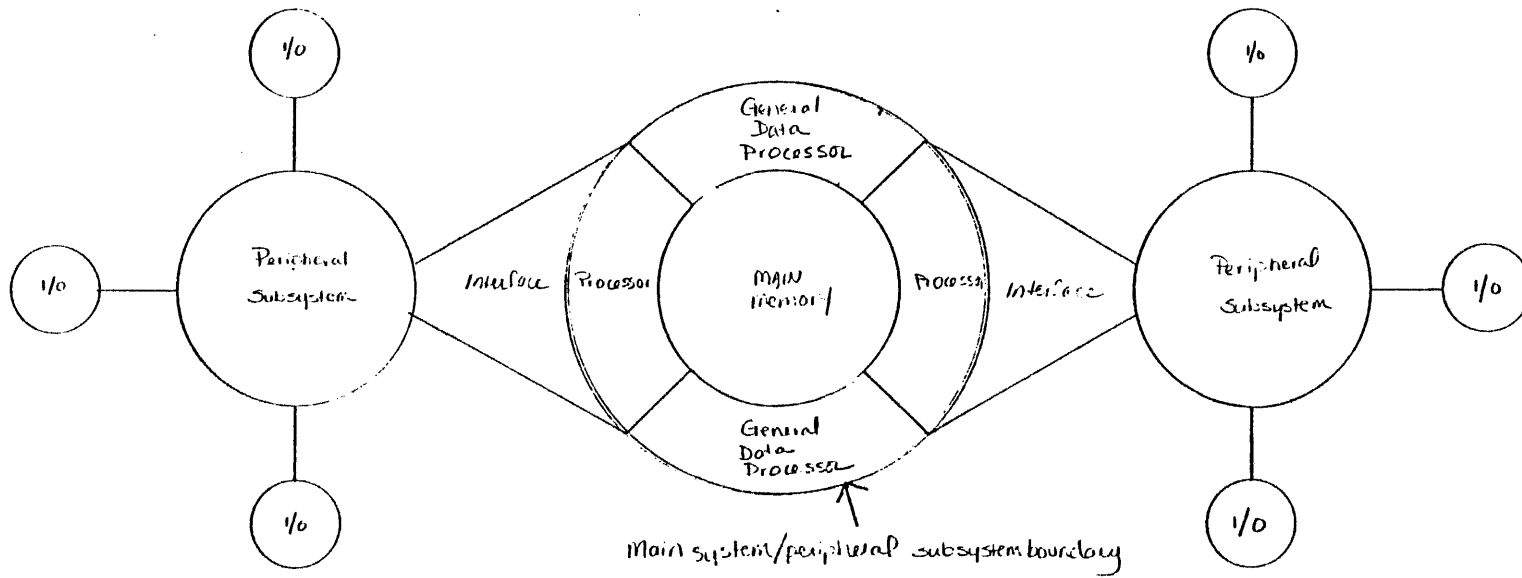
### INTRODUCTION TO 43203 INTERFACE PROCESSOR

This introduction to the architecture of the Intel 43203 Interface Processor (IP) includes four topics. The first introduces a basic I/O model. The second provides a brief discussion of the 43201 and 43202 General Data Processor (GDP) interface. (Refer to section 2 for a more detailed discussion). The peripheral subsystem/IP interface is the third topic, which provides information concerning the software interface as well as the hardware interface. The final topic introduces the supplementary IP facilities, including the physical reference mode and the interconnect access mode.

For a more detailed presentation of the following concepts, refer to the iAPX Interface Processor Architecture Reference Manual, Order no. TBS)

#### A BASIC I/O MODEL

A typical application based on the iAPX 432 microprocessor family consists of a main system and one or more peripheral subsystems. Figure 3-1 illustrates a hypothetical configur-



25A

Fig 3-1 Main System And Peripheral Subsystems

FIG. 3-1 MAIN SYSTEM AND PERIPHERAL SUBSYSTEM

ation that employs two peripheral subsystems. The main system hardware is composed of one or more iAPX 432 general data processors (GDPs) and a common memory that is shared by these processors. The main system software is a collection of one or more processes that execute on the GDP(s).

A fundamental principle of the iAPX 432 architecture is that the main system environment is self-contained; neither processors nor processes have any direct contact with the "outside world." Conceptually, the main system is enclosed by a wall that protects objects in memory from possible damage by uncontrolled I/O operations.

#### Figure 3-1 Main System and Peripheral Subsystems

In an iAPX 432-based system, the bulk of processing required to support input/output operations is delegated to peripheral subsystems; this includes device control, timing, interrupt handling and buffering. A peripheral subsystem is an autonomous computer system with its own local memory, I/O devices and controllers, at least one processor, and software. The number of peripheral subsystems employed in any given application depends on the I/O-intensiveness of the application, and may be varied with changing needs, independent of the number of GDPs in the system.

A peripheral subsystem resembles a mainframe channel in that it assumes responsibility for low-level I/O device support, executing in parallel with main system processor(s). Unlike a simple channel, however, each peripheral subsystem can be configured with a complement of hardware and software resources that precisely fits application cost and performance requirements. In general, any system that can communicate over a standard 8- or 16-bit microcomputer bus such as Intel's Multibus design may serve as an iAPX 432 peripheral subsystem.

A peripheral subsystem is attached to the main system by means of an IP. At the hardware level, the IP presents two separate bus interfaces. One of these is the standard iAPX 432 Packetbus and the other is a very general interface that can be adapted to most traditional 8-and 16-bit microcomputer buses.

To support the transfer of data through the wall that separates a peripheral subsystem from the main system, the IP provides a set of software-controlled windows. A window is used to expose a single object in main system memory so that its contents may be transferred to or from the peripheral subsystem.

The IP also provides a set of functions that are invoked by

software. While the operation of these functions varies considerably, they generally permit objects in main system memory to be manipulated as entities, and enable communication between main system processes and software executing in a peripheral subsystem.

It is important to note that both the window and function facilities utilize and strictly enforce the standard iAPX 432 addressing and protection systems. Thus, a window provides protected access to an object, and a function provides a protected way to operate in the main system. The IP permits data to flow across the peripheral subsystem boundary while preserving the integrity of the main system.

As figure 3-2 illustrates, input/output operations in an iAPX 432 system are based on the notion of passing messages between main system processes and device interfaces located in a peripheral subsystem. A device interface is considered to be the hardware and software in the peripheral subsystem that is responsible for managing an I/O device. An I/O device is considered to be a "data repository," which may be a real device (e.g., a terminal), a file, or a pseudo-device (e.g., a spooler).

A message sent from a process that needs an I/O service contains information that describes the requested operation (e.g., "read file XYZ"). The device interface interprets

31A

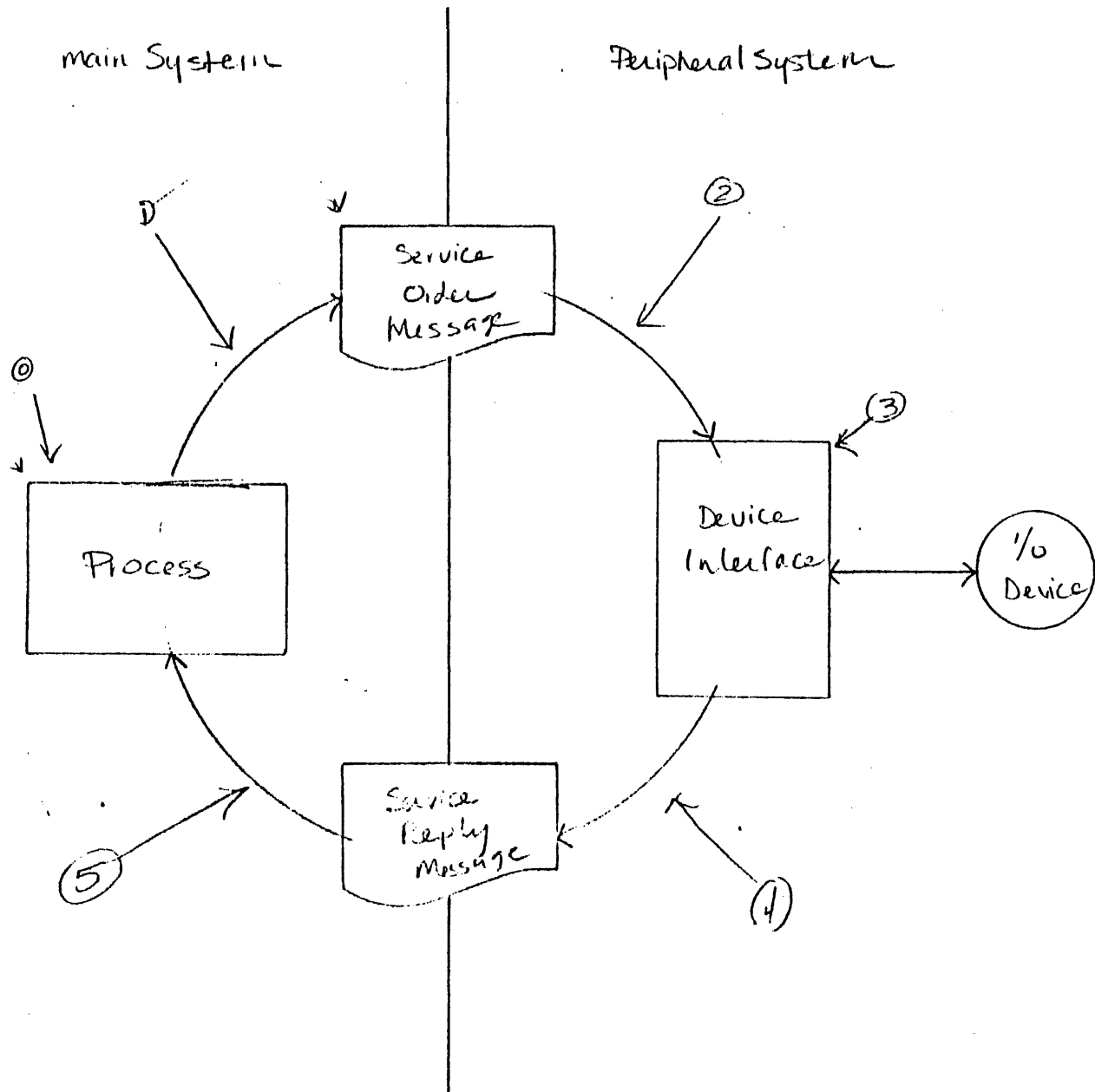


FIGURE 3-2 Basic I/O Service Cycle

the message and carries out the operation. If an operation requests input data, the device interface returns the data as a message to the originating process. The device interface may also return a message to positively acknowledge completion of a request.

A very general and very powerful mechanism for passing messages between processes is inherent in the iAPX 432 architecture. A given peripheral subsystem may, or may not, have its own message facility, but in any case, such a facility will not be directly compatible with the iAPX 432's. By interposing a peripheral subsystem interface at the subsystem boundary, the standard IP communication system can be made compatible with any device interface (see figure 3-2).

Figure 3-2 Basic I/O Service Cycle

Figure 3-3 Peripheral Subsystem Interface

### **iAPX 432 INTERFACE**

The IP exists in both the protected environment of the iAPX 432, and the conventional environment of the external

32A

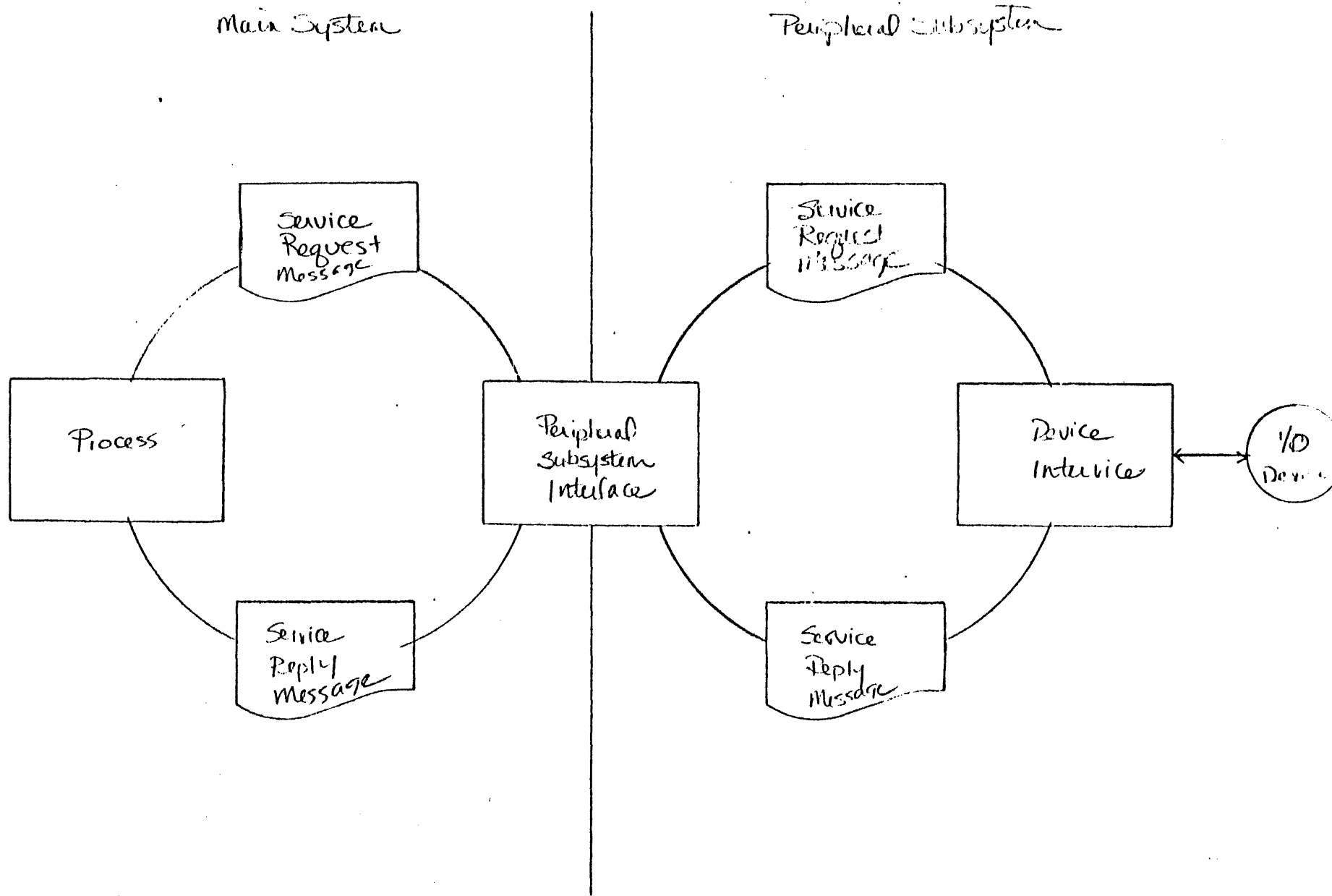


FIG. 3-3 Peripheral Subsystem Interface



subsystem. Because of this, an IP is able to provide a pathway over which data may flow between the iAPX 432 system and the external subsystem. The IP operates at the boundary between the systems, providing compatibility and protection. In this position, the Interface Processor presents two different views of itself, one to software and processors in the iAPX 432 environment and another to its external processor.

From the iAPX 432 side, an IP looks and behaves very much like any other processor. It attaches to the processor packetbus in the same way as a GDP. Within the iAPX 432 memory, the IP supports an execution environment that is compatible with, and largely identical to, the GDP. Thus, the IP recognizes and manipulates system objects representing processors, processes, ports, etc. It supports and enforces the iAPX 432's access control mechanisms, and provides full interprocess and interprocessor communication facilities.

The principle difference between the two processors is that the GDP manipulates its environment in response to the instruction it fetches, while the IP operates under the direction of its external processor. Indeed, the IP may be said to extend the instruction set of the attached processor (AP) so that it may function in the environment of the iAPX 432 system

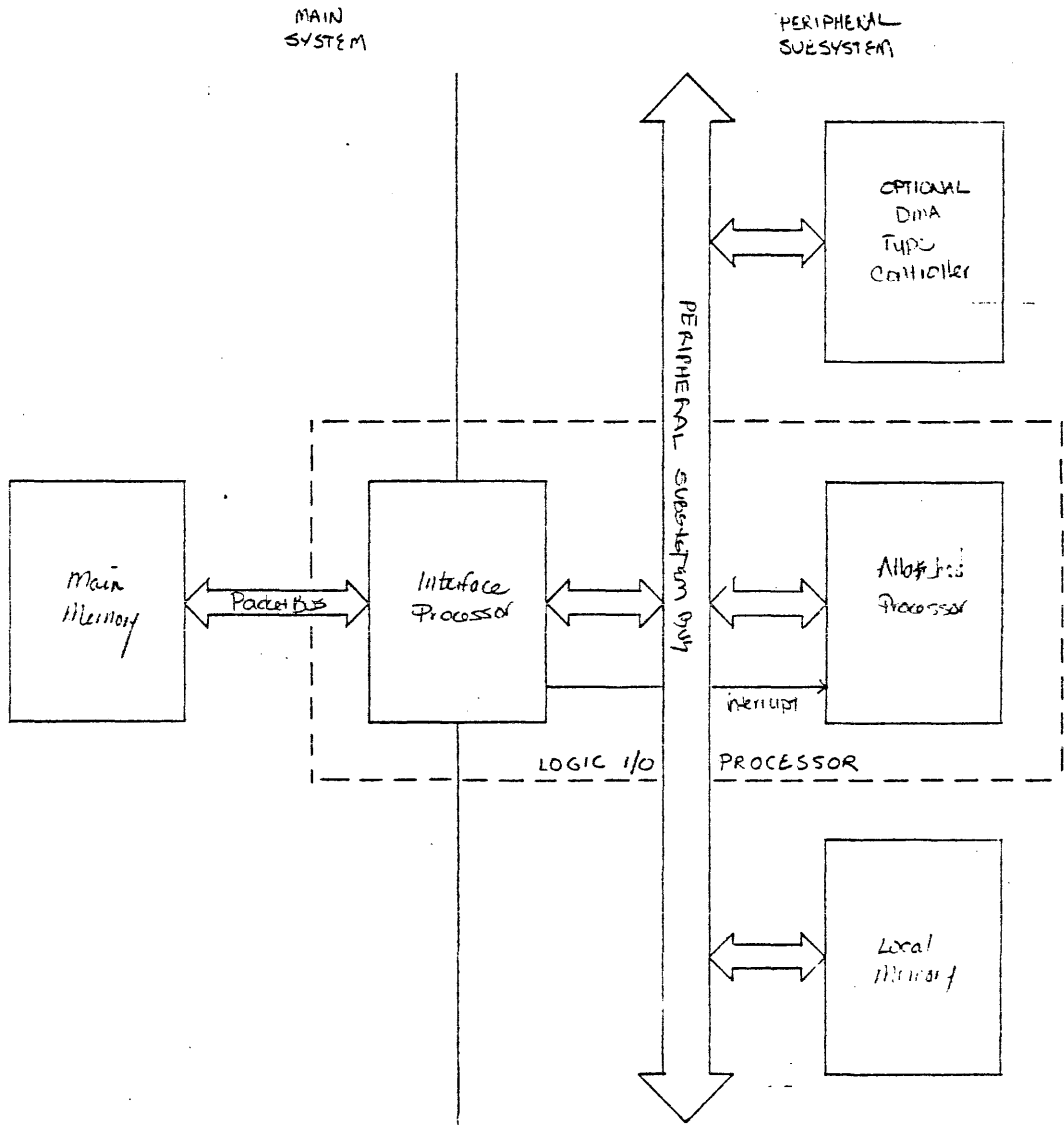
## PERIPHERAL SUBSYSTEM INTERFACE

A peripheral subsystem interface (PSI) is a collection of hardware and software that acts as an adapter that enables message-based communication between a process in the main system and a device interface in a peripheral subsystem (see figure 3-3). Viewed from the iAPX 432 side, the peripheral subsystem interface appears to be a process. The peripheral subsystem interface may be designed to present any desired appearance to a device interface. For example, it may look like a collection of tasks, or like a collection of subroutines.

### Hardware

The PSI hardware consists of an IP, an AP, and local memory (see figure 3-4 ). To improve performance, these may be augmented by a DMA controller. The AP and the IP work together as a team, each providing complementary facilities. Considered as a whole, the AP/IP pair may be thought of as a logical I/O processor that supports software operations in both the main system and the peripheral subsystem.

**Attached Processor** -- Almost any general-purpose CPU, such as an 8085, an iAPX 86 or an iAPX 88 can be used as an AP. The AP need not be dedicated exclusively to working with the IP.



Peripheral Subsystem Interface Hardware

FIGURE 3-4 PERIPHERAL SUBSYSTEM INTERFACE HARDWARE  
301 A

It may, for example, also execute device interface software. Thus, the AP may be the only CPU in the peripheral subsystem, or it may be one of several.

As figure 3-4 shows, the AP is "attached" to the interface processor in a logical sense only. The physical connections are standard bus signals and one interrupt line (which would typically be routed to the AP via an interrupt controller).

Continuing the notion of the logical I/O processor, the AP fetches instructions, and provides the instructions needed to alter the flow of execution, and to perform arithmetic, logic and data transfer operations within the peripheral subsystem.

#### Figure 3-4 Peripheral Subsystem Interface Hardware

**Interface Processor** -- The IP completes the logical I/O processor by providing data paths between the peripheral subsystem and the main system, and by providing functions that effectively extend the AP's instruction set so that software running on the logical I/O processor can operate in the main system. Since these facilities are software-controlled, they are discussed in the next section.

As figure 3-4 shows, the IP presents both a peripheral subsystem bus interface and a standard iAPX 432 Packetbus interface. By bridging the two buses, the IP provides the hardware link that permits data to flow under software control between the main system and the peripheral subsystem.

The IP connects to the main system in exactly the same way as a GDP. Thus, in addition to being able to access main memory, the IP supports other iAPX 432 hardware-based facilities, including processor communication, the alarm signal and functional redundancy checking.

On the I/O subsystem side, the IP provides a very general bus interface that can be adapted to any standard 8- or 16-bit microprocessor bus, including Intel's Multibus architecture, as well as the component buses of the MCS-85 and iAPX 86 families. The IP is connected to the peripheral subsystem bus as if it were a memory component; it occupies a block of memory addresses up to 64K bytes long. Like a memory, the IP behaves passively within the peripheral subsystem (except as noted below). It is driven by peripheral subsystem memory references that fall within its address range.

While the IP generally responds like a memory component, it also provides an interrupt request signal. The interface

processor uses this line to notify its AP that an event has occurred which requires its attention.

To summarize, the AP and the IP interact with each other by means of address references generated by the AP and interrupt requests generated by the IP. Since the IP responds to memory references, other active peripheral subsystem agents (bus masters), such as DMA controllers, may obtain access to main system memory via the IP.

### Software

**IP Controller** -- The peripheral subsystem interface is managed by software, referred to as the IP controller. The IP controller executes on the AP and uses the facilities provided by the AP and the IP to control the flow of data between the main system and the peripheral subsystem.

While there are no actual constraints on the structure of the IP controller, organizing it as a collection of tasks running under the control of a multitasking operating system (such as RMX-80 or iRMX-86) can simplify software development and modification. This type of organization supports asynchronous message-based communication within the IP controller, similar to the iAPX 432's intrinsic interprocessor communication facility. Extending this approach to the device interface as well results in a consistent,

system-wide communication model. However, communication within the IP controller and between the IP controller and device interfaces, is completely application-defined. It may also be implemented via synchronous procedure calls, with "messages" being passed in the form of parameters.

However it is structured, the IP controller interacts with the main system through facilities provided by the interface processor. There are three of these facilities: execution environments, windows, and functions.

**Execution Environments** -- The IP provides an environment within the main system that supports the operation of the IP controller in that system. This environment is embodied in a set of system objects that are used and manipulated by the IP. At any given time, the IP controller is represented in main memory by a process object and a context object. Like a GDP, the IP itself is represented by a processor object. Representing the IP and its controlling software like this creates an execution environment that is analogous to the environment of a process running on a GDP. This environment provides a standard framework for addressing, protection and communication within the main system.

Like a GDP, an IP actually supports multiple process environments. The IP controller selects the environment in which a function is to be executed. This permits, for

example, the establishment of separate environments corresponding to individual device interface tasks in the peripheral subsystem. If an error occurs while the IP controller is executing a function on behalf of one device interface, that error is confined to the associated process, and processes associated with other device interfaces are not affected.

**Windows** -- Every transfer of data between the main system and a peripheral subsystem is performed with the aid of an IP window. A window defines a correspondence, or mapping, between a subrange of peripheral subsystem memory addresses (within the range of addresses occupied by the IP) and an object in main system memory (see figure 3-5). When an agent in the peripheral subsystem (e.g., the IP controller) reads a local windowed address, it obtains data from the associated object; writing into a windowed address transfers data from the peripheral subsystem to the windowed object. The action of the IP, in mapping the peripheral subsystem address to the main system object, is transparent to the agent making the reference; as far as it is concerned, it is simply reading or writing local memory.

Figure 3-5 Interface Processor Window



FIGURE 3-5 INTERFACE PROCESSOR WINDOW

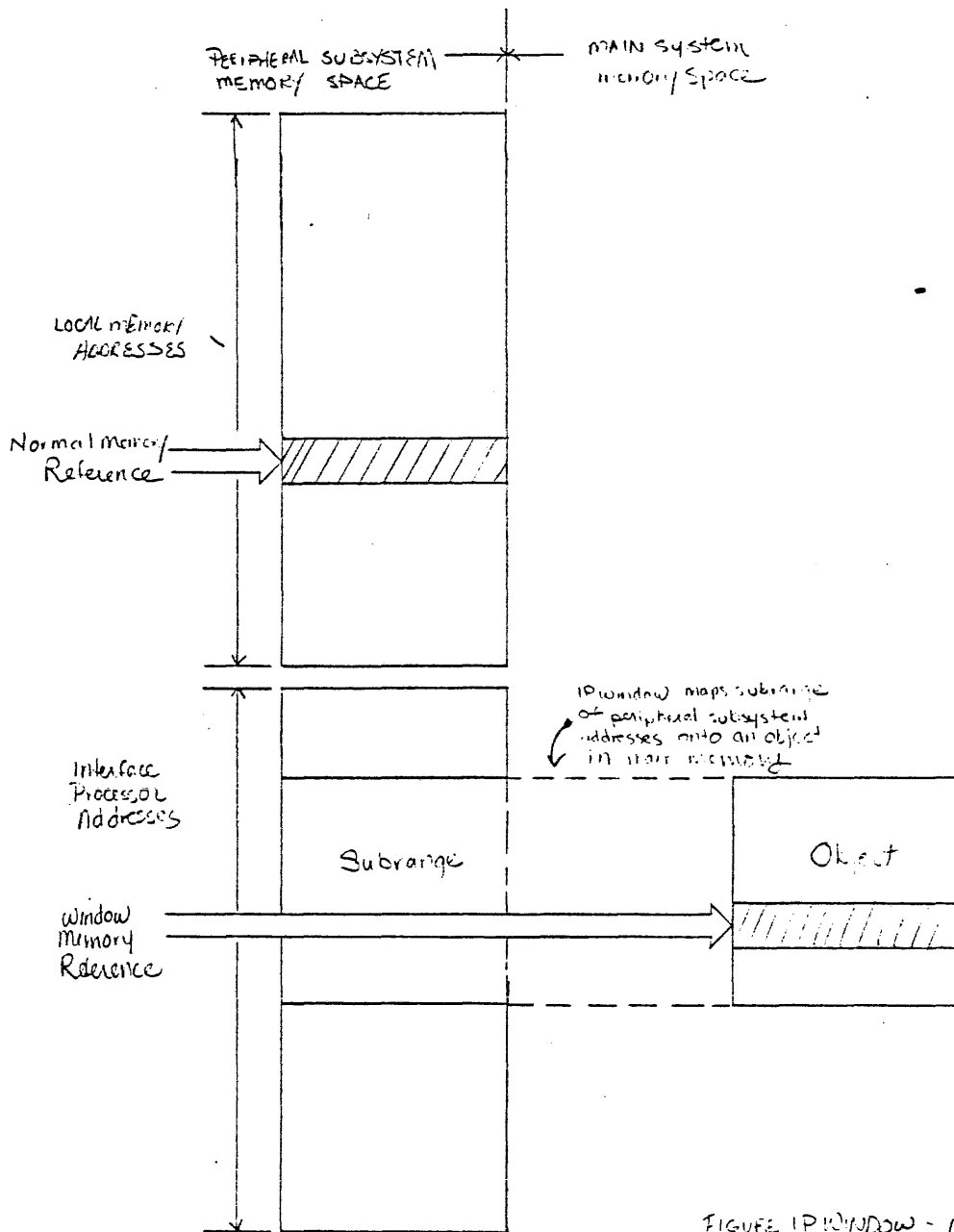


FIGURE 1P WINDOW - Interface Processor

Since a window is referenced like local memory, any individual transfer may be between an object and local memory, an object and a processor register, or an object and an I/O device. The latter may be appealing from the standpoint of "efficiency," but it should be considered with caution. Using a window to directly "connect" an I/O device and an object in main memory has the undesirable effect of propagating the real-time constraints imposed by the device beyond the subsystem boundary into the main system. It may seriously complicate error recovery as well. Finally, since there is a finite number of windows, most applications will need to manage them as scarce resources that will not always be instantly available. This means that at least some I/O device transfers will have to be buffered in local memory until a window becomes available. It may be simplest to buffer all I/O device transfers and use the windows to transfer data between local memory and main system memory.

There are four IP windows that may be mapped onto four different objects. The IP controller may alter the windows during execution to map different subranges and objects. References to windowed subranges may be interleaved and may be driven by different processors in the peripheral subsystem. For example, the AP and a DMA controller may be driving transfers concurrently, subject to the same bus arbitration constraints that would apply if they were accessing local memory.

**Functions** -- A fifth window provides the IP controller with access to the IP's function facility. By writing operands and an opcode into predefined locations in this window's subrange, the IP controller requests the IP to execute a function on its behalf. This procedure is very similar to writing commands and data to a memory-mapped peripheral controller (e.g., a floppy disk controller). Upon completion of the function, the IP provides status information that the IP controller can read through the window. The IP can perform transfer requests through the other four windows while it is executing a function.

The IP's function set permits the IP controller to:

- o alter windows;
- o exchange messages with GDP processes via the standard IP communication facility;
- o manipulate objects.

These functions may be viewed as instruction set extensions to the AP, which permit the IP controller to operate in the main system. The combination of the IP's function set and windows, the AP's instruction set, and possibly additional facilities provided by a peripheral subsystem operating

system, permits the construction of powerful IP controllers that can relieve the main system of much I/O-related processing. At the same time, by utilizing only a subset of the available IP functions, relatively simple IP controllers can also be built (in cases where this approach is more appropriate).

#### **SUPPLEMENTARY INTERFACE PROCESSOR FACILITIES**

The preceding sections describe the IP as it is used most of the time. The IP provides two additional capabilities that are typically used less frequently, and only in exceptional circumstances. These are physical reference mode and interconnect access.

##### **Physical Reference Mode**

The IP normally operates in logical reference mode; this mode is characterized by its object-oriented addressing and protection system. There are times when logical referencing is impossible because the objects used by the hardware to perform logical-to-physical address development are absent (or, less likely, are damaged). In these situations the IP can be used in physical reference mode.

In physical reference mode, the IP provides a reduced set of functions. Its windows operate as in logical reference

mode, except that they are mapped onto memory segments (rather than objects) that are specified directly with 24-bit addresses. In this respect, physical reference mode is similar to traditional computer addressing techniques.

Physical reference mode is most often employed during system initialization to load binary images of objects from a peripheral subsystem into main memory. Once the required object images are available, processors can begin normal logical reference mode operations.

#### **Interconnect Access**

In addition to memory, the iAPX 432 architecture defines a second address space called the processor-memory interconnect. One of the IP windows is software-switchable to either space. In logical reference mode, the interconnect space is addressed in the same object-oriented manner as the memory space, with the IP automatically performing the logical-to-physical address development. In physical reference mode, the interconnect space is addressed as an array of 16-bit registers, with a register selected by a 24-bit physical address.

## CHAPTER 4

### iAPX 432 PROCESSOR ENVIRONMENT DEFINITION

This chapter describes the requirements placed on the logical structure of the iAPX 432 hardware environment. These requirements are concerned directly with the constraints of local memory, the type of data transferred (address, control, or data), and the structure of the data types.

The first section presents the information structure for an iAPX 432 system and includes a discussion of memory system requirements, physical addressing, data formats, data representation, and hardware error detection. The second section presents the elements of the processor packet bus and includes associated timing diagrams for Read, Write, and Access cycles.

### iAPX 432 INFORMATION STRUCTURE

The iAPX 432 system contains both read/write (RAM) and read-only (ROM) memory. Any attached processor (8 bit or 16 bit) in the system can access all the contents of physical

memory. This section describes how information is represented and accessed.

## **Memory**

The iAPX 432 implements a two-level memory structure. The software system exists in a segmented environment in which a logical address specifies the location of a data item. The processor automatically translates this logical address into a physical address for accessing the value in physical memory.

## **Physical Addressing**

Logical addresses are translated by the processor into physical addresses. Physical addresses are transmitted to memory by a processor to select the beginning byte of a memory value to be referenced. A physical address is 24 binary bits in length. This results in a physical memory of over 16.5 Megabytes.

## **Data Formats**

When a processor executes the instructions of an operation within a context, operands found in the segments of the context may be manipulated. An individual operand may occupy one, two, four, eight, or ten bytes of memory (byte, double

byte, word, double word, or extended word, respectively). All operands are referenced by a logical address as described above. The displacement in such an address is the displacement in bytes from the base address of the data segment to the first byte of the operand. For operands consisting of multiple bytes, the address locates the low-order byte while the higher-order bytes are found at the next higher consecutive addresses.

### **Data Representation**

An iAPX 432 convention has been adopted for representing data structures stored in memory. The bits in a field are numbered by increasing numeric significance, with the least-significant bit shown on the right. Increasing byte addresses are shown from right to left. Examples of the five basic data lengths used in the iAPX 432 system are shown in figure 4-1.

Figure 4-1

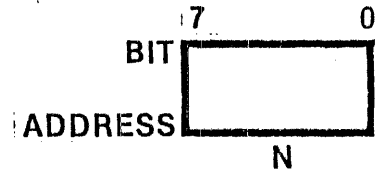
### **Data Positioning**

The data structures shown in figure 4-1 may be aligned on an arbitrary byte boundary within a data segment. Note that more efficient system operation may be obtained when multi-byte data structures are aligned on double-byte boundaries



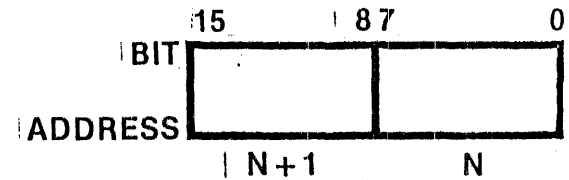
FIGURE 4-1 ~~DATA~~ BASIC 14PX 432 DATA LENGTHS

BYTE

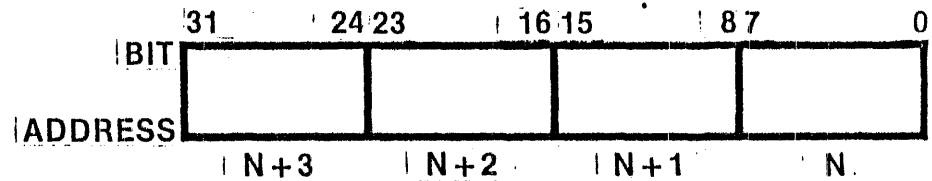


DOUBLE BYTE

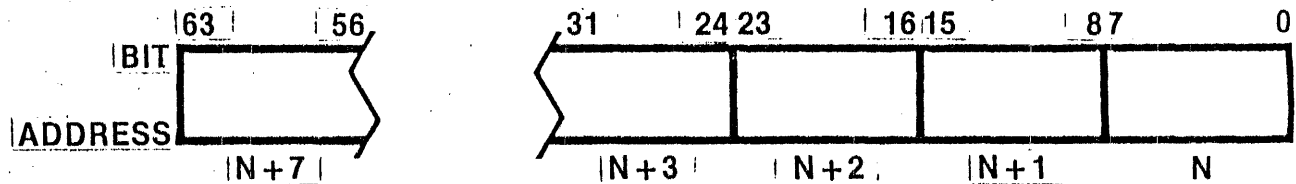
46A



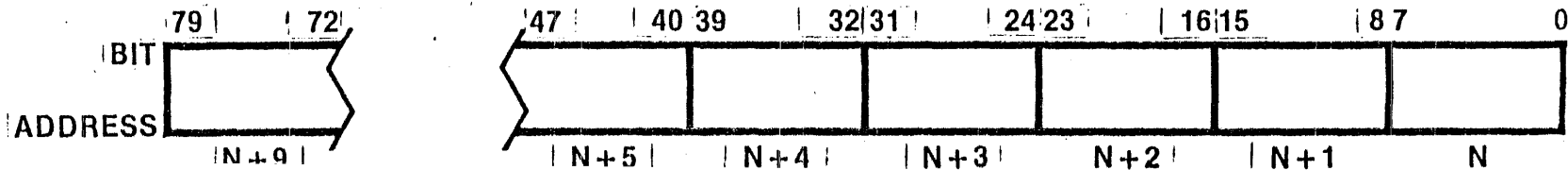
WORD



DOUBLE WORD



EXTENDED WORD



(if the memory system is organized in units of double bytes).

### Requirements of an iAPX 432 Memory System

The multiprocessor architecture of the iAPX 432 places certain requirements on the operation of the memory system to ensure the integrity of data items that can potentially be accessed simultaneously. Indivisible read-modify-write (RMW) operations to both double-byte and word operands in memory are necessary for manipulating system objects. When an RMW-read is processed for a location in memory, any other RMW-reads from that location must be held off by the memory system until an RMW-write to that location is received (or until an RMW timeout occurs). Note that while the memory system is writing the RMW-write, any other types of reads and writes are allowed. Also, for ordinary reads and writes of double-byte or longer operands, the memory system must ensure the entire operand has been either read or written before beginning to process another access to the same location; e.g., if two simultaneous writes to the same location occurs, the set of locations used to store the operand could contain some interleaved combination of the two written values.

### iAPX 432 Hardware Error Detection

iAPX 432 processors include a facility to support the hardware detection of functional errors. At INIT/ time each iAPX 432 processor is configured to operate as either a master or checker processor. A master operates in the normal manner. A checker places all output pins that are being checked into a high-impedance state. Thus a master and checker processor may be parallel-connected such that the checker is able to compare a master's output pin values to those computed in the checker. Any comparison error causes the checker to assert HERR/, FATAL/, and go idle. No further activity will occur at the disagreeing master-checker processor.

Figure 4-2 Hardware Error Detection

#### **PACKET BUS DEFINITION**

This section describes and defines the significance of the 19 signal lines that make up the processor packet bus, and the general scheme by which timing relationships on these lines are derived. Although this section defines all legal bus activities, the processors do not necessarily perform all allowed activities.

The packet bus consists of 3 control lines:

- o Packet bus Request (PRQ),

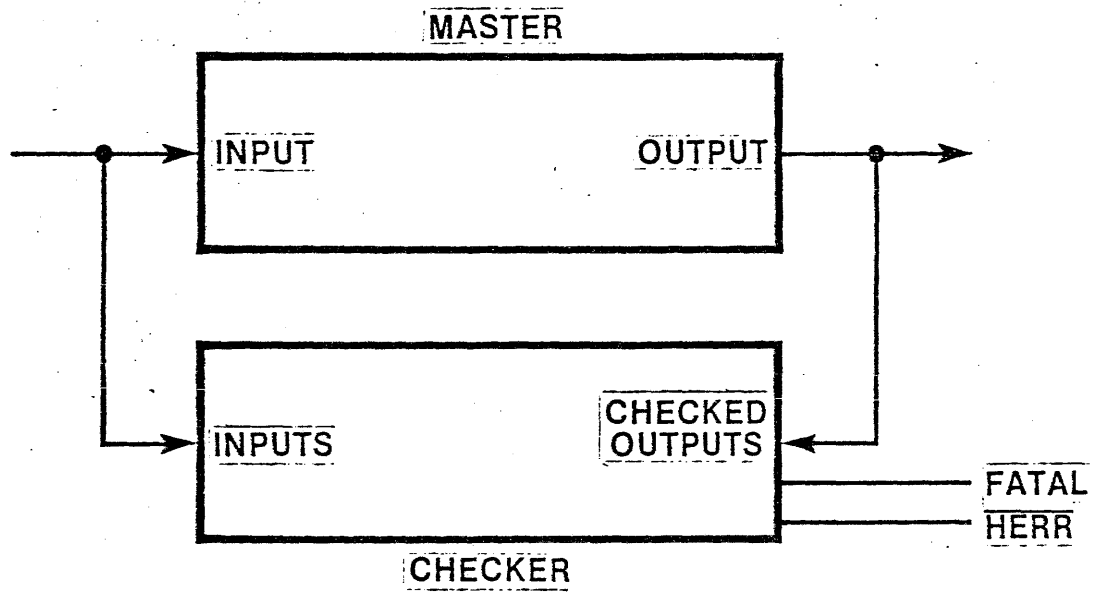


FIGURE 4-2 HARDWARE ERROR DETECTION

- o Enable Buffer Outputs ( BOUT).
  
- o Interconnect Status (ICS),

This bus also includes sixteen 3-state Address-Control-Data lines (ACD15 through ACDO). PRQ has two functions whose use depends upon the application, i.e., PRQ either indicates the first cycle of a transaction on the processor component bus or the cancellation of a transaction initiated in the previous cycle. Of the three control lines, BOUT has the simplest function, serving as a direction control for buffers in large systems requiring more electrical drive than the processor components can provide. The ICS signal has significance pertaining to one of three different system conditions and depends on the state of the processor component bus transaction. The processor interprets the ICS input as an indication of one of the following:

- o Whether or not an interprocess communication (IPC) is waiting,
  
- o Whether or not the slave requires more time to service the processors request,
  
- o Whether or not a bus ERROR has occurred.

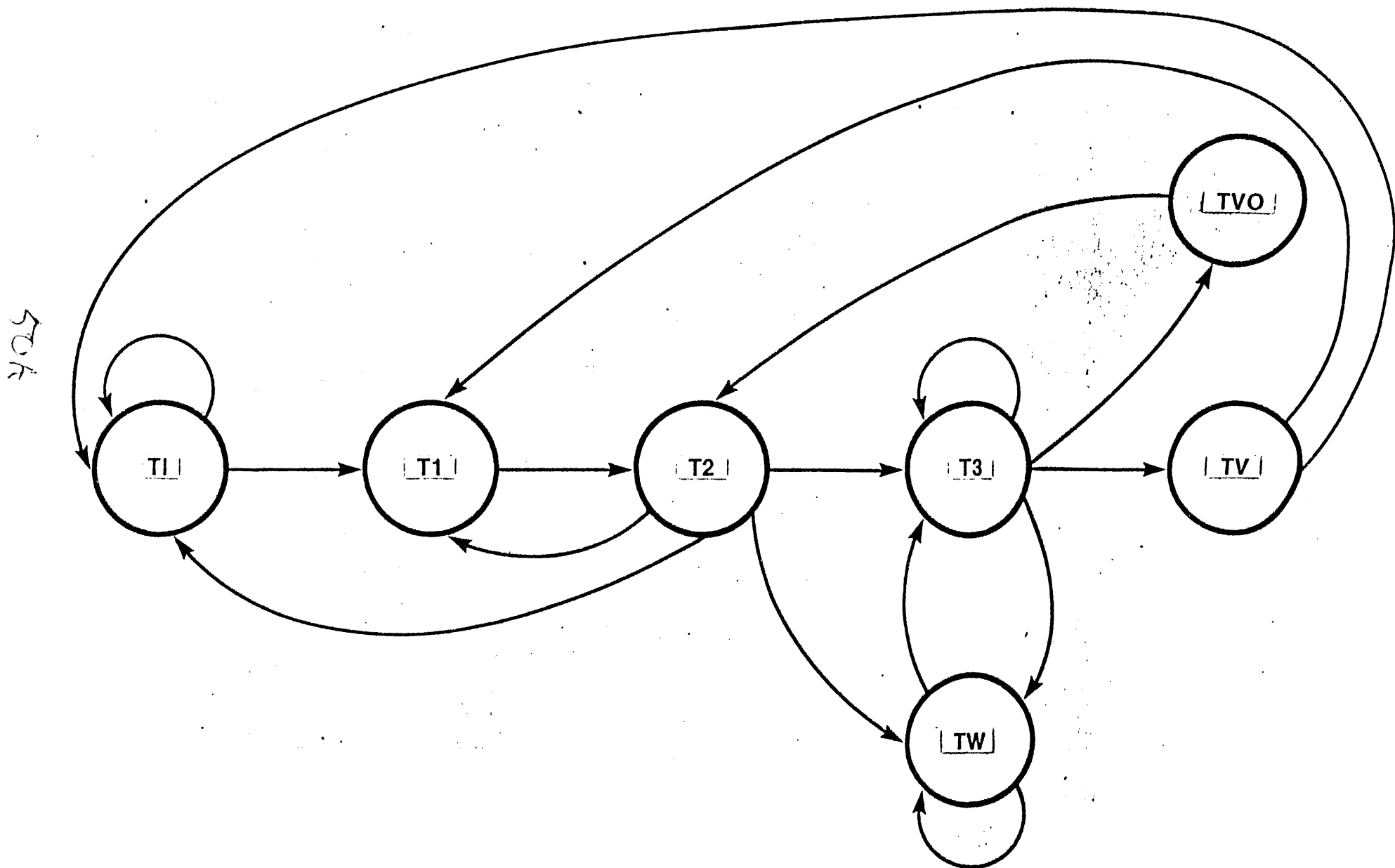
The Address/Control/Data lines emit output specification information to indicate the type of cycle being initiated, e.g., addresses, data to be written, or control information. They also receive data returned to the processor during reads. Details of the ACD line operation and the associated control lines are summarized below.

**ACD15 - ACDO (Address/Control/Data)** -- During the first cycle ( $T_1$  or  $T_{v0}$  (See Figure 4-3) of a processor component bus transaction (indicated by the rising edge of PRQ), the high-order 8 ACD bits (ACD15...ACD8) specify the type of the current transaction. In this first cycle, the low-order ACD bits (ACD7...ACD0) contain the least significant eight bits of the 24 bit physical address.

During the subsequent cycle ( $T_2$ ), the remainder of the address is present on the ACD pins (aligned such that the most significant byte of the address is on ACD15 through ACD8, the mid-significant byte on ACD7 through ACD0). If PRQ is asserted during  $T_2$ , the access is cancelled and the ACD line is not defined.

During the third cycle ( $T_3$  or  $T_w$ ) of a processor packet bus transaction the processor presents a high impedance to the ACD lines for read transactions and  $T_w$  or  $T_3$  asserts write data for write transactions.

FIGURE 4-3 STATE DIAGRAM FOR PROCESSOR PACKET BUS  
(a)







Once the bus has entered T3 or Tv, the sequence of state transactions depends on the type of cycle requested during the preceding T1 or Tvo. Accesses ranging in length from 1 to 32 bytes may be requested (see Table 4-1). If a transfer of more than one double byte has been requested, it is necessary to enter T3 for every double-byte that is transferred. After any transfer the processor may simply re-enter T3 or it may enter Tw for any number of cycles (as dictated by ICS) and the number of double bytes remaining to be transferred.

After all data is transferred, the processor enters either Tv or Tvo. Tvo can be entered only when the internal state of execution is such that the processor is prepared to accomplish an immediate write transfer (overlapped write). During Tvo, the ACD lines contain address and specification information aligned in the same fashion as in T1. If the processor does not require an overlapped write, the bus state moves to Tv (the ACD lines will be floating). After Tv, a new bus cycle can be started with T1, or the processor may enter the idle state(Ti).

**ICS (Interconnect Status)** -- ICS has three possible interpretations depending on the state of the bus transac-

ACD 15	ACD 14	ACD 13	ACD 12	ACD 11	ACD 10	ACD 9	ACD 8
Access	Op	RMW	Length			Modifiers	
0 - Memory	0 - Read	0 - Nominal	000 - 1 Byte	001 - 2 Bytes	010 - 4 Bytes	011 - 6 Bytes	100 - 8 Bytes
1 - Other	1 - Write	1 - RMW	101 - 10 Bytes	110 - 16 Bytes*	111 - 32 Bytes*		
			* Not implemented			ACD 15 = 0: 00-Inst Seg Access 01-Stack Seg Access 10-Context Ctl Seg Access 11-Other  ACD 15 = 1: 00-Reserved 01-Reserved 10-Reserved 11-Interconn Register	

Table ~~4-1~~ ACD Specification Encoding

4-1

TABLE 4-2

tion (see ~~Figure ICS~~). Notice that under most conditions ICS has IPC significance for more than one cycle. It is important to note that a valid low during any cycle with IPC significance will signal the processor that an IPC has been received. An iAPX 432 processor is required to record and service only one IPC at a time. Logic in the interconnect system must record and sequence multiple (possibly simultaneous) IPC occurrences to the processor. Thus the logic that forms ICS must accommodate global and local IPC arrivals and requests for reconfiguration as individual events:

1. Assert IPC significance on ICS for the arrival of an IPC.
2. When the iAPX 432 processor reads interconnect address register 2, it will respond to one of the status bits for the IPC signalled on ICS in the following order:

Bit 2 (1=reconfigure, 0=Do not reconfigure)

Bit 1 (1=global IPC arrived, 0= No global IPC)

Bit 0 (1= Local IPC arrived, 0= No local IPC)

3. The logic in the interconnect system must clear the highest order status bit that was serviced by the iAPX 432 processor and if additional IPC information has arrived the interconnect system logic must signal an additional IPC indication to the iAPX 432 processor.

**PRQ (Processor Packet Bus Request)** -- PRQ is normally low and can go high only during T1, T2 and Tvo. High levels during Tvo and T1 indicate the first cycle of an access. A high level during T2 indicates that the current cycle is to be cancelled. See ~~Figure PRQ~~ <sup>TABLE 4-3</sup>.

**BOUT (Enable Buffered Outputs)** -- BOUT is provided to control external buffers when they are present. Table BOUT and the waveforms show its state under various conditions. Note that high to low transitions of BOUT will occur during T3 (when required) and low to high during Ti (when required). Refer to TABLE 4-4,

figure 4-3 State Diagram for Processor Packet Bus

Table 4-1 ACD Specification Encoding

Figure 4-4 Nominal Write Cycle Timing

	LEVEL		STATE
	HIGH	LOW	
IPC	NONE	WAITING	Ti, T1, T2* See Note.
STRETCH	DON'T	STRETCH	T3, Tw
ERR	BUS ERROR	NO ERROR	Tv, Tvo

\*Note: ICS has no significance in a cycle following a T2 where PRQ is asserted (cancelled access) or in any cycle during which HERR/ is asserted.

TABLE 4-2 ~~XXXXXXXXXX~~ - ICS Interpretation

State	PRQ	Condition
Ti	0	Always
T1	1	Initiate access
T2	0	Continue access
	1	Cancel access
T3	0	Always
Tv	0	Always
Tvo	1	Initiate overlapped access

TABLE 4-3 ~~XXXXXXXXXX~~ - PRQ Interpretation

BOUT	High	Low
Write	Always	Never
Read	Ti, T1, T2	T3, Tv, Tw

TABLE 4-4 ~~XXXXXXXXXX~~ BOUT INTERPRETATION

iAPX 432 SIGNALLING SCHEME

	PROCESSOR		SLAVE	
Inputs Sampled	ACD: Others:	CLKA CLKA	A11:	CLKB
Outputs Driven	A11 (except BOUT): BOUT:	CLKA CLKA	ACD: Others:	CLKB CLKB

Table ~~4-5~~ iAPX 432 Component Signalling Scheme  
4-5

Figure 4-5 Stretched Write Cycle Timing

Figure 4-6 Minimum Write Cycle Timing

Figure 4-8 Minimum Read Timing/No External Buffers (BOUT not used)

Figure 4-7 Minimum Read Cycle (Buffered System)

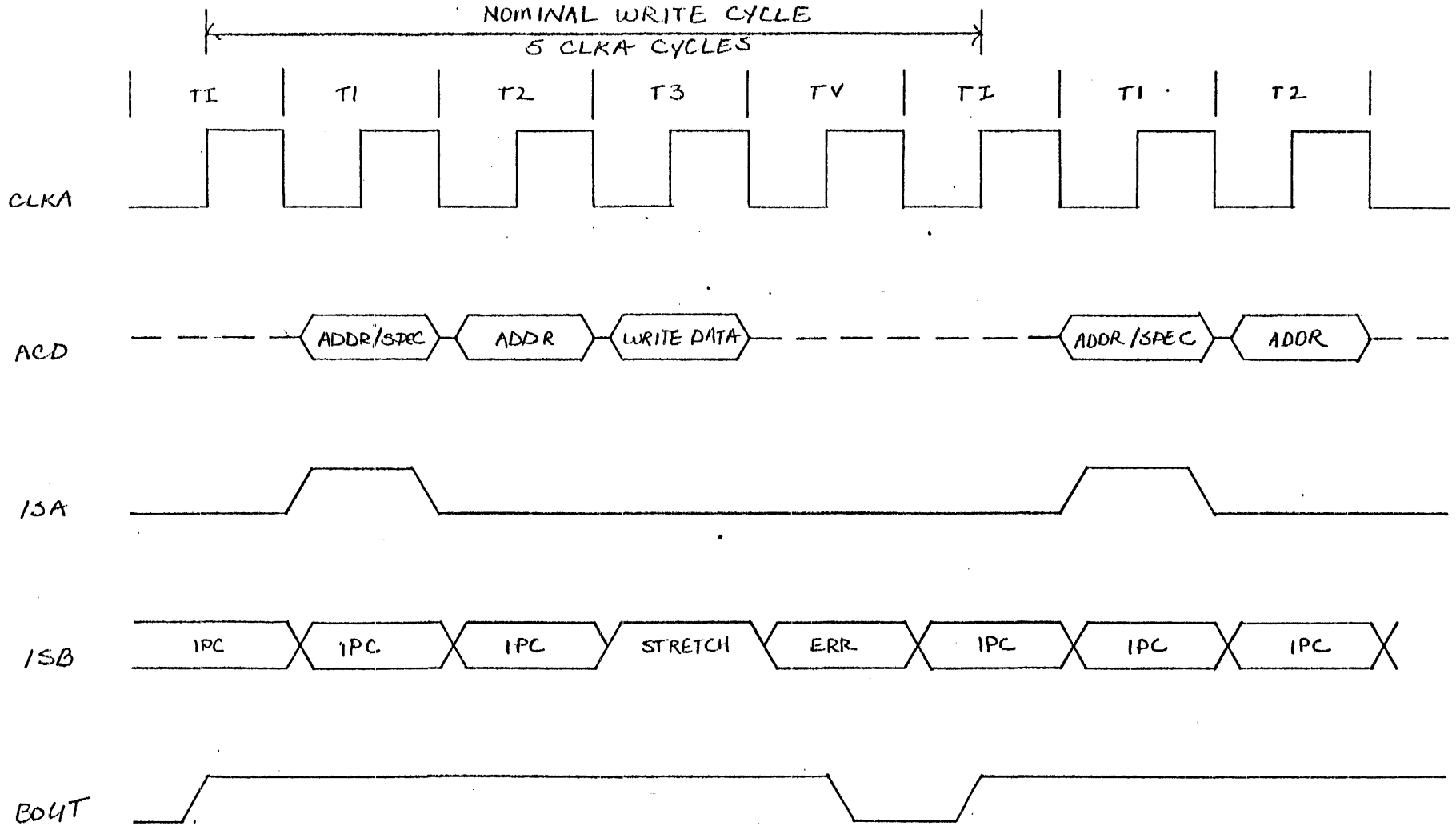
Figure 4-9 Minimum Faulted Access Cycle Timing (PRQ Cancellation)

#### **Processor Packet Bus Timing Relationships**

All timing relationships on the processor packet bus are derived from a simple scheme and related to Figure 4-3. Each timing diagram shown in the following pages provides a separate table illustrating the various system states during the cycle. This approach to transfer timing was designed to allow maximum time for the transfer to occur and yet guarantee hold time.

Any agent connected to the processor packet bus is recognized as either a processor or a slave. Examples of processors are the GDP and the IP. A memory system provides an example of a slave.

# FIGURE 4-4 NOMINAL WRITE CYCLE TIMING



ACD15	ACD6	ACD7	ACD0	State
hi z	hi z			Ti
spec	Lo-adr			T1
Hi-adr	Mid-adr			T2
Hidatal	Lodatal			T3
Hi z	Hi z			Tv
Hi z	Hi z			T1
Spec	Lo-adr			T1
Hi-adr	Mid-adr			T2

\*Undefined if Single byte write

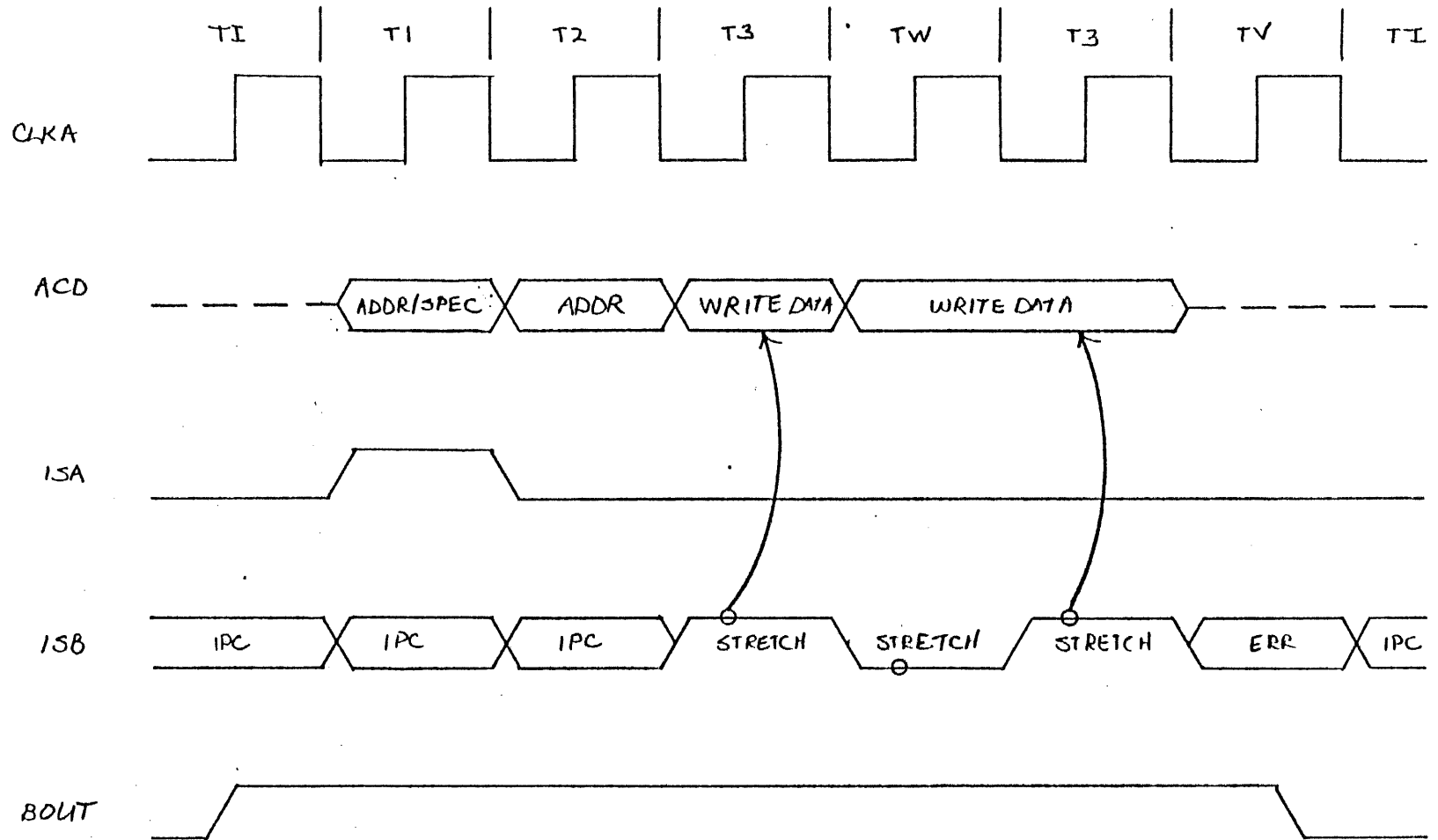
54A

(talk)

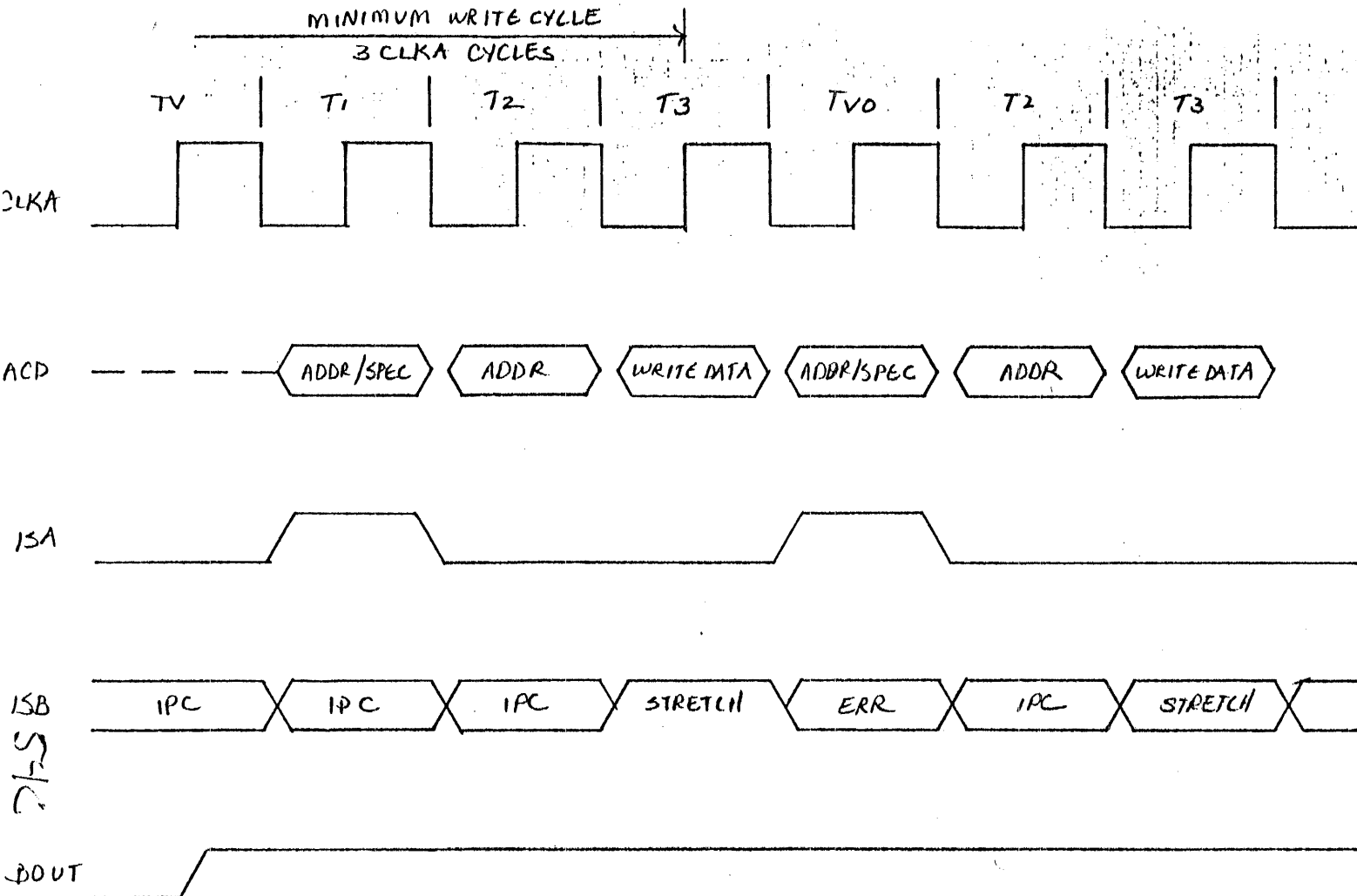


ACD15	ACD8	ACD7	ACD0	State
hi-z		hi z		T1
spec		Lo-adr		T1
Hi-adr		Mid-adr		T2
Hi-data1		Lo-data1		T3
Hi-data1		Lo-data1		TW
Hi-data2		Lo-data2		T3
Hi-z		Hi-z		Tv
Hi-z		Hi-z		Ti

FIGURE 4-5 STRETCHED WRITE CYCLE TIMING



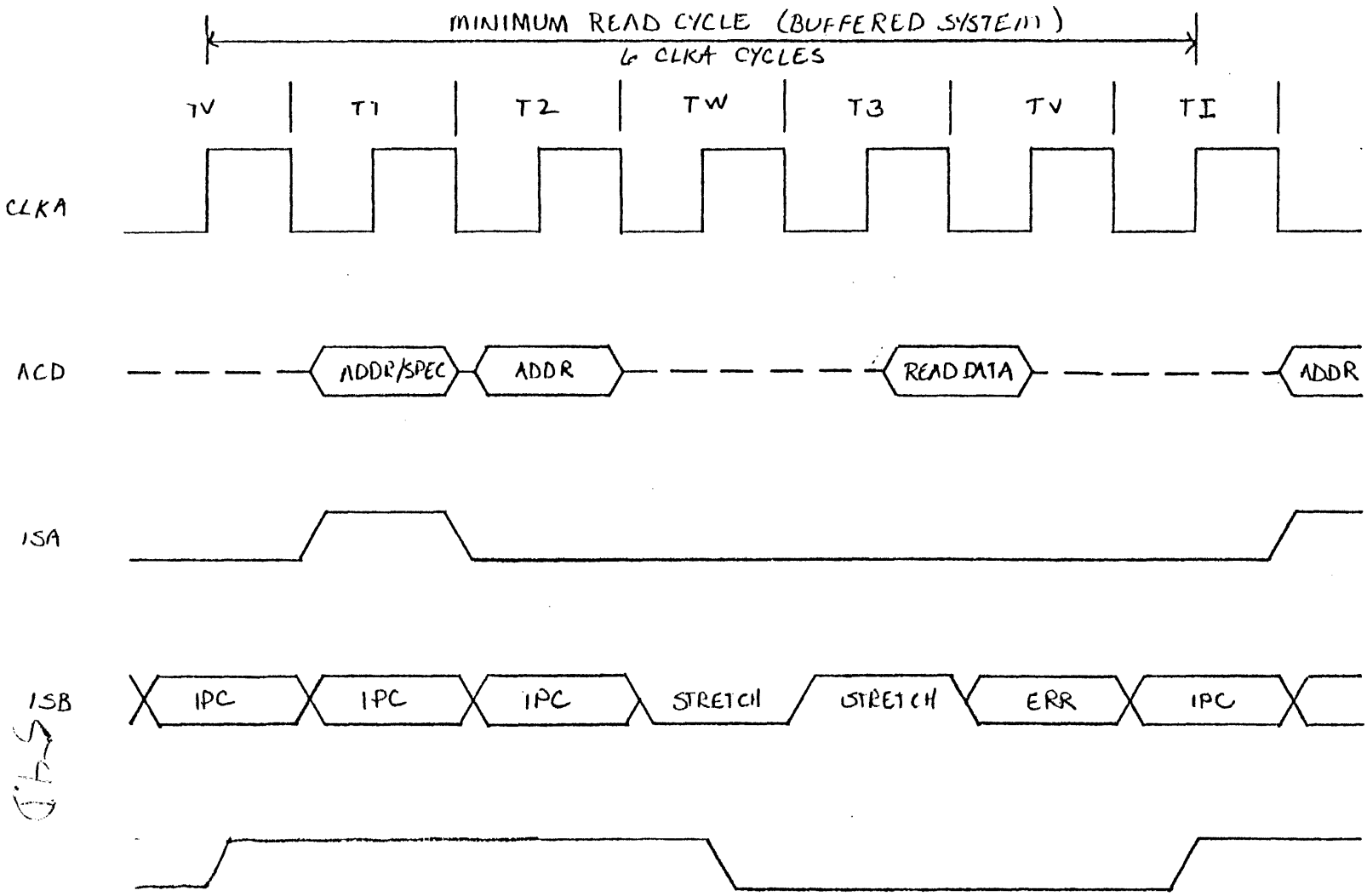
54B



ACD15	ACD8	ACD7	ACD0	State
Hi-Z	Hi-Z			Tv (Preceded by read cycle)
Spec	Lo-adr			T1
Hi-adr	Mid-adr			T2
Hi-datal*	Lo-datal			T3
Spec	Lo-adr			Tvo
Hi-adr	Mid-adr			T2
Hi-datal	Lo-datal			T3

\* Undefined if single byte write

FIGURE 4-6 MINIMUM WRITE CYCLE TIMING



(table)

ACD15	ACD8	ACD7	ACD0
Hi-Z		Hi-Z	
spec		Lo-adr	
Hi-adr		Mid-adr	
Hi-Z		Hi-Z	
Hi-datal*		Lo-datal	
Hi-Z		Hi-Z	
Hi-Z		Hi-Z	

State

T<sub>v</sub> (Preceded by a read)

T<sub>1</sub>

T<sub>2</sub>

T<sub>w</sub>

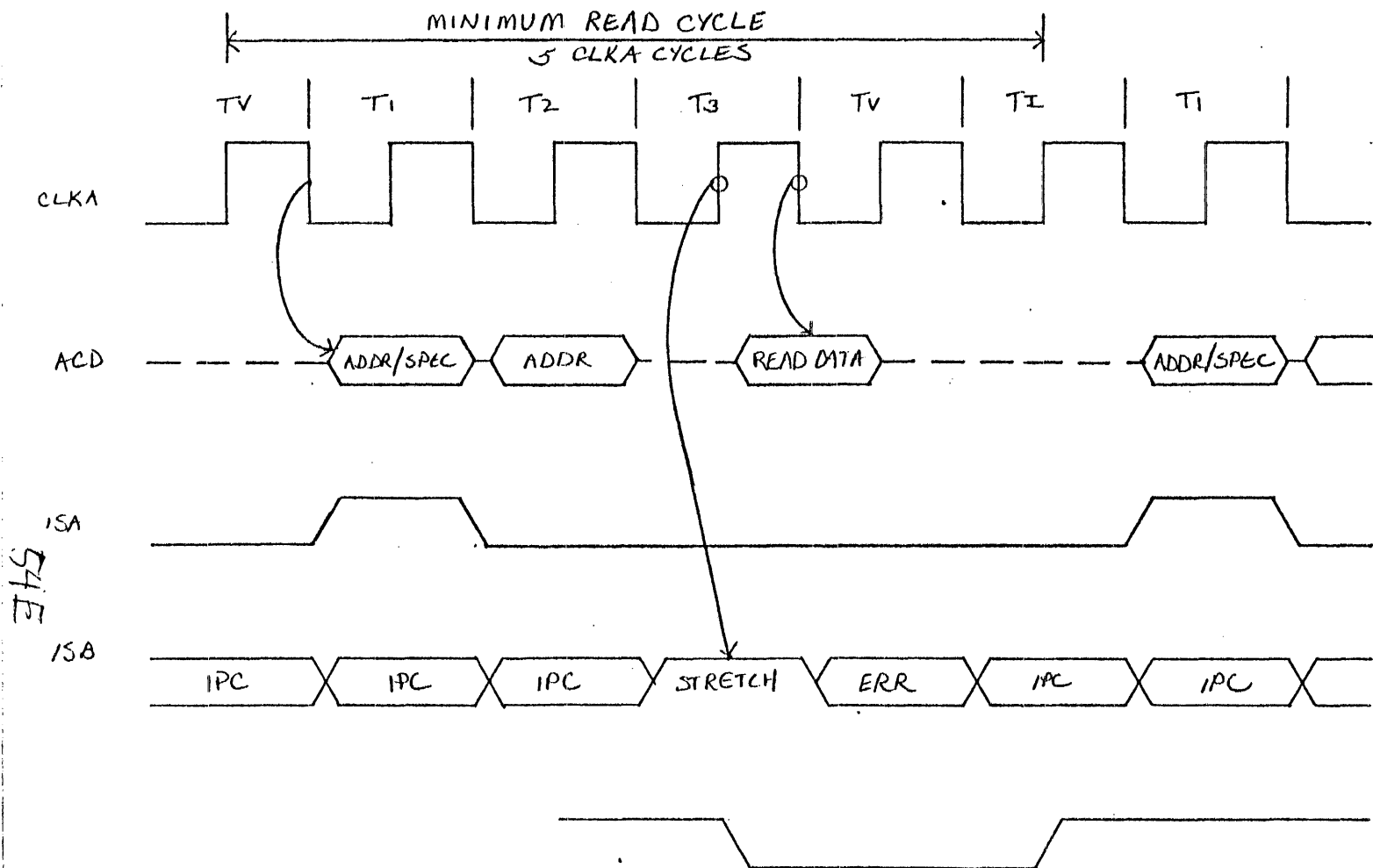
T<sub>3</sub>

T<sub>v</sub>

T<sub>i</sub>

\* Undefined if single byte read

FIGURE 4-7 MINIMUM READ CYCLE (BUFFERED SYSTEM)



NO EXTERNAL BUFFERS (BOUT NOT USED)

ACD15	ACD8	ACD7	ACD0
Hi-Z		Hi-Z	
Spec		Lo-Adr	
Hi-Adr		Mid-Adr	
Hi-data*		Lo-data1	
Hi-Z		Hi-Z	
Hi-Z		Hi-Z	
Spec		Lo-adr	

State

Tv (Preceded by a read): AD TIMING

T1

T2

T3

Tv

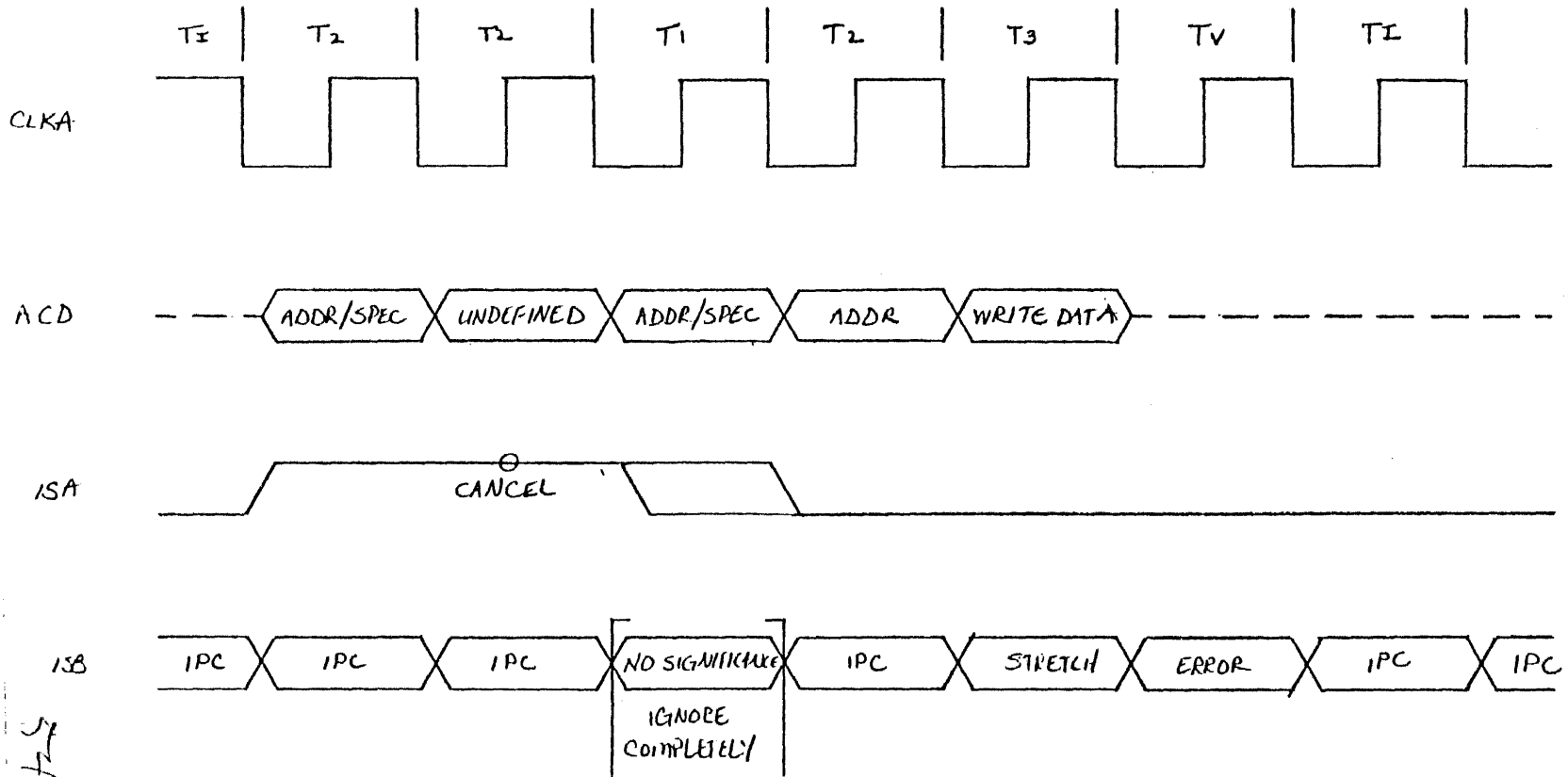
T1

T1

\* Undefined if single byte read

FIGURE 4-8 MINIMUM READ CYCLE (NOT BUFFERED)

(cl)



ACD15	ACD8	ACD7	ACD0
Hi-Z		Hi-Z	
Spec		Lo-adr	
Undefined		Undefined	
Spec		Lo-adr	
Hi-adr		Mid-adr	
Hi-data*		Lo-data	
Hi-Z		Hi-Z	
Hi-Z		Hi-Z	

State

- T<sub>1</sub>
- T<sub>1</sub>
- T<sub>2</sub> Access Cancelled
- T<sub>1</sub>
- T<sub>2</sub>
- T<sub>3</sub>
- T<sub>v</sub>
- T<sub>i</sub>

\* Undefined if single byte write

FIGURE 4-9 MINIMUM FAULTED ACCESS CYCLE

(Halle)

57F

In all transfers between a processor and a slave, the data to be driven are clocked three-quarters of a cycle before they are to be sampled. The BOUT timing is unique because BOUT is intended as a direction control for external buffers.

Detailed set-up and hold times depend on the processor implementation and can be found in the A.C. characteristics section.

Note that in all transfers between a processor and a slave, data is clocked three quarters of a cycle before it is to be sampled. This allows adequate time for the transfer and ensures sufficient hold time after sampling.

Table 4-2 ICS Interpretation

Table 4-3 PRQ Interpretation

Table 4-4 BOUT

Table 4-5 iAPX 432 Component Signaling Scheme

## CHAPTER 5

### AN iAPX 432 MULTIPROCESSOR SYSTEM IMPLEMENTATION

The prototype system described here is a simple but functional multiprocessor system that demonstrates the major characteristics of iAPX 432 system implementations.

The first section of this Chapter deals with the processor component interconnection and inspects the associated Interconnect Status (ICS) logic, the Processor Request (PRQ) logic, and the Interconnect Processor (IPC) logic of iAPX 432 systems.

The second section describes the memory system logic and includes a discussion of the system clock generator, the PCLK generator, and a static byte-memory system. This section also discusses the important concepts of memory alignment, multibyte sequencing, and peripheral subsystem connections.

#### SYSTEM DESCRIPTION

The iAPX 432 demonstration system contains one General Data

Processor (GDP, consisting of an Intel iAPX 43201/43202 device pair), one Interface Processor (43203 IP device), and a 2147-based static memory system. This system is capable of supporting a total of four attached processors. The IP connects via cabling to the Multibus interface of a peripheral subsystem. Figure 5-1 is a block diagram of the two-processor iAPX 432 system. Figure 5-2 shows the physical partitioning of the system.

Each processor provides demultiplexing and buffering logic for the processor packet bus, IPC logic, and bus request logic. The memory system contains the system clock generator, the bus arbitration unit, the memory array, the memory sequencer, and bus buffering logic.

Figure 5-1 Prototype System Block Diagram

Figure 5-2 Physical System Partitioning

#### PROCESSOR COMPONENT INTERCONNECTION

iAPX 432 processor components share some common requirements in the system described. Refer to Figure 5-11 and Figure 5-12 (Appended to this Chapter) for schematics of the two



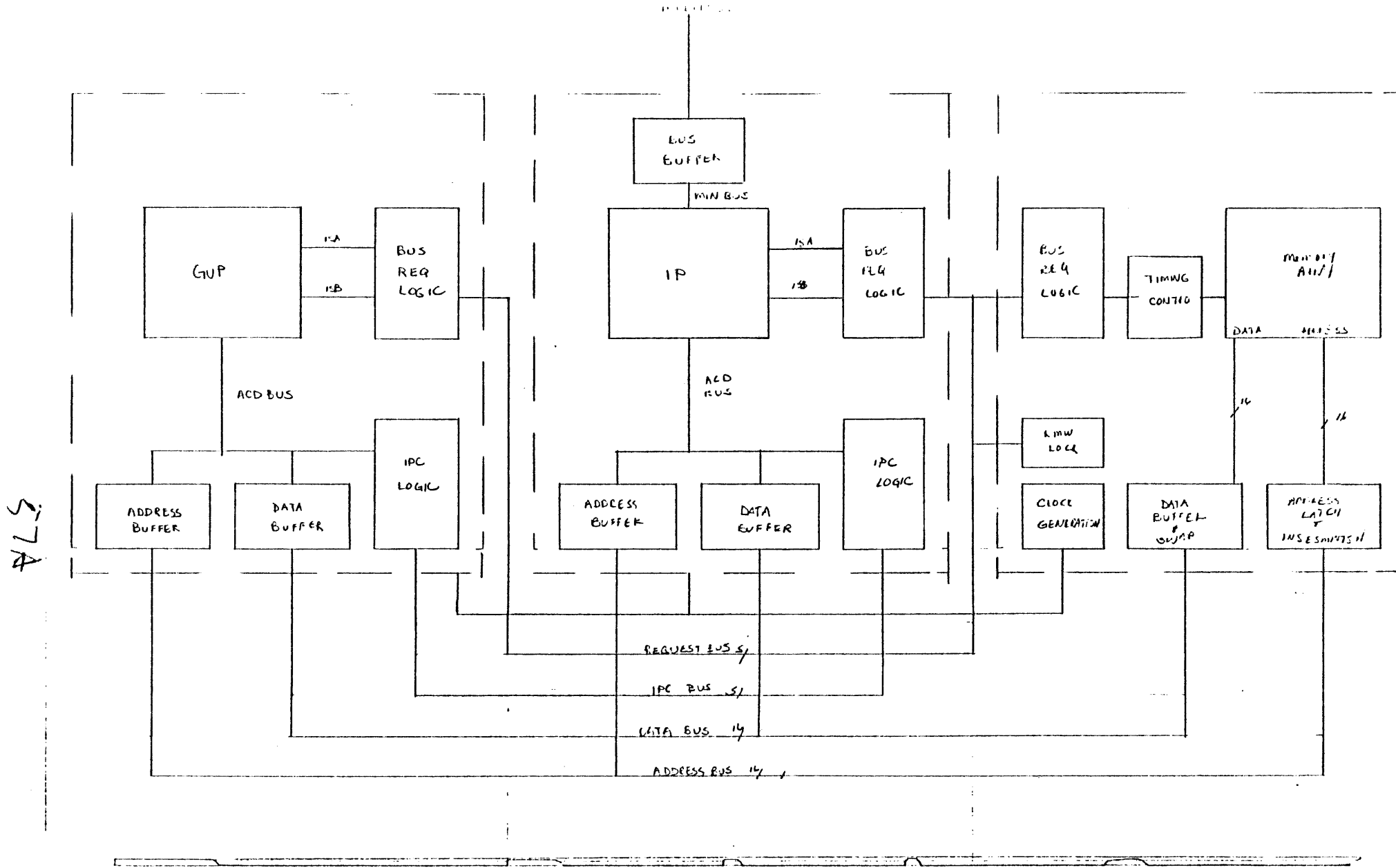


FIGURE 5-1 BLOCK DIAGRAM (TWO PROCESSOR 432 SYSTEM)

57B

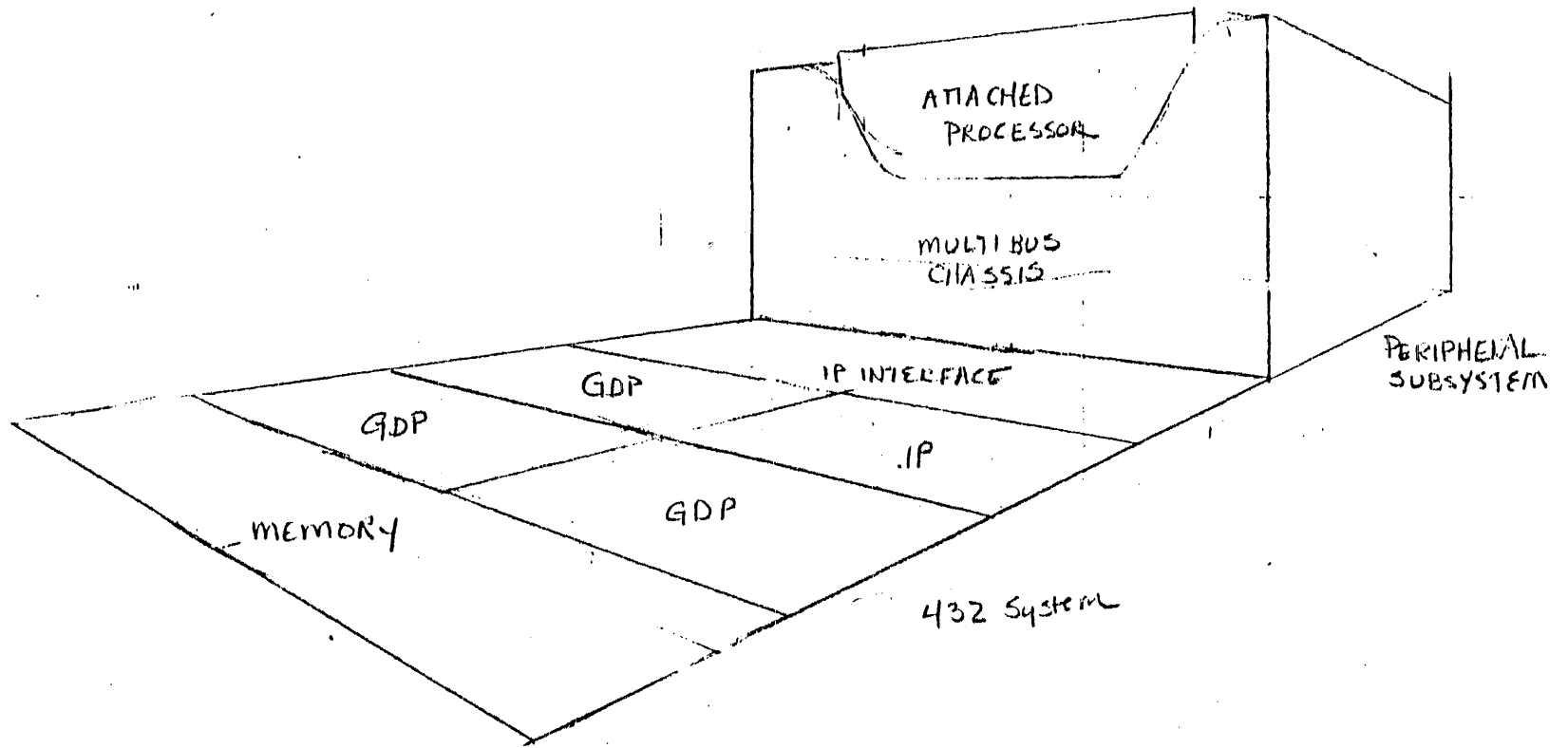


FIGURE 5-2 PHYSICAL PARTITIONING (TWO PROCESSOR 432 SYSTEM)

<u>Component</u>	<u>Pin</u>	<u>Connect to</u>
43201	MASTER CLR/ RDROM/ ALARM/	VCC VCC VCC VCC
43202	MASTER CLR/	VCC VCC
43203	HERR/ CLR/ ALARM/	GND VCC VCC

TABLE 5-1 MISCELLANEOUS PIN CONNECTIONS

57c

processor boards.

Power supplied to iAPX 432 components is distributed across multiple  $V_{CC}$

and GND pins. Each power pin must be connected to the appropriate supply. Each  $V_{CC}$  pin should be separately bypassed through a short, low-inductance path to ground.

Connections for CLKA and CLKB are made to all iAPX 432 components.

Each GDP or IP component requires interface logic to various circuits within the system. The following paragraphs will discuss the three specific areas of concern:

- o PRQ - ISC Logic
- o Bus Request/Grant Logic
- o Processor ID and IPC Logic

Figure 5-3 Address Specification Demultiplexing

Figure 5-4 Local Access Operation

PRQ and ICS Logic

58A

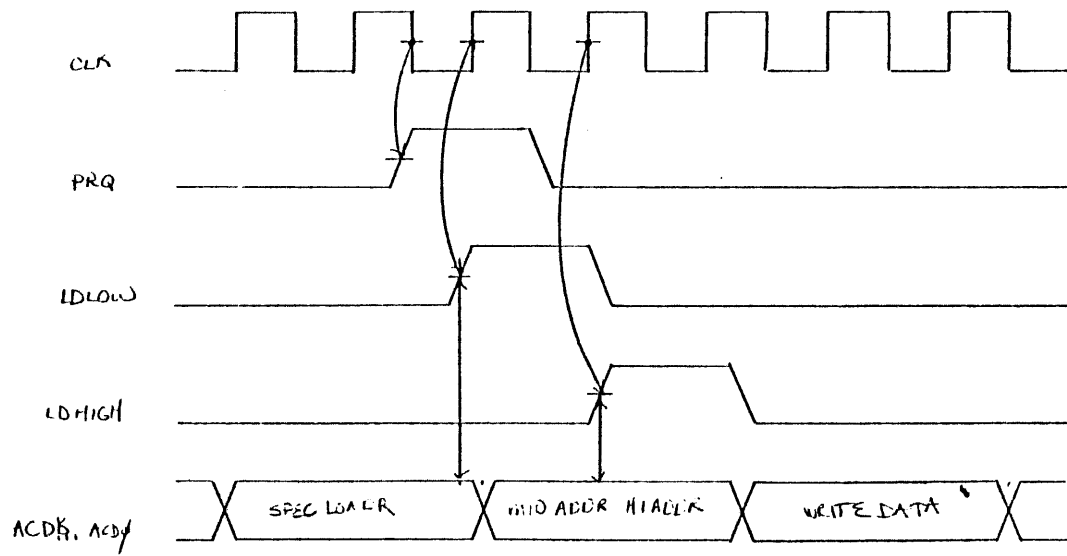
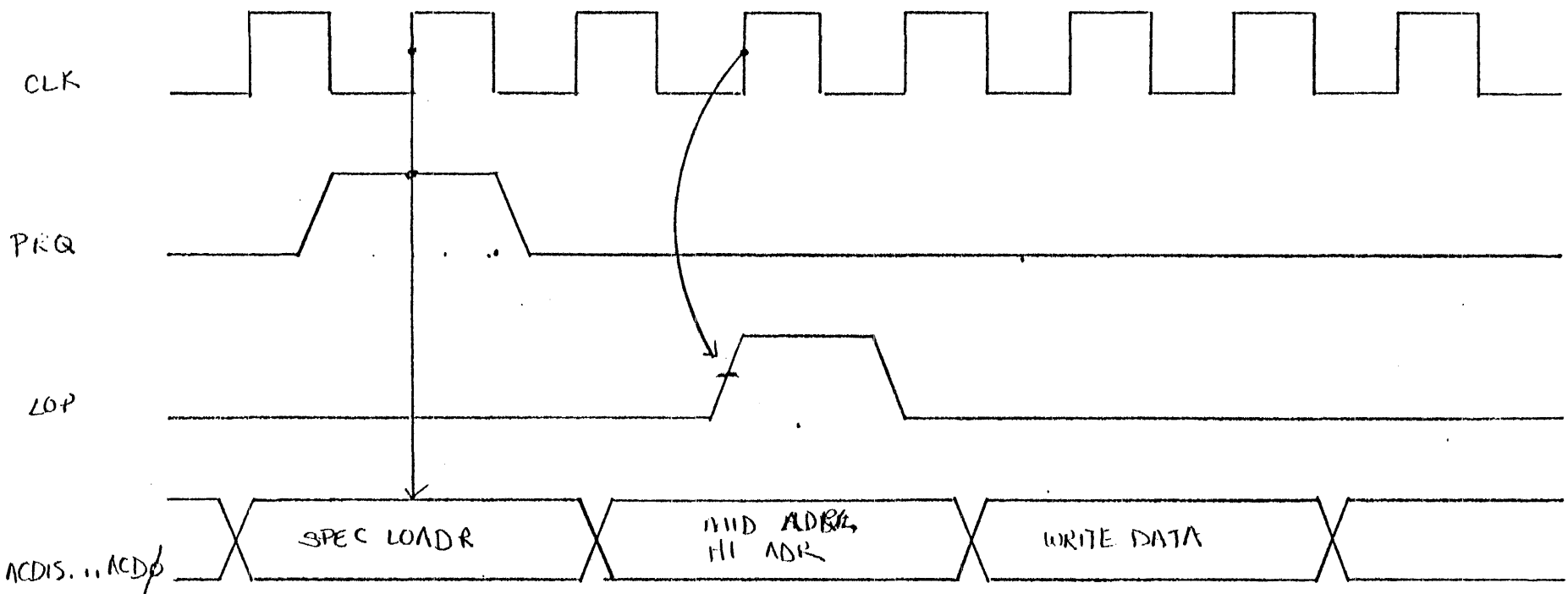


FIG 5-3 Address/Specification Demultiplexing

# FIGURE 5-3 ADDRESS/SPECIFICATION DEMULTIPLEXING

58B



NOTE: SPEC FIELD CONTAINS AN ACCESS TO LOCAL ADDRESS SPACE

FIGURE ~~5-1~~ - LOCAL ACCESS OPERATION

FIGURE 5-1 LOCAL ACCESS OPERATION

587

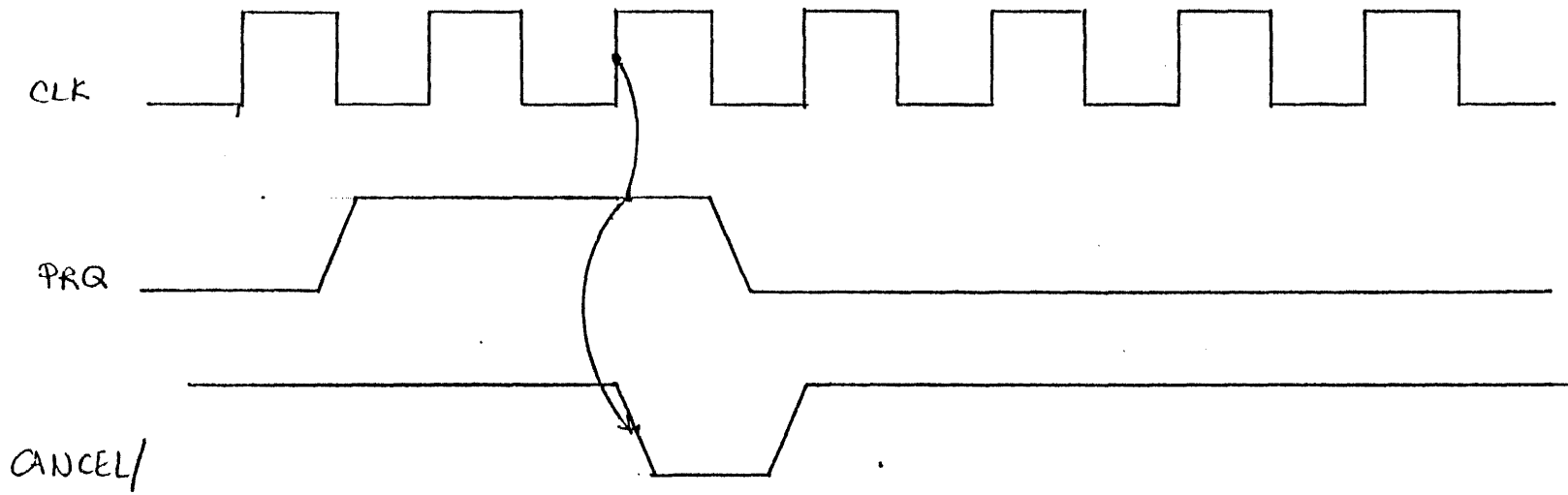


Fig. **5-5** - cancelled Access Detection

PRQ (Packet bus Request) and ICS (Interconnect Status) are the two signals that control data transfers on the processor packet bus. ICS is a processor output examined by a sequencer in managing the movement of information on the bus. PRQ is a processor input that may either signal an IPC, signal a bus error, or acknowledge the transfer of data.

#### PRQ-Related Logic

A TTL 74S175 register is employed as a state sequencer in decoding PRQ and generating several control signals. Figure 5-6 demonstrates the generation of LDLOW and LDHIGH strobes for demultiplexing the processor packet bus address and specification fields. The rising edge of LDLOW strobes the specification field and the low 8 bits of the 24-bit physical address into the ADDR<sub>0</sub> through ADDR<sub>7</sub> latch/bus drivers. The rising edge of LDHIGH strobes the mid-8 bits of the physical address into the ADDR<sub>8</sub> through ADDR<sub>F</sub> latch/bus drivers. The uppermost 8 bits of the physical address are discarded in this system. The state sequencer also decodes an access to the interconnect address space (ACD<sub>15</sub> = 1 when PRQ is asserted) and generates LOP (interconnect operation) as shown in Figure 5-5. Furthermore, cancelled accesses (denoted by two successive PRQ assertions) are detected by the sequencer, which generates the CANCL/ pulse. (See Figure



5-5.) Three consecutive assertions of PRQ cause an access to start, the CANCL/ signal to be asserted, and a new access to begin. (See Figure 5-6.)

Figure 5-5 Cancelled Access

Figure 5-6 Cancelled Access with Overlapped Access

### ICS-Related Logic

ICS (Interconnect Status) is a combination of several signals that indicate either the status of a transfer, the occurrence of access errors, or the signalling of IPC (Interprocessor Communication). The processor packet bus definition in the iAPX 432 component data sheets defines the time-dependent nature of ICS.

### Processor ID and IPC Logic

After INIT/ is pulsed low, each processor reads interconnect address space (address 0) to obtain an 8-bit processor ID number. Processor number 0 is not allowed as a processor ID

60A

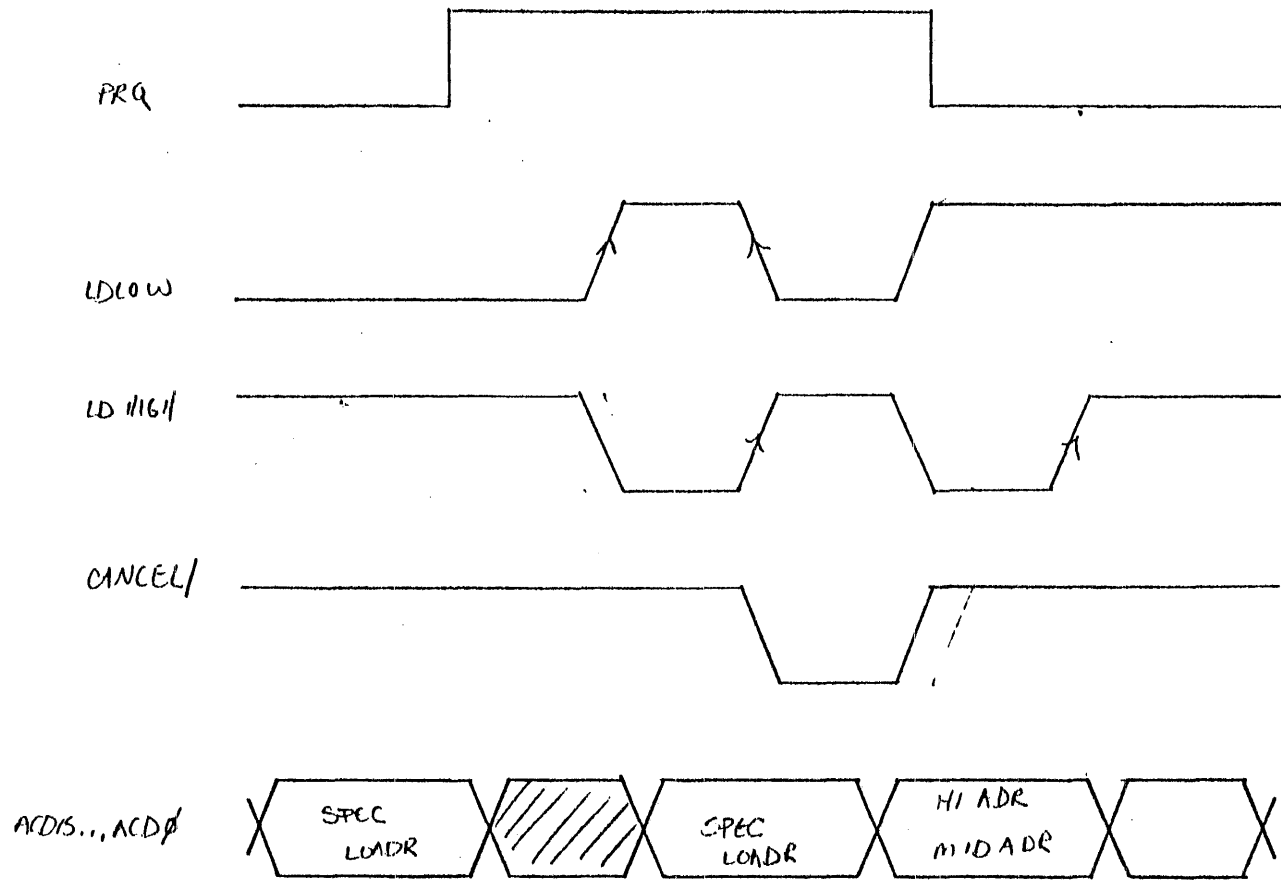


FIGURE 5-6 A cancelled Access w/ Subsequent access

code. The hardware at each processor location returns a unique 8-bit processor ID code.

IPC signals are transferred to a processor board by the GLOBAL/ and LOCAL/ signals being held active for one clock cycle. Each type of IPC is latched at the respective processor. Interconnect address space (address 2) may be read by a processor to disclose whether a local or global (or both) IPC has been signaled. Once read, the IPC status bits are cleared one at a time. See Table 5-2 for the IPC bit encodings.

A processor that signals an IPC will send either a global or a local IPC by writing to local address space register 2. Logic on each processor board decodes the specification field, interconnect address 2, and the processor ID field of the data packet double byte. A processor may signal any processor, all other processors, or itself.

Table 5-2 IPC Register Bit Designation

#### MEMORY SYSTEM LOGIC

The memory system (Figure 5-7) contains a static memory system, an access sequencer, a bus arbiter, a PCLK generat-

ACD	15	0	
	X X X X X X X X	0 0 0 0 0 0	G L Interconnect Address 2 ( <u>READ</u> )
	L = Local IPC waiting G = Global IPC waiting X = Undefined		
ACD	15	0	
	X X X X X X X X	0 0 0 0 0 0	C C C Interconnect Address 2 ( <u>WRITE</u> )
		0 0 0	Signal Global IPC
		0 0 1	Signal Local IPC to Processor 1
		0 1 0	Signal Local IPC to Processor 2
		0 1 1	Signal Local IPC to Processor 3
		1 0 0	Signal Local IPC to Processor 4

X = Undefined

TABLE 5-2 ~~XXXXXXXXXX~~ - IPC Register Bit Designation

or, and a clock generator for the system.

### **System Clock Generator**

CLKA and CLKB for iAPX 432 components are generated at four times the component operating frequency by a crystal oscillator module that drives a Johnson counter. CLKA and CLKB are symmetrical square waves in quadrature (90 degrees phase shift).

### **PCLK Generator**

PCLK, the timing signal for the system timer and process timer in iAPX 432 components, is generated by dividing CLKA by 1024.

Figure 5-7 Memory system schematic

### **Static Byte Memory System**

The static memory system demonstrates the requirements of an iAPX 432 system memory: alignment of data transfers, multibyte sequencing, and Bus Arbitration. Figure 5-7 a schematic of the memory system described.



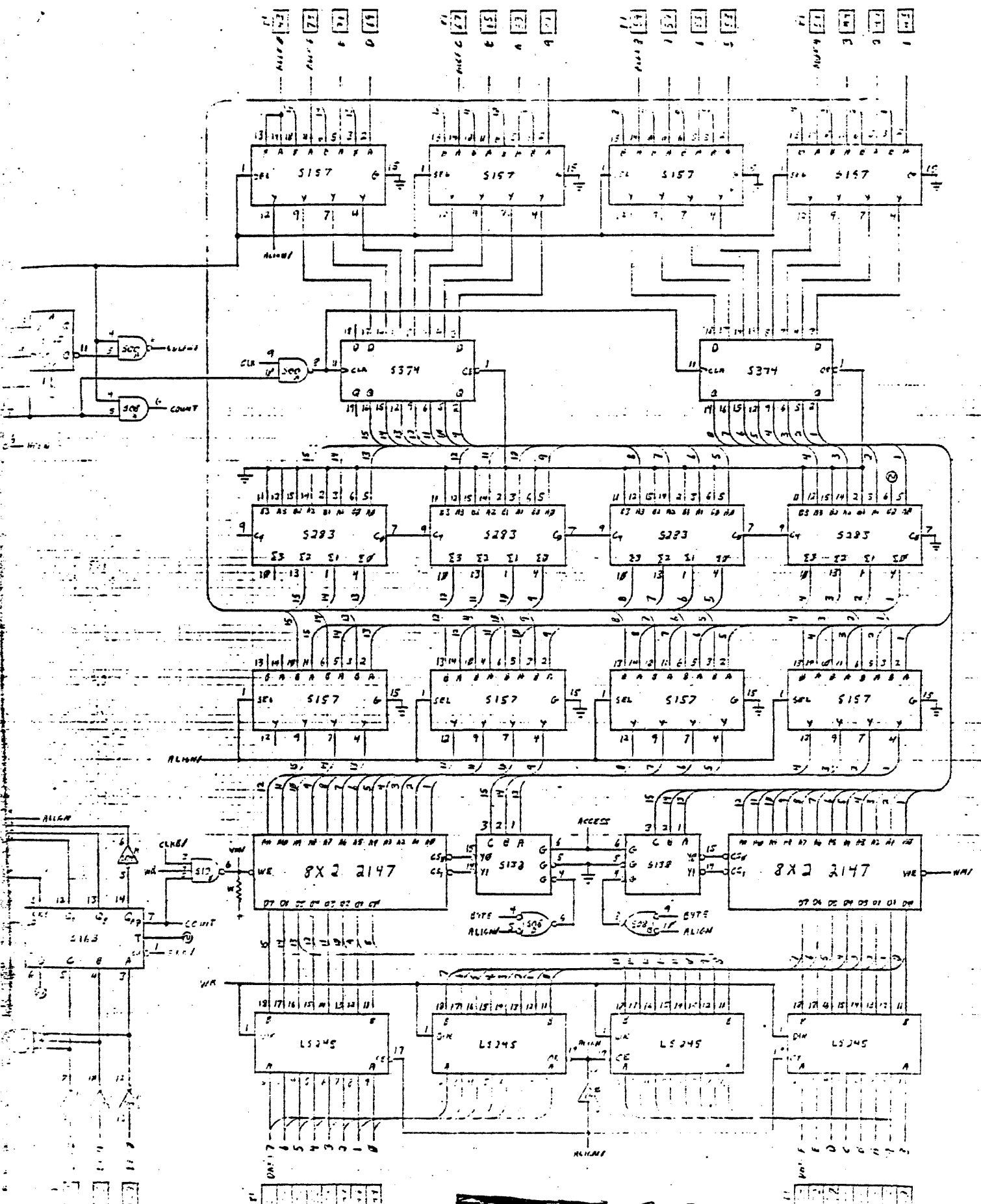


FIGURE 5-7 ~~STATIC MEMORY SYSTEM~~ 20F2  
62B

## Alignment

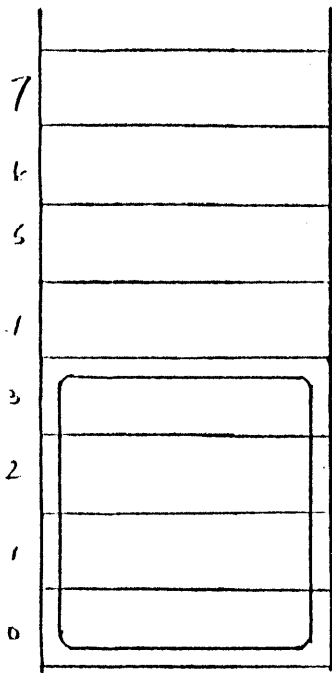
Figure 5-8 depicts the transfer of a word operand (four bytes). A processor may request this operand from memory without concern for its alignment in physical memory. The word operand consisting of bytes 0 to 3 is aligned to a double word boundary. The word operand consisting of bytes 3 to 6 is not aligned. The processor packet bus specification field for each of these accesses is the same. (Only the physical starting addresses differ).

Physical address bit 0 indicates whether an access is aligned on a double-byte boundary (address bit 0 = 0) or unaligned (address bit 0 = 1). When signal ALIGN/ is asserted, the memory system will perform the appropriate byte swapping to guarantee that right-justified data is transferred on the processor packet bus.

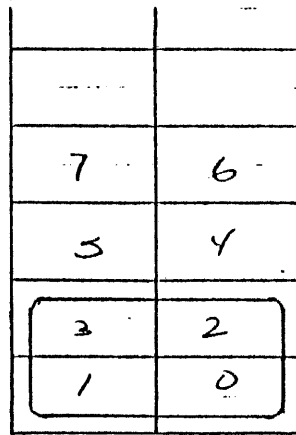
In Figure 5-8 the double word operand reference to location 3 demonstrates an unaligned reference to a word operand. In addition to the byte swapping requirement, references to two consecutive memory locations are required to transfer each double byte. Figure 5-9 shows how separately addressed parallel memory arrays are used to accomplish the transfer of any double-byte pair (aligned or unaligned).



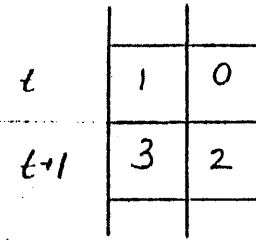
PROGRAMMER VIEW



MEMORY SYSTEM

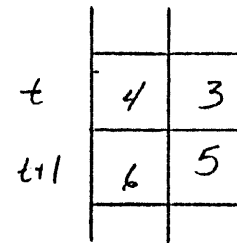
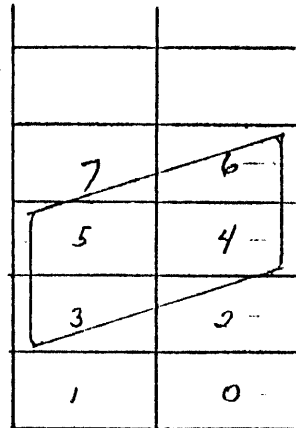
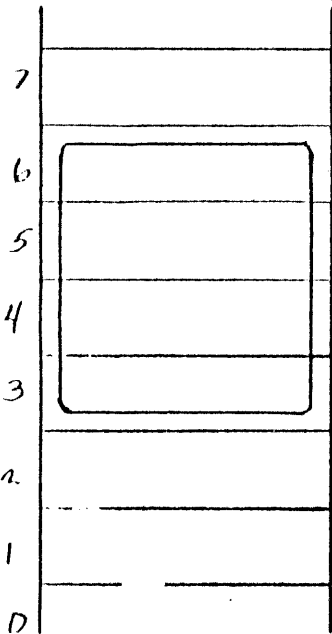


DATA TRANSFERRED ON PROCESSOR PACKET BUS



63A

FIGURE 5-8 BYTE SWAPPING FOR 32-BIT OPERAND



(all numbers in or out?)

Figure 5-8 Operand References with Alignments

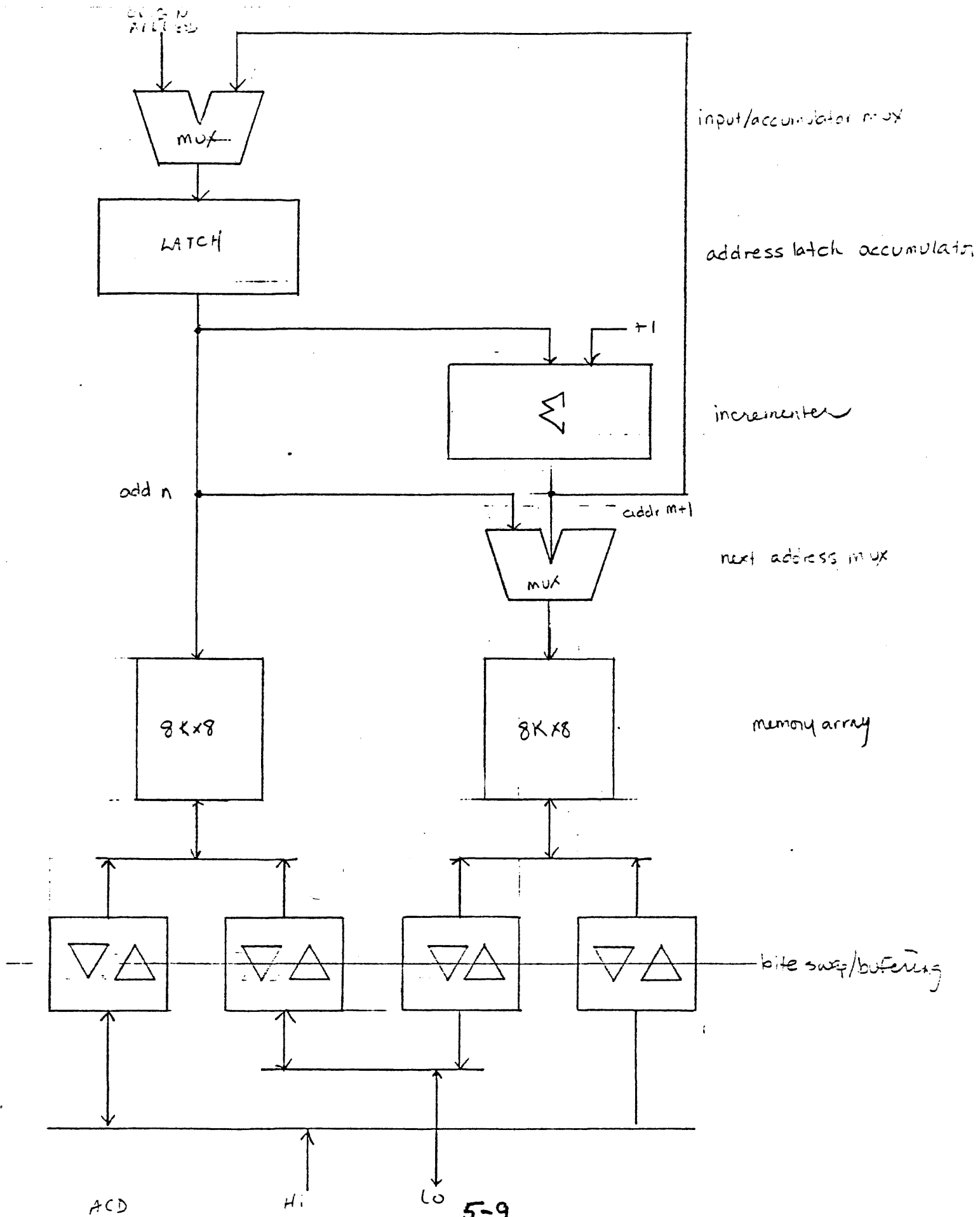
Figure 5-9 Parallel Memory Array Organization

### MultiByte Sequencing

The specification field of a processor packet bus transaction indicates the number of sequential bytes to be transferred at one time. The memory system uses the parallel bank memory structure just described, with appropriate control, to sequence multibyte accesses.

A TTL 74S163 counter is loaded with a recoded value for the length information in the specification field of a processor packet bus request. Accesses of encoded length 0 will cause the BYTE signal to be asserted. The 74S163 counts the number of double-byte accesses required to complete the transfer. At the end of a transfer the signal ERR is asserted, which indicates (via signal MERR) the ICS bus error significance (always errorless in this system).

The access sequencer generates one HOLD assertion for each unaligned double byte transferred. HOLD generates a wait state via the requesting processor's ICS pin. Aligned accesses and single byte accesses occur with no HOLD



5-9  
 FIGURE 5-9 - PARALLEL MEMORY BANKS

64A

penalty.

### Bus Arbiter

The bus arbiter section of the memory system arbitrates requests from among the four processors for use of the bus and system memory. The arbiter uses BREQ4/ as the highest priority and BREQ1/ as the lowest to resolve simultaneous requests. The resolved bus grant is returned to the highest-priority requester by asserting the appropriate BGR<sub>Tn</sub>/ line. BGR<sub>Tn</sub>/ remains asserted for the duration of processor "n's" access.

### Peripheral Subsystem Connection

The IP board (Figure 5-12) connects via two buffered flat cables to the Multibus slave interface (Figure 5-10) in the peripheral subsystem containing the attached processor. The buffers and synchronization logic to generate the SYNC signal to the IP are located on the IP end of the cable

The Multibus interface board employs an Intel 8218 Multibus Arbiter to provide the peripheral subsystem interlock function. Even though the Multibus interface is a slave during data transfer operations, the IP uses its HLD/HDA pins to lock the Multibus (a function of a Multibus master) at certain key times. The HLD/HDA interlock is used



to prevent any peripheral subsystem processor from generating a new transfer to or from the IP while it is in critical sections of its internal operations (e.g., altering windows into iAPX 432 memory).

Eight output port locations are used by the board. These ports are located at I/O addresses 0 through 7 and may be byte-addressed only. Table 5-3 outlines the assignment of functions to ports.

#### Table 5-3 Peripheral Subsystem Dedicated Ports

Chip select for the IP is decoded from  $ADR_{10}$  through  $ADR_{13}$  of the Multibus, allowing the IP to be memory-mapped within any 64K-byte portion of the Multibus.

<u>Multibus Port</u>	<u>Designated Function</u>
0	Deactivate INIT/
1	Activate INIT/
2	Pulse ALARM/
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

~~TABLE 5-3~~ - Peripheral Subsystem Dedicated Ports

TABLE 5-3

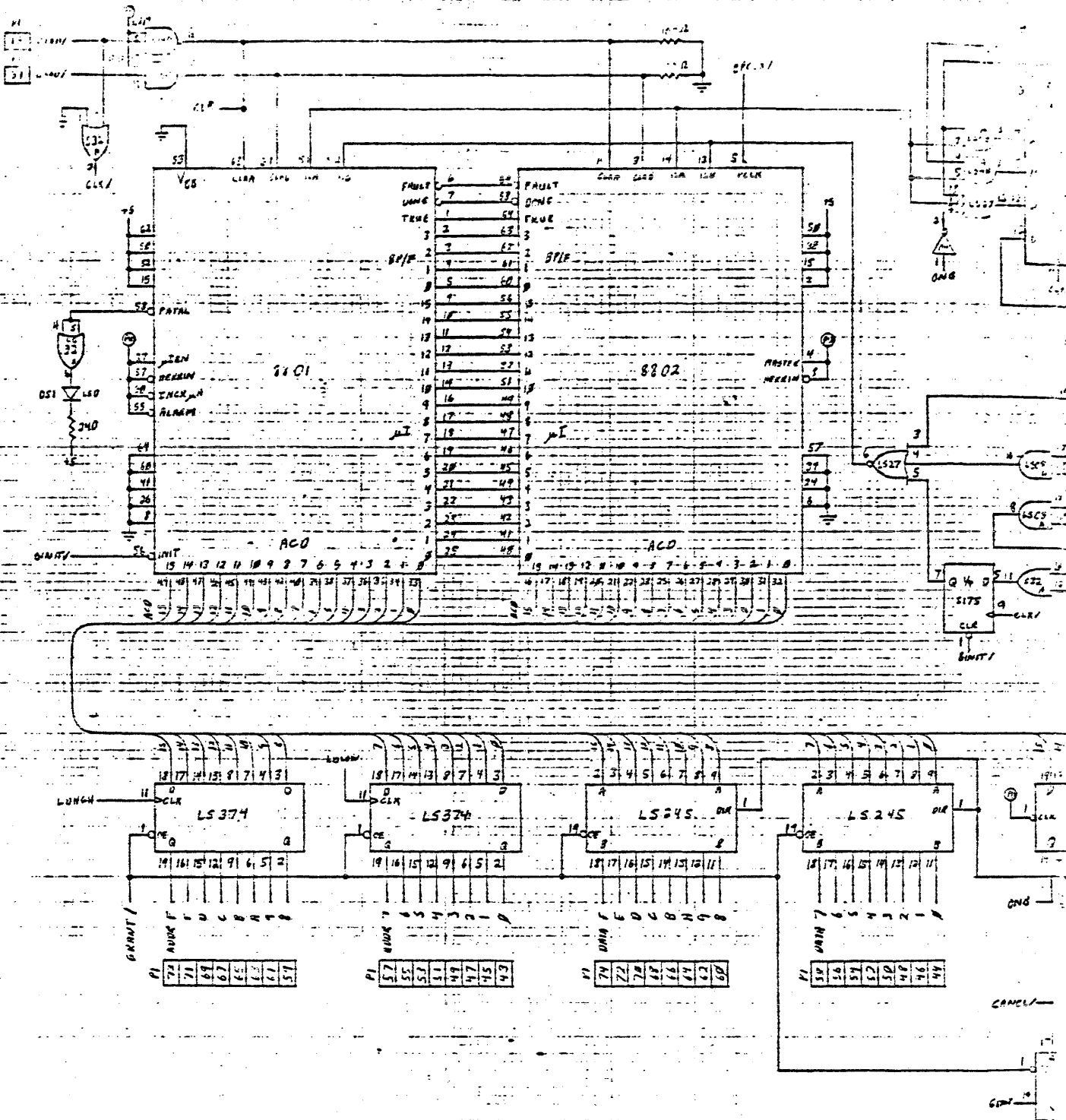
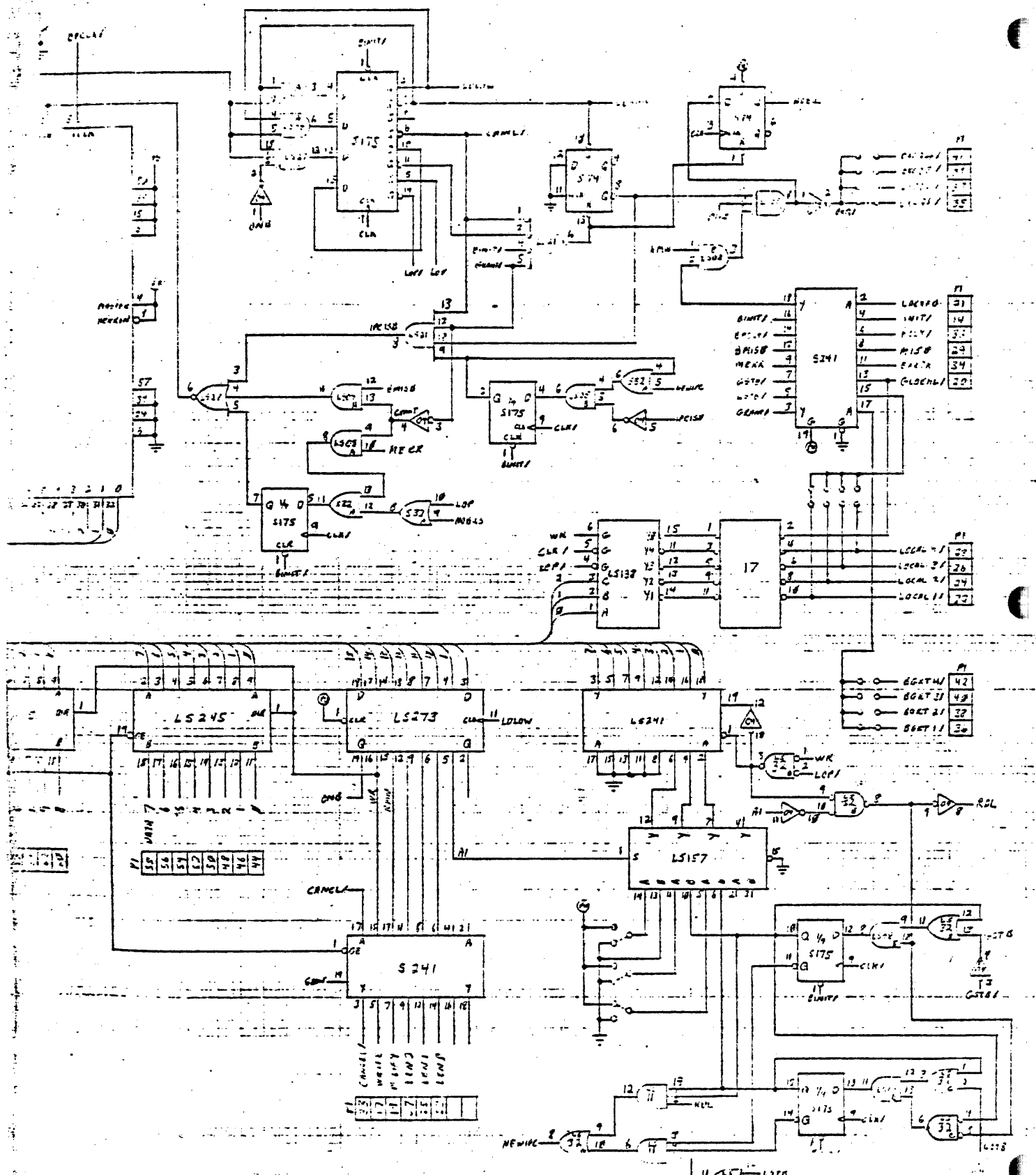


FIGURE 5-11 GDP SCHEMATICS  
~~FIGURE 5-11~~ 10/2

66B



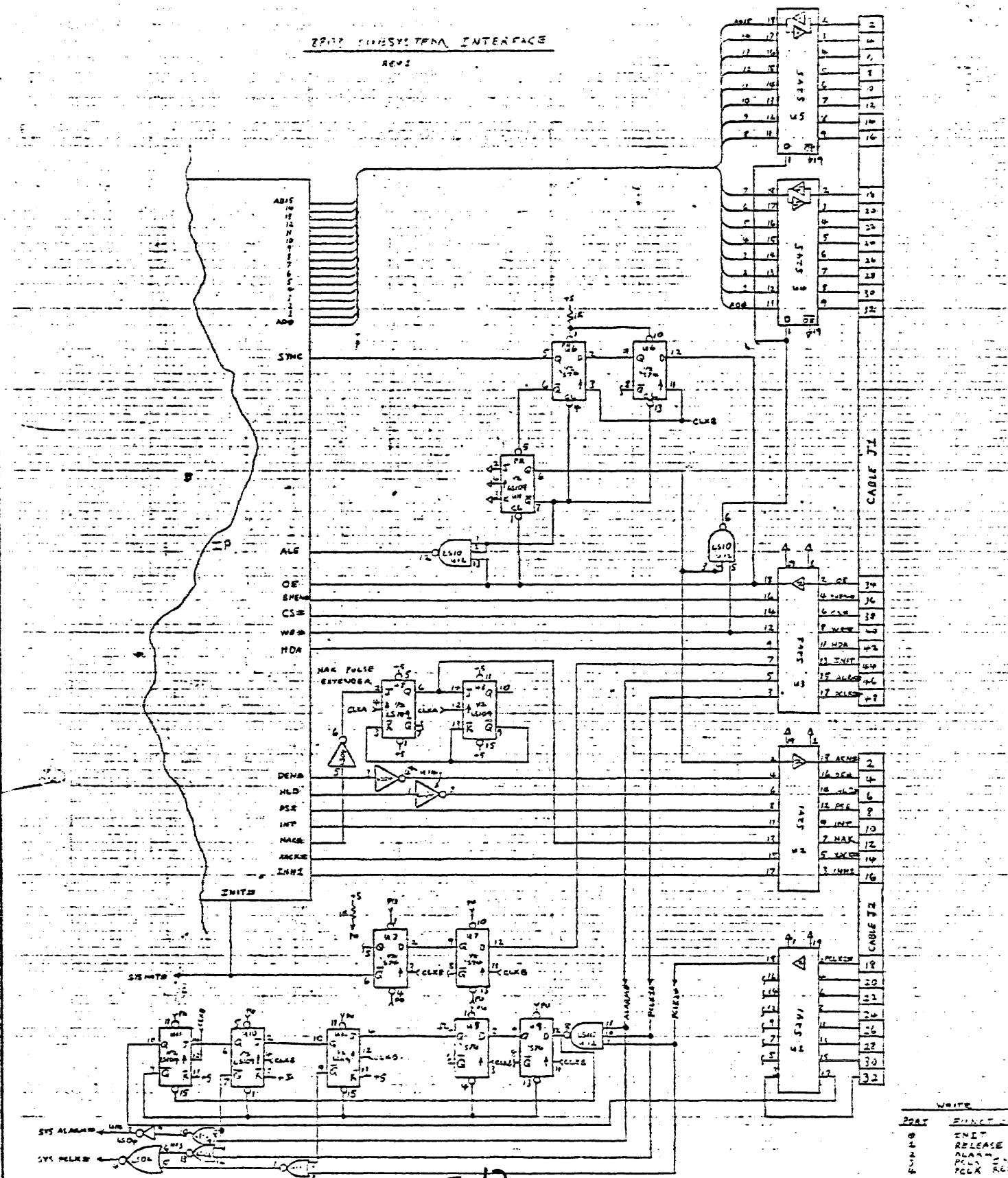


5-11  
 FIGURE 6PP 20F2

66C

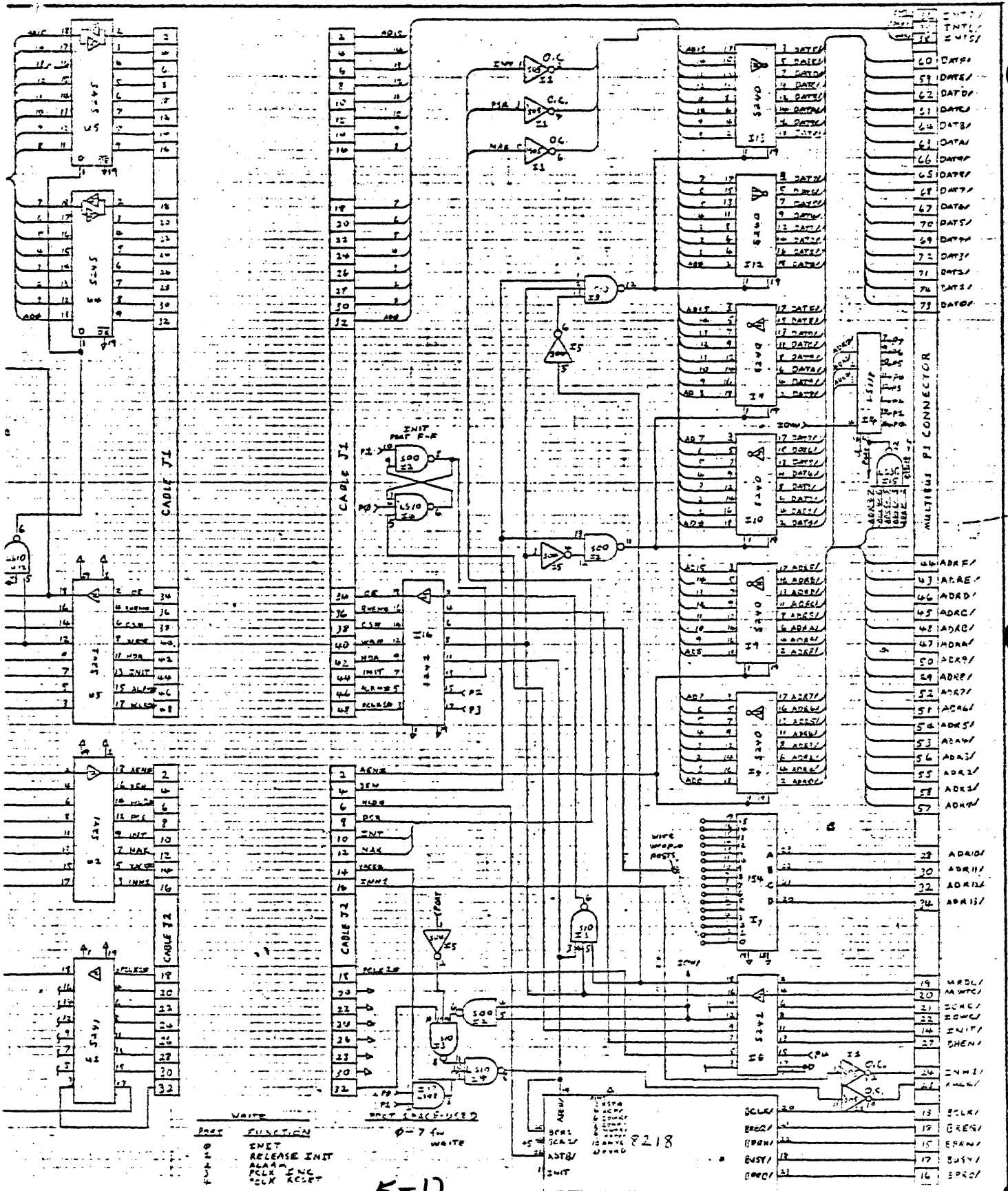
2717 SUBSYSTEM INTERFACE

REV 2



PORT	SIGNAL
1	INT
2	RELEASE
3	ACKR
4	ACKRE
5	INHZ
6	NAKB
7	PSE
8	HLD
9	DENB
10	HDA
11	WP
12	CS
13	SPEN
14	OE
15	ALE
16	AD0
17	NACK
18	ACK
19	INT
20	PSE
21	HLD
22	DENB
23	HDA
24	WP
25	CS
26	SPEN
27	OE
28	ALE
29	SYNC
30	AD0
31	AD1
32	AD2

REV.	DESCRIPTION OF REVISION	ENG. APPR.	DES. APPR.	DATE	REV.	REV. CONT.	ENR. APPR.	OPR. APPR.	DATE	NOTE
						FIGURE 10F2				



NOTES

FIGURE IP 20F2

		3545 S.W. 133TH. AVE. INTEL CORPORATION ALMOHA, OR. 97005	
DRAWN BY: [ ]		DEVICE NUMBER: [ ]	
CONTY BY: [ ]		DEVICE FUNCTION: [ ]	
ENCL APPR: [ ]		PROCESS: [ ] SMT: [ ] QF: [ ]	

iAPX 43201

iAPX 43202

VLSI GENERAL DATA PROCESSOR

- o Self-dispatching processors for software-transparent multiprocessing.
- o Capability-based addressing and protection.
- o 2 to the 40th bytes of virtual address space.
- o Functional redundancy checking mode for hardware error detection.
- o Hardware implemented inter-process communication and dynamic storage allocation
- o High-level language directed instruction set with 0-3 operand references.
- o Symmetrical support of all 8-, 10-, 32-bit scalar data types and proposed IEEE standard 32-, 64-, and 80-bit floating point.
- o Object-based architecture for improved programmer productivity.

The Intel iAPX 432 General Data Processor (GDP) consists of two VLSI devices, the 43201 and the 43202. These companion devices (Shown in Figures 1. and 2.) provide the general data processing facility of the iAPX 432 Micromainframe. The combination of VLSI technology and advanced architecture in the iAPX 432 results in mainframe functionality with a microcomputer form factor. The new object-based architecture significantly reduces the cost of large software systems and enhances their reliability and security.

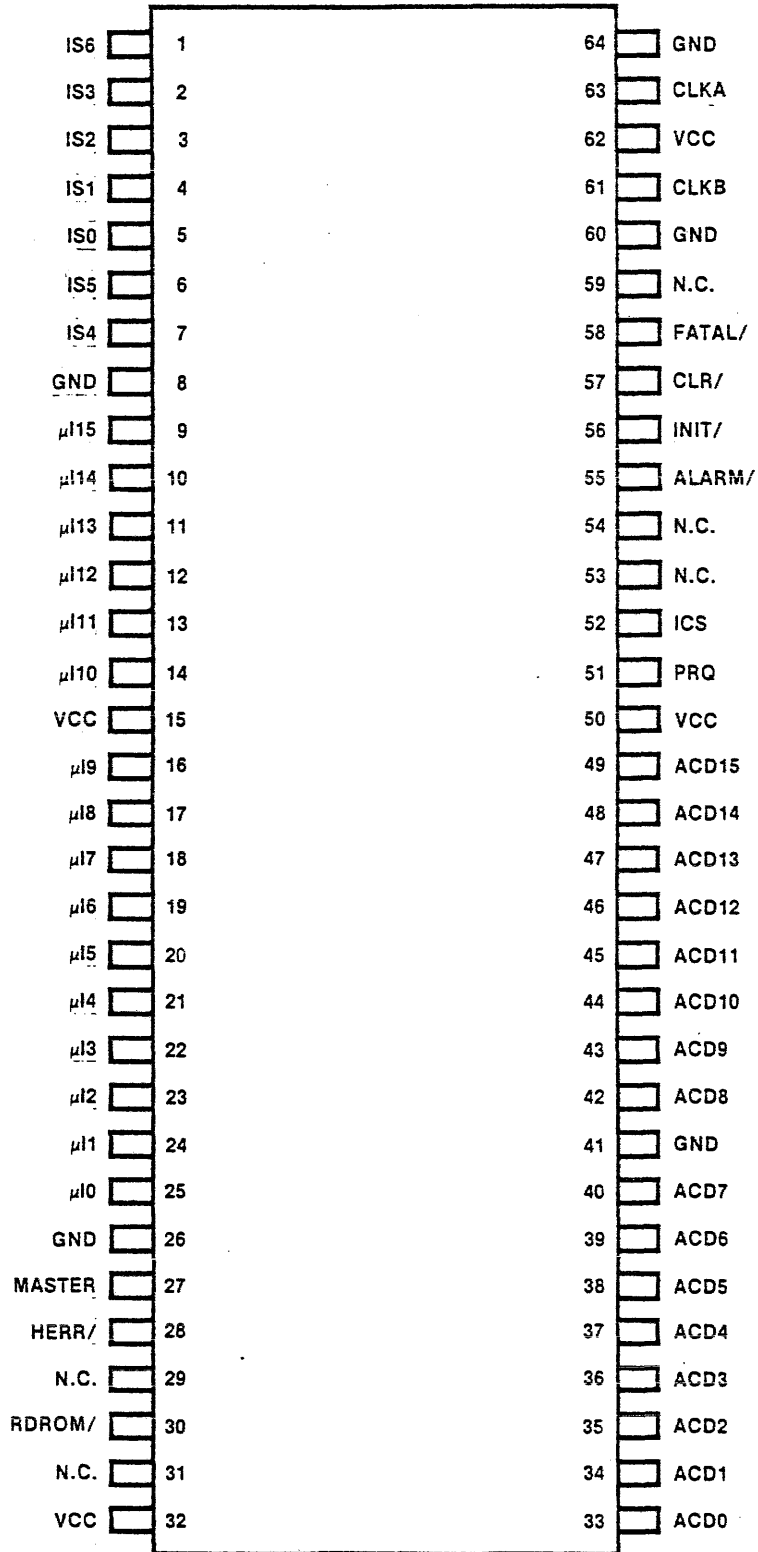
Software-transparent multiprocessing allows the user to configure systems matched to the required performance and provides an easy growth path. Hardware support for operating systems and high level languages ease their implementation.

The GDP provides  $2^{40}$  bytes of virtual address space and supports 8-, 16-, and 32-bit machines with capability-based addressing and protection. In addition, a hardware-implemented functional redundancy checking mode is provided for the detection of hardware errors.

The iAPX 43201 and iAPX 43202 are fabricated with Intel's highly reliable +5-Volt, depletion load, N-channel, silicon gate HMOS technology and is packaged in a 64-pin Quad In-Line Package (QUIP).

489

**FIGURE 1.** 432101 PIN ASSIGNMENT  
INSTRUCTION DECODER/MICROINSTRUCTION SEQUENCER



689

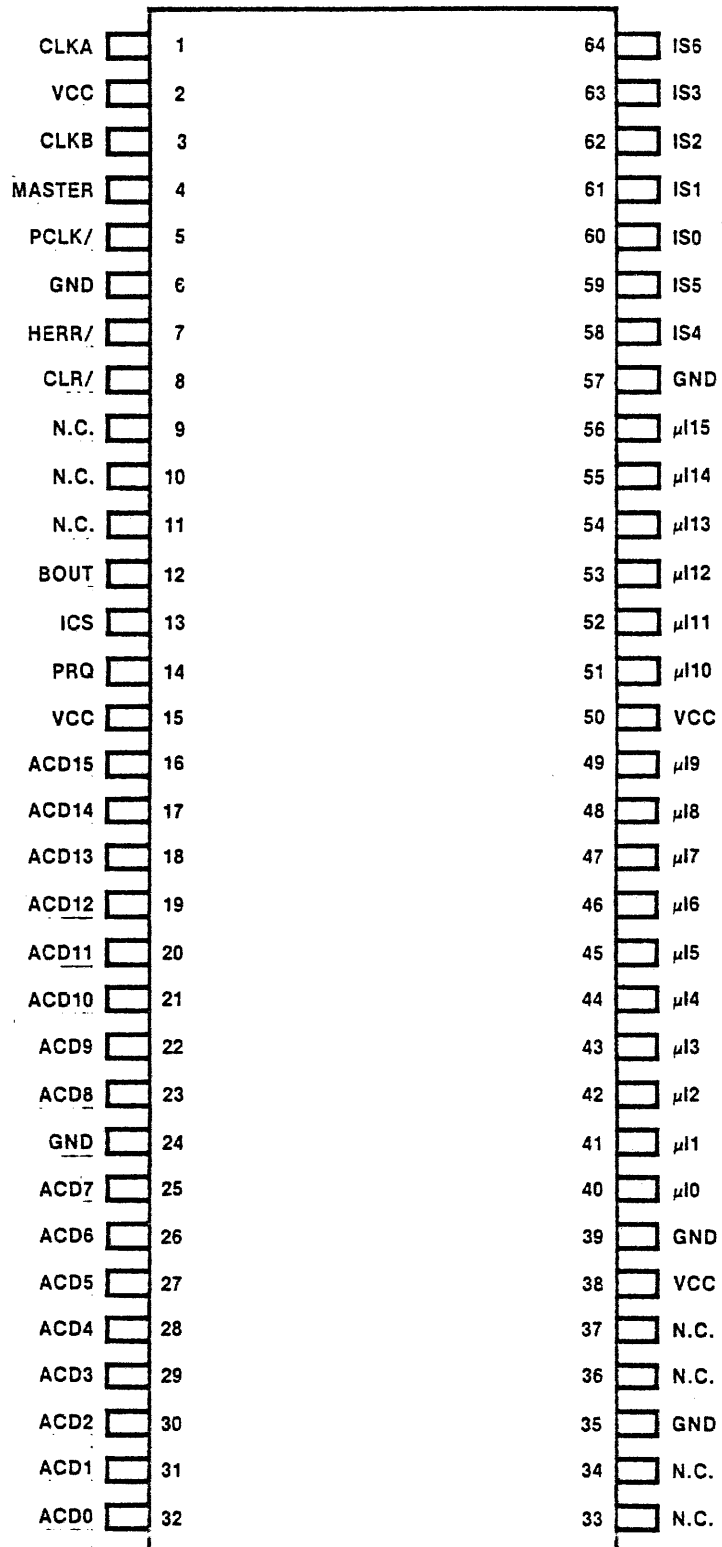


FIGURE 2.432/02 PIN ASSIGNMENT EXECUTION UNIT

Figure 1. Instruction Decoder/Microinstruction Sequencer

Figure 2. Execution Unit 43202 Pin Assignment

Figure 3. 43201 Block Diagram

Figure 4. 43202 Block Diagram

Figure 5. Hardware Error Detection



## iAPX 432 GDP Functional Description

The generalized data processor is organized internally as a three-stage microprogram-controlled pipeline. The first stage is the instruction decoder (ID); the second stage is the microinstruction sequencer (MS); and the third stage is the execution unit (EU).

The first two stages of the pipeline are physically located on the 43201 (Figure 3.). Each stage of the pipeline can be considered an independent subprocessor which operates until the pipeline is full and then halts and waits for more work to do.

### Instruction Decoder

The first subprocessor of the pipeline is the ID, which performs the following functions:

1. Receives macroinstructions
2. Processes variable-length fields
3. Extracts logical addresses
4. Generates starting addresses for the microinstruction procedures

70A

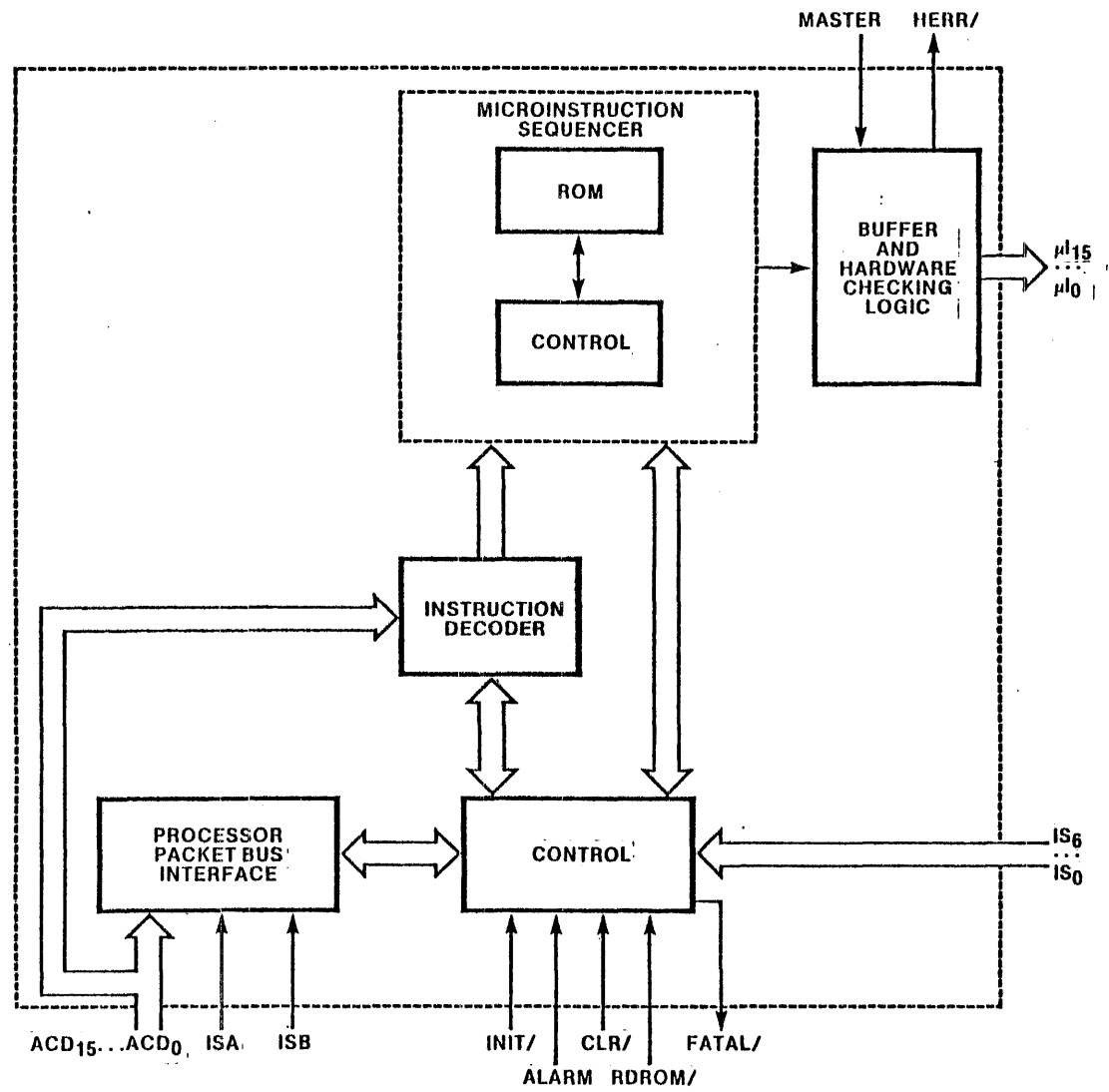


FIGURE 3. 43201 BLOCK DIAGRAM

5. Generates microinstructions for simple operations

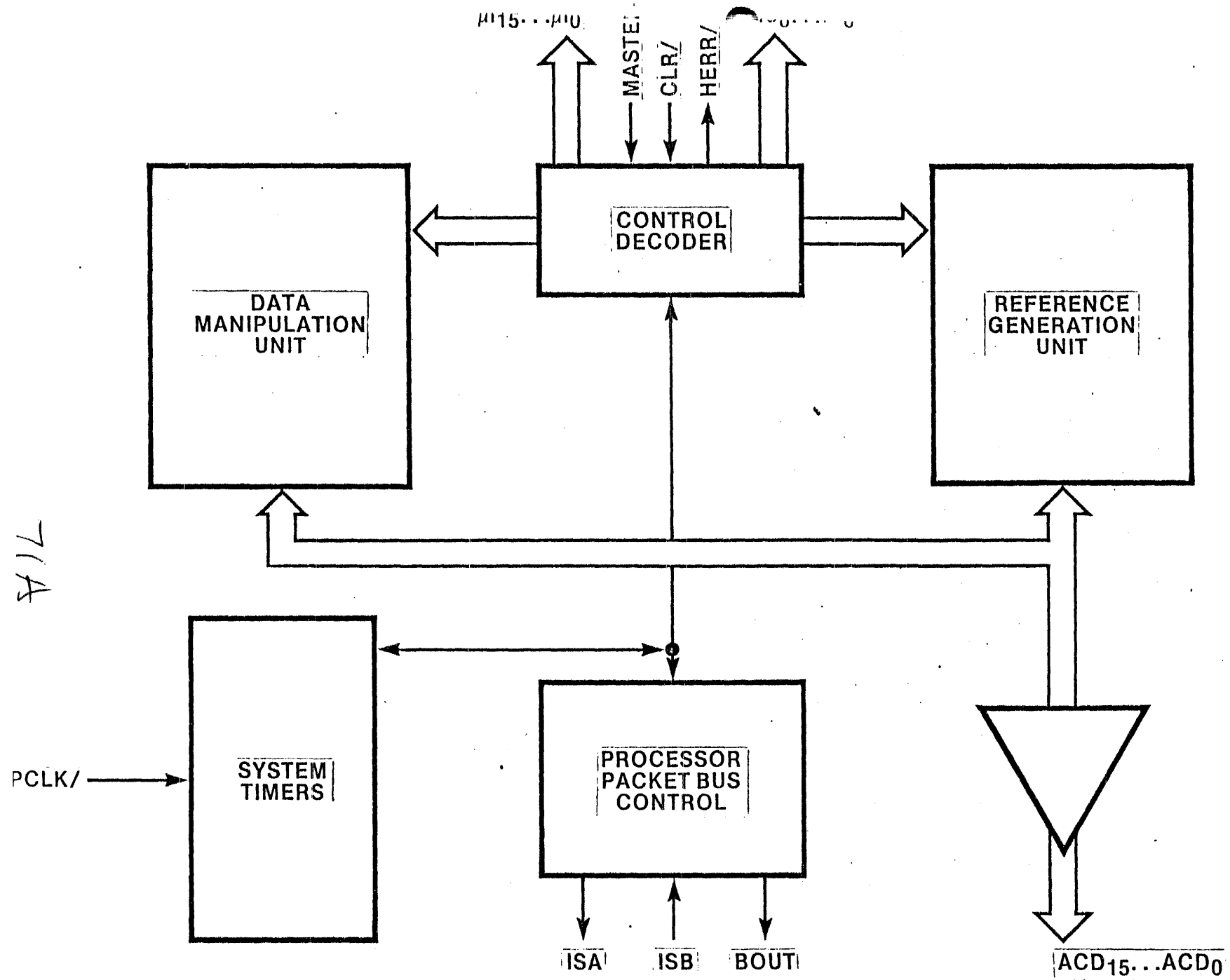
### Microinstruction Sequencer

The second subprocessor in the pipeline is the MS, which performs the following functions:

1. Issues microinstructions to the EU (43202)
2. Executes microcode sequences out of an on-chip 3.5K x 16-bit microcode ROM
3. Responds to the bus control signals
4. Invokes macroinstruction fetches
5. Initiates interprocessor communication and fault handling sequences

### Execution Unit

The 43202 contains the third stage of the GDP pipeline--the EU. (Refer to Figure 4.) This unit receives microinstructions from the 43201 and routes them to one of the two independent subprocessors that make up the EU. These two are the data manipulation unit (DMU) and the reference generation unit (RGU).



71A

43202

FIGURE 9a BLOCK DIAGRAM

71B

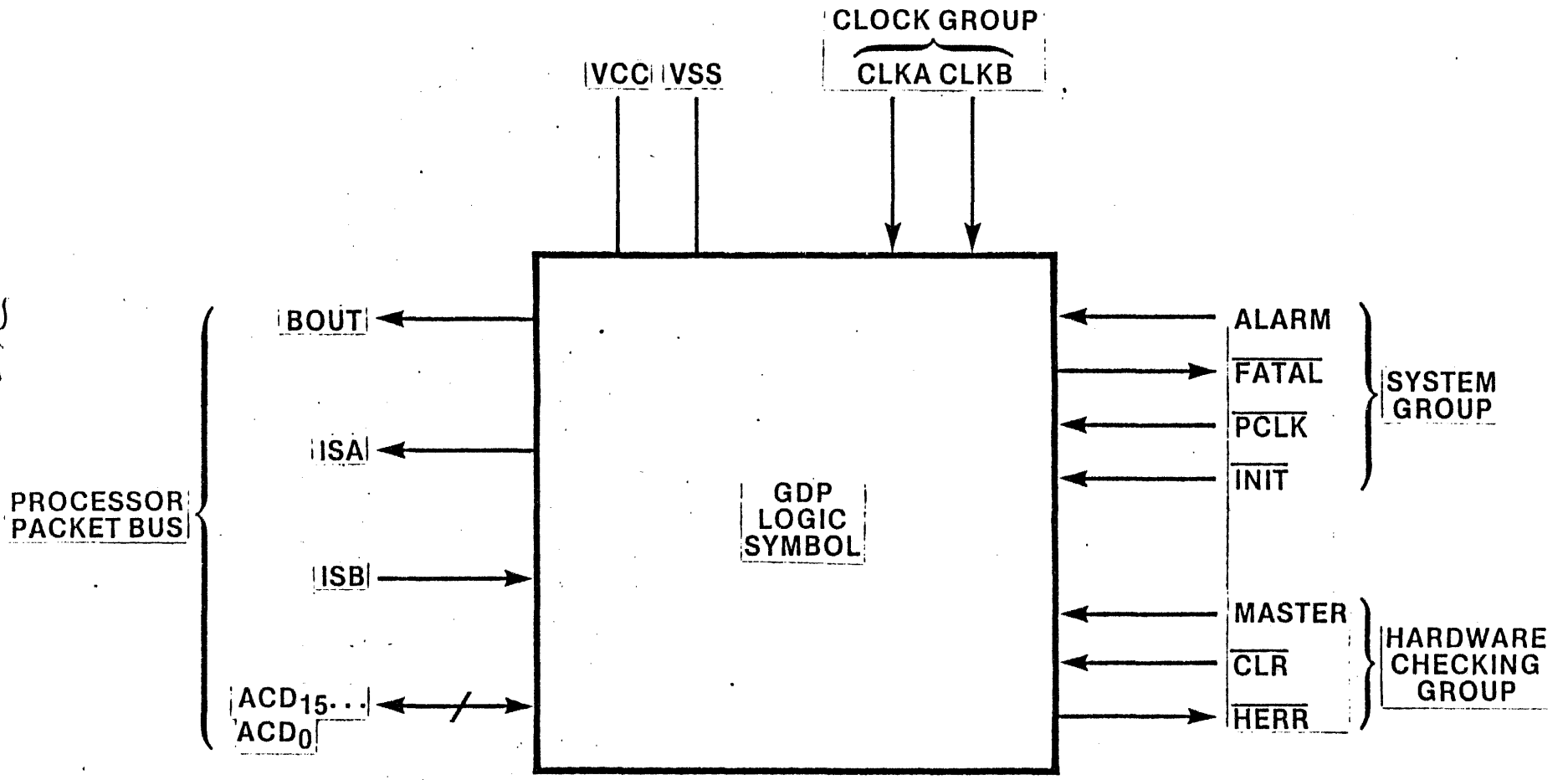


FIGURE 4b. GDP LOGIC SYMBOL

The EU executes most microinstructions in one clock cycle. However, each of the subprocessors has an associated sequencer that may run for many cycles in response to certain microinstructions. Those sequencers are invoked for complicated arithmetic operations (in the DMU) and processor packet bus transactions (in the RGU).

The DMU contains the registers and arithmetic capabilities to perform the following functions:

1. Hardware recognition of nine (9) data types
2. Built-in state machine for 16-, 32-, 64-bit multiply, divide and modulus
3. Control functions for 32-, 64-, and 80-bit floating point arithmetic.

The RGU performs the following functions:

1. Provides the translation of a 40-bit virtual address into a 24-bit physical address
2. Provides for a hardware-enforced domain protection system (read, write, alter, accessed)
3. Handles sequencing for 8-, 16-, 32-, 64-, and 80-bit

memory accesses

#### 4. Controls on-chip top-of-stack registers

The 43201 and 43202 components, described above, together form the GDP. Figure 4b is a diagram that shows both units interfacing to the Packet bus as a single processor.

#### Hardware Error Detection On iAPX 432 Processors

iAPX 432 processors include a facility to support the hardware detection of functional errors. At INIT time, each iAPX 432 processor is configured to operate as either a master or a checker processor. A master operates in the normal manner. A checker places all output pins that are being checked in the high-impedance state. Thus, a master and checker are parallel-connected, pin for pin, so the checker can compare its master's output values with its own. Any comparison error causes the checker to assert HERR, and go idle (Refer to Figure 5.). No further activity can then occur at the disagreeing master-checker GDP pair.

#### iAPX 43201/43202 Physical Interconnect

Figure 6 illustrates the iAPX 432 microcomputer form factor using a Dual QUIP. This layout promotes the most efficient

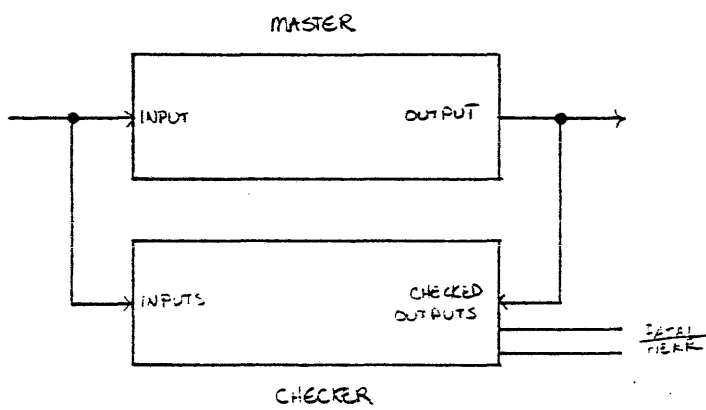


FIGURE 5. HARDWARE ERROR DETECTION 



utilization of the GDP. Also shown are the accessible GDP output/input signals.

Figure 6. QUIP Layout

### 432 Instructions

Intel iAPX 432 instruction codes have been designed to minimize the space the instructions occupy in memory and still allow for efficient encoding. In order to achieve the ultimate in efficiency of storage the instructions are encoded without regard for byte, word or other artificial boundaries. The instructions may be viewed as a linear sequence of bits in memory, with each instruction occupying exactly the number of bits required for its complete specification.

iAPX 432 processors view these instructions as composed of fields of varying numbers of bits that are organized to present information to the instruction decoder in the sequence required for decoding. A unified form for all instructions allows instruction decoding of all instructions to proceed in the same fashion.

In general, GDP instructions consist of four main fields. These fields are called the class field, the format field, the reference field, and the opcode field. The reference

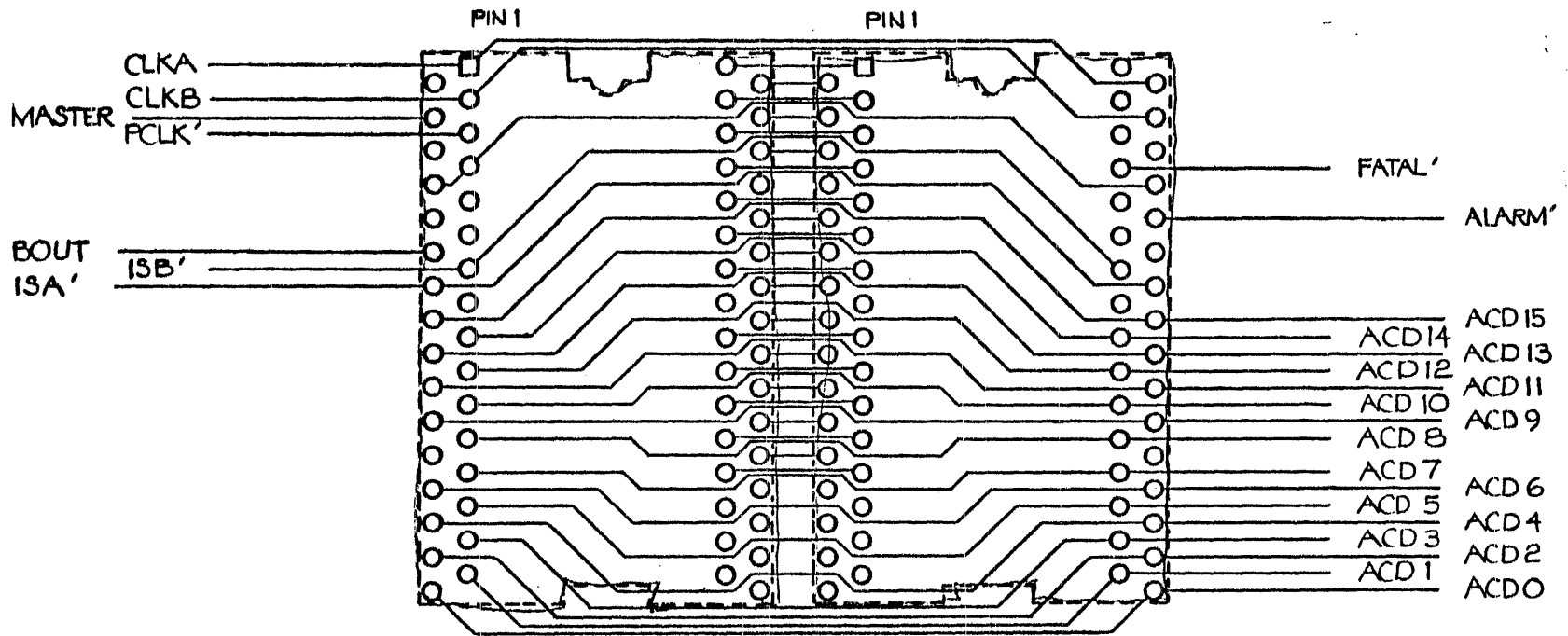


FIGURE 6. QUIP LAYOUT

field, in turn, may contain several other fields, depending upon the number and complexity of the operand references in the instruction. The fields of a GDP instruction are stored in memory in the following format:

The class field is either 4 or 6 bits long, depending on its encoding. The class field specifies the number of operands required by the instruction and the primitive types of the operands. The class field may indicate 0, 1, 2 or 3 references.

If the class field indicates one or more references, a format field is required to specify whether the references are implicit or explicit and their uses.

In the case of explicit references the format field can indicate whether or not the reference is direct or indirect. Further, the format field may indicate that a single operand plays more than one role in the execution of the instruction. As an example, consider an instruction to increment the value of an integer in memory. This instruction contains a class field, which specifies that the operator is of order two and that the two operands both occupy a word of storage, followed by a format field, whose value indicates that a single reference specifies a logical address to be used both for fetching the source operand and for storing the result, followed by an explicit data reference to the integer to be incremented, and finally

followed by an opcode field for the order-two operator INCREMENT INTEGER. It is possible for a format field to indicate that an instruction contains fewer explicit data references than are indicated by the instruction's class field. In such a case the other required data references are implicit references, and the corresponding source or result operands are obtained from or returned to the top of the operand stack. The use of implicit references is illustrated in the following example, which considers the high-level language statement

$$A = A + B * C$$

The instruction stream fragment for this statement consists of two instructions and has the following form:

opcode	reference	format	class
--------	-----------	--------	-------

<-----< Increasing address

Assume that A, B, and C are integer operands. The first class field (the rightmost field in the picture above) specifies that the operator requires three references and that all three references are to word operands.

The first format field contains a code specifying two explicit data references. These references are to supply only the two source operands. The destination is referenced implicitly so that the result of the multiplication is to be pushed onto the operand stack. The second class field is identical to the first and specifies three required references by the operator. In addition, all three references are to word operands. The second format field specifies one explicit data reference to be used for both the first source operand and the destination. The second source operand is referenced implicitly and is to be popped from the operand stack when the instruction is executed.

The reference fields themselves can be of various lengths and can appear in various numbers, consistent with their specification in the class and format fields. If implicit references are specified, reference fields for them will not appear. Direct references will require more bits to specify than indirect references.

Following the class, format, and reference fields, the opcode field appears. The opcode field specifies the operator to be applied to the operands specified in the preceding fields.

## Modes of Generation

Figures 7 and 8 illustrate the two iAPX 432 system modes of generation, selector generation and displacement generation.

The modes of selector generation are concerned with the object structure and how they are accessed by the operands.

The four modes of selector generation shown are:

1. Short Direct
2. Long Direct
3. Stack Indirect
4. General Indirect

The modes of displacement generation seek the physical location and displacement of objects within a given segment or segment. The four modes of displacement are:

1. Scalar data Reference mode
2. Record Item Reference Mode
3. Static Vector Element Reference mode
4. Dynamic Vector Element Reference mode

MODES OF SELECTOR GENERATION

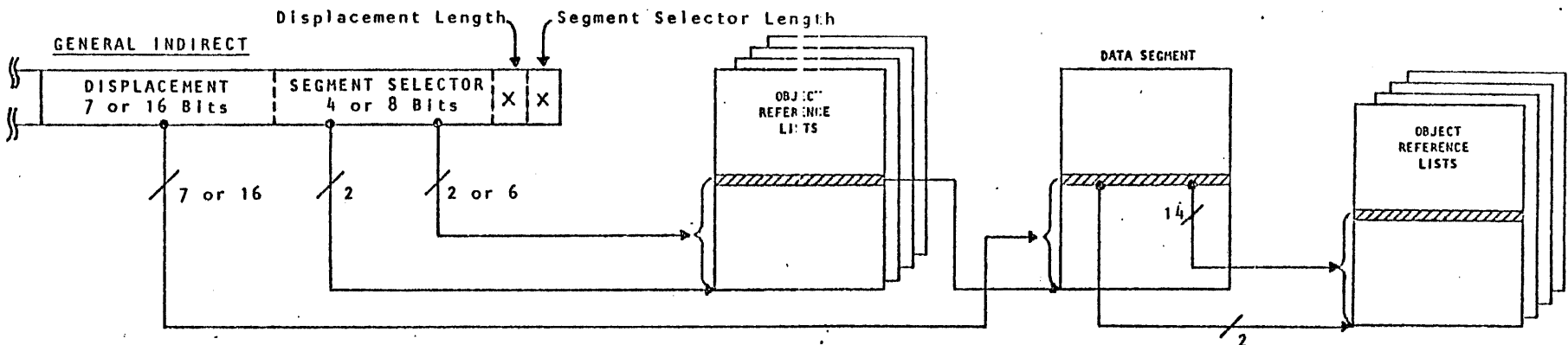
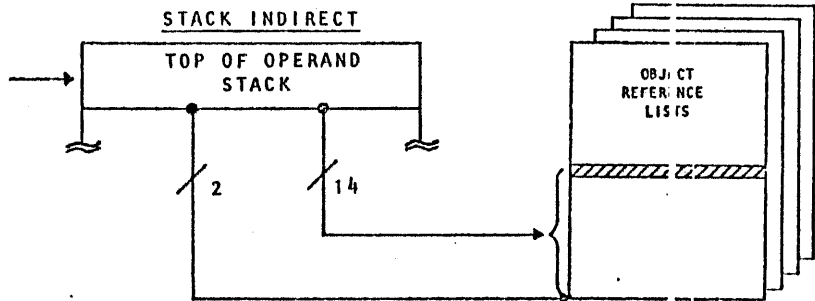
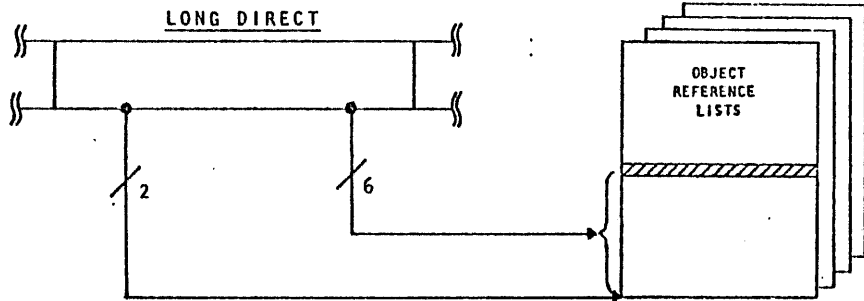
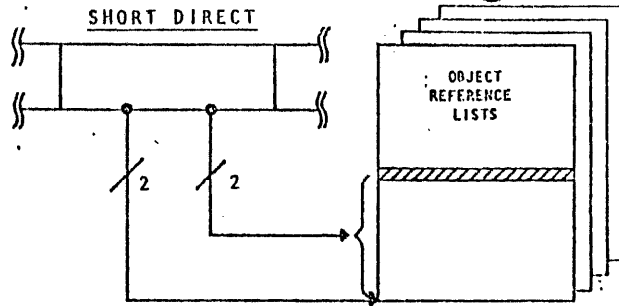
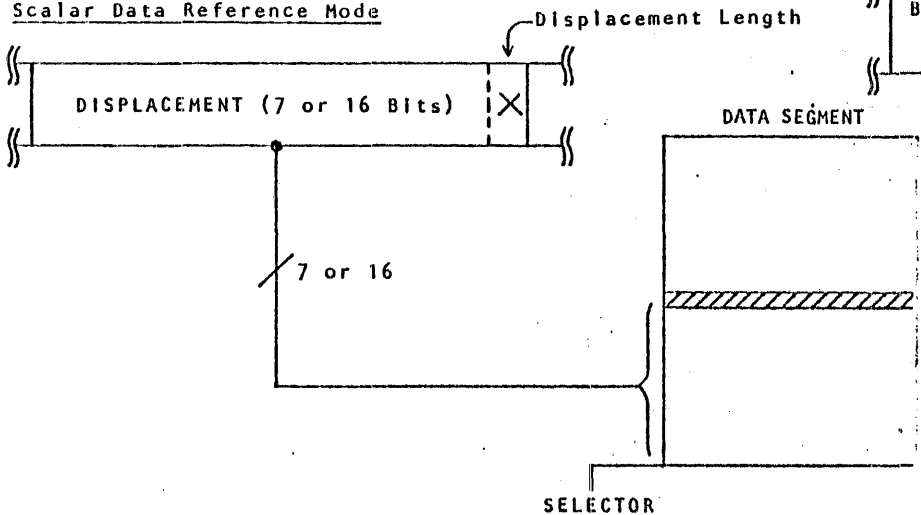


FIG 7. MODES OF NOTATION 39  
OF SECTON 1.911  
SELECTOR GENERATION

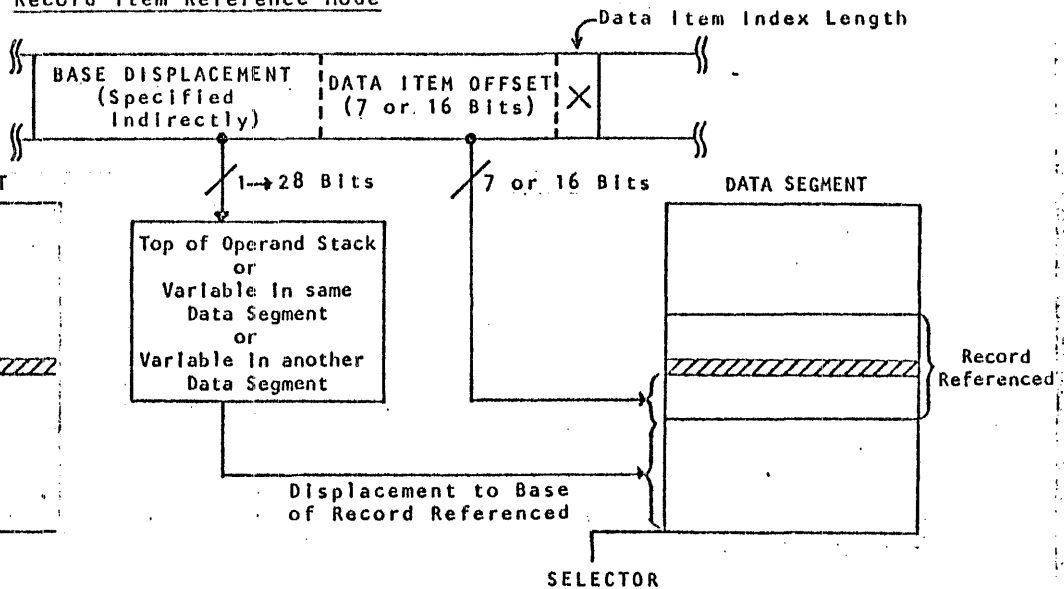
78A

# MODES OF DISPLACEMENT GENERATION

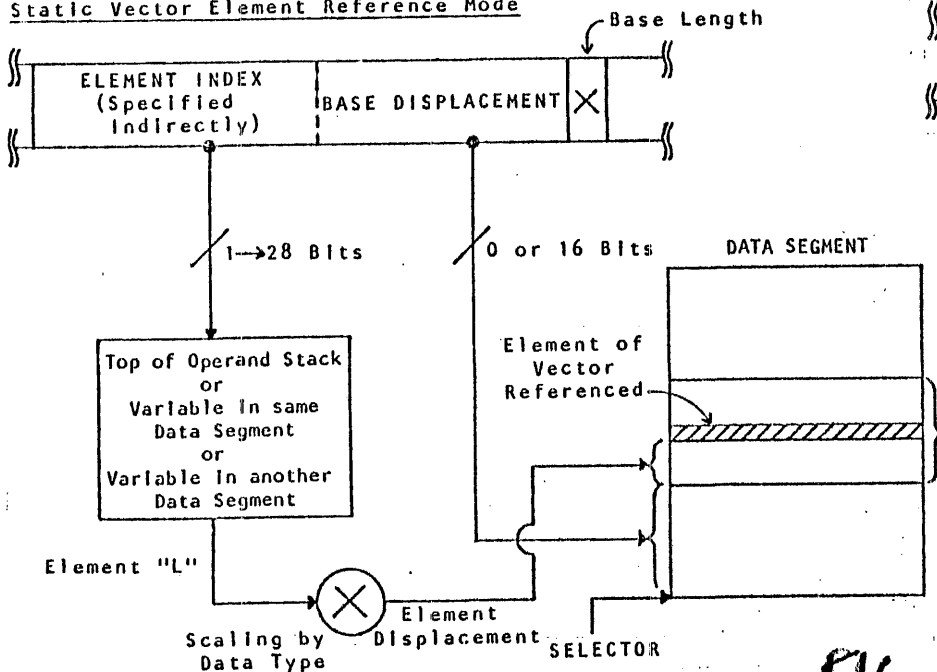
## Scalar Data Reference Mode



## Record Item Reference Mode



## Static Vector Element Reference Mode



## Dynamic Vector Element Reference Mode

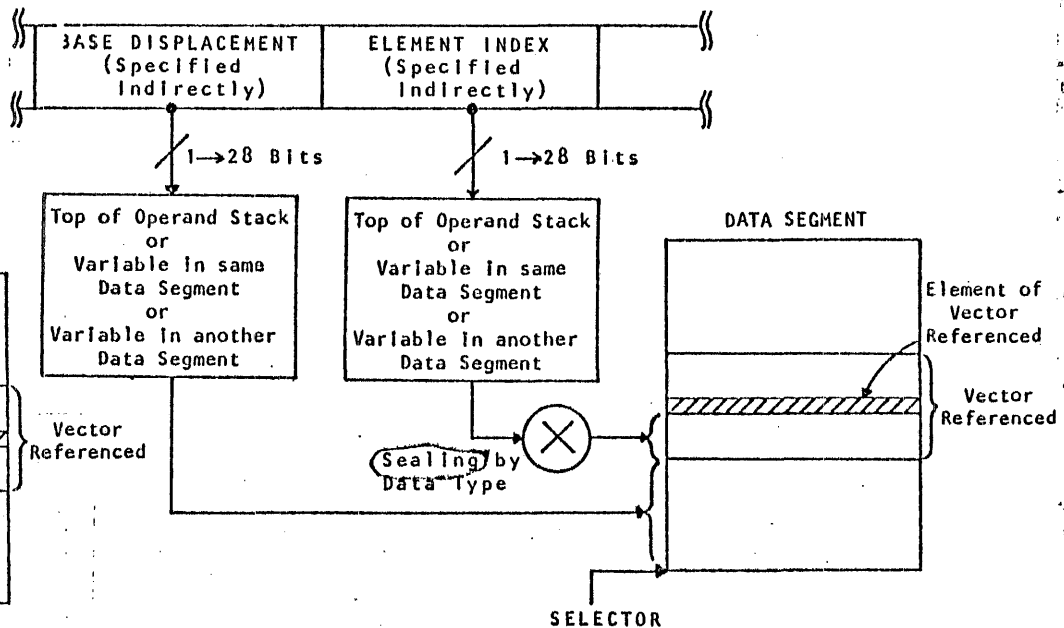


FIG 8. MODES OF DISPLACEMENT GENERATION



### Microarchitecture example

This subsection describes the general internal organization (microarchitecture) of the GDP and describes an example of its operation. The example is intended to provide a "feeling" for the microarchitecture and is by no means comprehensive. Subsequent sections will describe specific hardware elements. No attempt is made to describe the detailed techniques by which exceptional events are handled.

The GDP pipeline starts when the ID passes starting addresses (determined by interpreting the instruction stream) to the MIS, which in turn feeds to the EU sequences of microinstructions that are appropriate to the current macroinstruction. The EU then executes the microinstructions. Most micro instructions are executed by the EU in a single cycle each. However, both the RGU and the DMU contain state machines that are capable of multiple-cycle responses to certain microinstructions. In the case of those that require multiple cycles for execution, the MIS waits for notification of completion from the EU before advancing its state. The ID, however, can advance to the next macroinstruction and often goes on decoding without waiting for the MIS.

With some knowledge of the instruction set code and the

hardware pipeline it becomes possible to understand the tasks facing each of the pipeline stages.

### Instruction Operator Set

Refer to Table 1 for the iAPX General Data Processor operator set summary.

Figure 9. GDP Packet Bus State Diagram

SCA

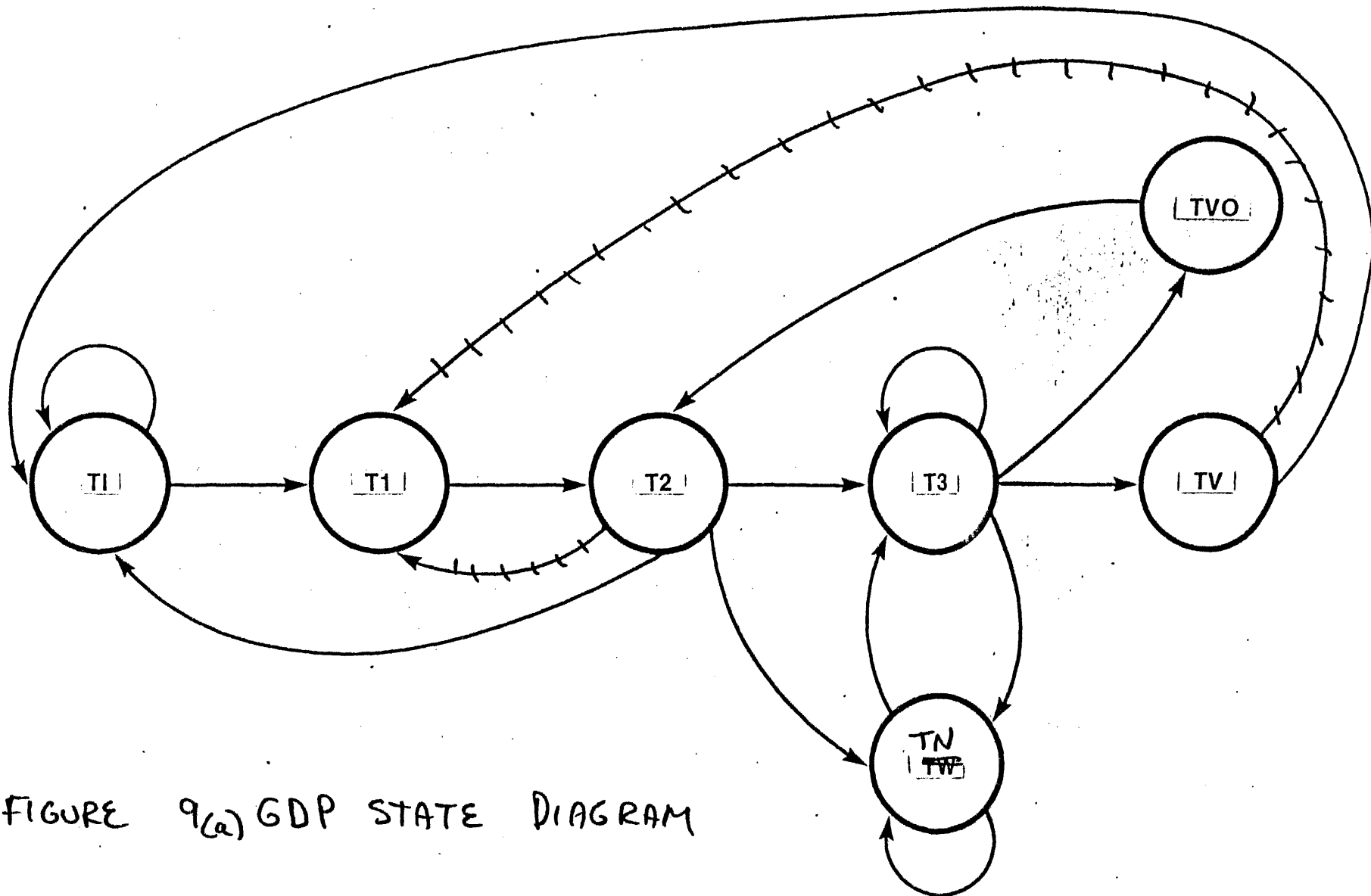
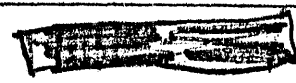


FIGURE 9(a) GDP STATE DIAGRAM

838

<u>INITIAL STATE</u>	<u>NEXT STATE</u>	<u>TRIGGER</u>
T1	T1	Bus cycle desired
T1	T1	No bus cycle desired
T2	T2	Unconditional
	T3	ICS high
	TW	ICS low
	<del>T1</del>	<del>Cancelled, Access pending</del>
	<del>T1</del>	<del>Access Cancelled, No Access pending</del>
T3	T3	Additional transfer required
	TW	ICS low
	TV	All transfers completed if current cycle
	<del>TV</del>	<del>overlapped write</del>
TV	T1	is lead or if write but no pending write
	<del>T1</del>	<del>Current write with pending write</del>
	T2	Current write with overlapped write
<del>TV0</del>	<del>T2</del>	<del>Overlapped write</del>
TW	TW	ICS low
	T3	ICS high

FIGURE 9 (b)



## 43201 PIN DESCRIPTION

### Processor Packet bus Group

**ACD<sub>15</sub> - ACD<sub>00</sub>** (Address/Control/Data lines, Inputs)

The processor Packet bus Address/Control/Data lines are the basic communication path between the GDP and its environment. These lines are always inputs to the 43201 and are driven by either the 43202 or the external environment. Note that the 43201 must receive the specification byte from the 43202 during T1 of a bus transaction (Figure 9). As a result, the ACD receivers must be capable of slave timing as well as processor timing (see processor Packet bus timing relationships for definition of processor and slave timing).

**PRQ** (Processor Packet bus Request, Input, high asserted)

The PRQ input is used to initiate a transaction between the GDP and the bus interface. PRQ is normally held low by the 43202 whenever there is no transaction. PRQ is asserted high during the first cycle of a bus transaction and returns low during the second cycle if the transaction is to be completed. The GDP may cancel a bus transaction by continually asserting

PRQ high (instead of returning it low) during the second cycle of the transaction. The GDP will cancel a transaction if a bounds or access rights violation for the transaction has been detected. PRQ is sampled on the rising edge of CLKB.

**ICS** (Interconnect Status, Input, high asserted)

The ICS input is continually monitored by the 43201 to determine the state of bus transactions. The interpretation of ICS depends on the present cycle of a bus transaction and will indicate one of the following states:

1. Interprocessor communication (IPC) message waiting.
2. Input data invalid, a stretched access.
3. Output data not taken, a stretched access.
4. Bus error in external environment.

During idle periods (GDP not using the bus) the bus interface may signal the GDP on ICS that an Interprocessor communication message has been received. During a bus transaction, the bus interface will use ICS to handle bus protocol. (Refer to Figure 9 and Table 1

for the system relationship of the Packet bus group.)

### Intra-GDP Bus Group

**uI<sub>15</sub>...uI<sub>0</sub>** (Microinstruction Bus lines, Outputs)

These lines are used to transmit microinstructions from the 43201 to the 43202. These pins are high impedance in the checker state (Refer to Hardware Error Detection Group). They are monitored by the hardware error checking logic.

**IS<sub>6</sub>...IS<sub>0</sub>** (Interchip Status lines, Inputs)

The 43201 receives information pertaining to micro program status from the 43202 over these lines.

### System Error Group

**FATAL/** (Fatal, Output, low asserted)

FATAL/ is asserted by the 43201 under microcode control and is used by the GDP microcode to indicate to the system that the GDP cannot continue due to grossly incorrect information structures in memory. FATAL/ is synchronously asserted low and remains low until the processor is initialized. FATAL/ is not affected by the hardware checking logic.

**ALARM/** (Alarm signal, Input, low asserted)

The ALARM/ input signals the occurrence of an unusual system-wide condition (such as power fail). The 43201 does not respond to ALARM/ until it has completed execution of the current 432 instruction, i.e., if any instruction is currently under execution. ALARM/ is active low and is sampled on the rising edge of CLKA.

#### System-Wide Group

**INIT/** (Initialization, Input, low asserted)

The INIT/ pin is used to establish initialization. INIT/ must be asserted low for at least 10 CLKA cycles before the initial state is reached to allow time for the 43201 to begin execution of a microcode sequence that initializes all of the 43201 and 43202 internal registers. Once this initialization sequence has been completed, normal operation begins.

**CLR/** (Clear, Input, low asserted)

Assertion of CLR/ causes the 43201 to immediately trap to a microcode flow that halts the 43202, asserts FATAL/ and waits for a local IPC.



## Hardware Error Detection Group

**MASTER** (Master, Input, high asserted)

The MASTER pin is used to place the processor in either master or checker mode. MASTER is sampled during initialization (INIT/ asserted). If MASTER is asserted throughout initialization, the 43201 functions normally and drives the microinstruction bus. If MASTER is low throughout initialization, microinstruction bus signals  $uI_{15}$ - $uI_0$  go to their high-impedance state. A 43201 thus conditioned does not drive the microinstruction bus; rather, the bus is monitored and compares the data on the bus to its internally generated result, signaling disagreement on its HERR/ line. MASTER should be tied to  $V_{CC}$  for normal operation and pulsed low to enable hardware error checking and disable the bus ( $uI_{15}$ - $uI_0$ ) outputs.

**HERR/** (Hardware Error, Output, low asserted)

HERR/ is a signal produced by the 43201 to indicate disagreement between the data appearing on the microinstruction bus ( $uI_{15}$  -  $uI_0$ ) and the internally generated result of the 43201. HERR/ is asserted low when disagreement occurs and is valid during CLKA.

## Clock Group

**CLKA, CLKB** (Clock A, Clock B, Inputs)

Clock A provides the basic timing reference for the 43201. Clock B (CLKB) overlaps CLKA by nominally 1/4 cycle (90 degrees phase shift). All external signals are referenced to CLKA. Refer to the A. C. Electrical Characteristics for exact statement of timing relationships.

## Testing Input

**RDROM/** (Read ROM, Input, low asserted)

The RDROM input line is used to force a sequential read of Read-Only-Memory. When the RDROM/ line is asserted low throughout initialization (INIT/ asserted), the 43201 goes into a special diagnostic mode. In this mode, the 43201 microinstruction sequencer steps through the 43201 microprogram ROM, sequentially displaying (but not executing) the 43201 microprogram on the uI<sub>15</sub>-uI<sub>0</sub> lines. The sequencer continues to cycle through the microprogram ROM until INCR uA/ is no longer asserted. The INCR uA/ feature is useful for testing, but should not be used during normal operation since it could lead to unpredictable results. INCR uA/ should be tied to V<sub>CC</sub> for normal

operation and only asserted low for testing (Power, ground, not connected).

**V<sub>CC</sub>** (4 pins)

These pins supply +5 V  $\pm$  10 % referenced to GND pins.

**GND** (5 pins)

These pins supply ground reference for the 43201.

**N.C.** (No Connection, 5 pins)

## 43202 PIN DESCRIPTION

### Processor Packet Bus Group

**ACD<sub>15</sub>-ACD<sub>0</sub>** (Address/Control/Data lines, Inputs or Three-state Outputs, high asserted)

The processor Packet bus Address/Control/Data lines are the basic communication path between the GDP and its environment. These pins are used three ways:

1. They may indicate control information for bus transactions,
2. they may issue physical addresses generated within by the GDP for an access, or
3. they may transfer data (either direction).

The ACD pins are monitored by the hardware error checking logic when the 43202 is in checker mode and are conditionally in the high impedance mode.

**PRQ** (Processor Packet bus Request, Three-state Output, low asserted)

PRQ is used to indicate the presence of a transaction between the GDP and its external environment. Normally

low, the PRQ pin is brought high during the same cycle as the first double-byte of address information is being driven onto the ACD pins. PRQ remains high for only one cycle during the access, unless an address development fault occurs. The 43202 will leave PRQ high for a second cycle to indicate the GDP has detected an addressing or segment rights fault in completing address generation. PRQ is checked by the hardware error logic. PRQ is in a high impedance state when the 43202 is in checker mode (see MASTER description).

**ICS (Interconnect Status, Input, low asserted)**

ICS is an indication to the 43202 from the bus interface circuitry concerning the status of a bus transaction. The interpretation of the ICS state is dependent upon the present cycle of a bus transaction and may indicate:

1. Interprocessor communication (IPC) message waiting,
2. Input data invalid,
3. Output data not taken,
4. Bus error in external environment.

During idle periods (when the GDP is not using the bus) the bus interface may signal the GDP on ICS an Interprocessor communication message has been received. During a bus transaction, the bus interface will use ICS to handle bus protocol, i.e., data not taken, or (for a read) data invalid. When the 43202 is in checker mode (see MASTER description) ICS is always asserted.

**BOUT** (Enable Output Buffers, Output, high asserted)

BOUT is used to control bus external transceivers to buffer the 43201, 43202 from the processor component bus load. Though not required, the use of buffers may be desired in systems with heavy loading. BOUT is asserted when information is to leave the 43202 on the ACD lines.

#### Intra-GDP Bus Group

**uI<sub>15</sub>-uI<sub>0</sub>** (Microinstruction lines, Inputs, high asserted)

The uI<sub>15</sub>-uI<sub>0</sub> input lines provide the 43202 with microinstruction information sent from the 43201.

**IS<sub>6</sub>-IS<sub>0</sub>** (Microprogram Status lines, Outputs, high asserted)

The IS<sub>6</sub>-IS<sub>0</sub> lines drive microprogram status information from the 43201 to the 43202.

### System Group

**PCLK/** (Processor Clock, Input, low asserted)

PCLK/ is implemented to change the state of two processor timers. The affected timers are called the system timer and the service timer. Assertion of PCLK/ for one cycle causes the system timer to increment and the service timer to decrement. Assertion of PCLK/ for more than one cycle causes the system timer to be cleared and decrements the service timer. For proper operation PCLK/ must be unasserted for at least one cycle before being asserted. PCLK/ is synchronous with respect to CLKA, but is generally unrelated to other interface timings.

**CLR/** (Clear, Input, low asserted)

The Clear (CLR/) input causes the 43202 to cease the execution of any multiple cycle microinstruction under way when CLR/ is asserted.

## Hardware Error Detection Group

**MASTER** (Master, Input, high asserted; 25 k nominal pullup on-chip).

The MASTER input determines whether the 43202 is to function as a master or a checker. MASTER is continually sampled by an internal flip-flop. When MASTER is seen to have changed state, all output lines will be controlled appropriately (driven or allowed to float) within 2 cycles (Such a change in state will not result in a spurious HERR/ assertion). The Checker mode affects the behavior of ACD<sub>15</sub>-ACD<sub>0</sub>, PRQ, and BOUT. ACD<sub>15</sub>-ACD<sub>0</sub> and PRQ enter the high impedance state and BOUT is unconditionally low.

**HERR/** (Hardware Error, Output, low asserted).

The HERR/ output indicates a hardware error has been detected.

## Clock Group

**CLKA, CLKB** (Clock A, Clock B, Inputs)

Clock A (CLKA) provides the basic timing reference for the 43202. Clock B (CLKB) overlaps CLKA by nominally 1/4 cycle (90 degrees phase shift). Refer to the A. C.



Electrical Characteristics for exact statement of timing relationships. All external signals are referenced to CLKA. Power, ground not connected.

**V<sub>CC</sub>** (Power Supply, 4 pins)

These pins supply +5 V  $\pm 10\%$ , referenced to GND pins.

**GND** (Ground, 5 pins)

These pins supply ground reference for the 43202.

**N.C.** (No Connection, 7 pins)

Table 8<sup>2</sup> 1

General Data Processor Operator Set Summary

Character Operators

Move Character

Zero Character

One Character

Save Character

AND Character

OR Character

XOR Character

XNOR Character

Complement Character

Add Character

Subtract Character

Increment Character

Decrement Character

Equal Character

Not Equal Character

Equal Zero Character

Not Equal Zero Character

Greater Than Character

Greater Than or Equal Character

## Convert Character to Short Ordinal

### Short-Ordinal Operators

Move Short Ordinal

Zero Short Ordinal

One Short Ordinal

Save Short Ordinal

AND Short Ordinal

OR Short Ordinal

XOR Short Ordinal

XNOR Short Ordinal

Complement Short Ordinal

Extract Short Ordinal

Insert Short Ordinal

Significant Bit Short Ordinal

Add Short Ordinal

Subtract Short Ordinal

Increment Short Ordinal

Decrement Short Ordinal

Multiply Short Ordinal

Divide Short Ordinal

Remainder Short Ordinal

Equal Short Ordinal  
Not Equal Short Ordinal  
Equal Zero Short Ordinal  
Not Equal Zero Short Ordinal  
Greater Than Short Ordinal  
Greater Than or Equal Short Ordinal  
  
Convert Short Ordinal to Character  
Convert Short Ordinal to Ordinal  
Convert Short Ordinal to Temporary Real

#### Short-Integer Operators

Move Short Integer  
Zero Short Integer  
One Short Integer  
Save Short Integer  
  
Add Short Integer  
Subtract Short Integer  
Increment Short Integer  
Decrement Short Integer  
Negate Short Integer  
Multiply Short Integer  
Divide Short Integer  
Remainder Short Integer  
  
Equal Short Integer

Not Equal Short Integer  
Equal Zero Short Integer  
Not Equal Zero Short Integer

Greater Than Short Integer  
Greater Than or Equal Short Integer  
Positive Short Integer  
Negative Short Integer

Convert Short Integer to Integer  
Convert Short Integer to Temporary Real

#### Ordinal Operators

Move Ordinal  
Zero Ordinal  
One Ordinal  
Save Ordinal

AND Ordinal  
OR Ordinal  
XOR Ordinal  
XNOR Ordinal  
Complement Ordinal

Extract Ordinal  
Insert Ordinal

Significant Bit Ordinal

Add Ordinal

Subtract Ordinal

Increment Ordinal

Decrement Ordinal

Multiply Ordinal

Divide Ordinal

Remainder Ordinal

Equal Ordinal

Not Equal Ordinal

Equal Zero Ordinal

Not Equal Zero Ordinal

Greater Than Ordinal

Greater Than or Equal Ordinal

Convert Ordinal to Short Ordinal

Convert Ordinal to Integer

Convert Ordinal to Temporary Real

#### Integer Operators

Move Integer

Zero Integer

One Integer

Save Integer

Add Integer  
Subtract Integer  
Increment Integer  
Decrement Integer  
Negate Integer  
Multiply Integer  
Divide Integer  
Remainder Integer

Equal Integer  
Not Equal Integer  
Equal Zero Integer  
Not Equal Zero Integer  
Greater Than Integer  
Greater Than or Equal Integer  
Positive Integer  
Negative Integer

Convert Integer to Short Integer  
Convert Integer to Ordinal  
Convert Integer to Temporary Real

#### Short-Real Operators

Move Short Real  
Zero Short Real  
Save Short Real

Add Short Real - Short Real  
Add Short Real - Temporary Real  
Add Temporary Real - Short Real  
Subtract Short Real - Short Real  
Subtract Short Real - Temporary Real  
Subtract Temporary Real - Short Real  
Multiply Short Real - Short Real  
Multiply Short Real - Temporary Real  
Multiply Temporary Real - Short Real  
Divide Short Real - Short Real  
Divide Short Real - Temporary Real  
Divide Temporary Real - Short Real  
Negate Short Real  
Absolute Value Short Real

#### Short-Real Operators

Equal Short Real  
Equal Zero Short Real  
Greater Than Short Real  
Greater Than or Equal Short Real  
Positive Short Real  
Negative Short Real  
  
Convert Short Real to Temporary Real

#### Real Operators



Move Real

Zero Real

Save Real

Add Real - Real

Add Real - Temporary Real

Add Temporary Real - Real

Subtract Real - Real

Subtract Real - Temporary Real

Subtract Temporary Real - Real

Multiply Real - Real

Multiply Real - Temporary Real

Multiply Temporary Real - Real

Divide Real - Real

Divide Real - Temporary Real

Divide Temporary Real - Real

Negate Real

Absolute Value Real

Equal Real

Equal Zero Real

Greater Than Real

Greater Than or Equal Real

Positive Real

Negative Real

Convert Real to Temporary Real

## Temporary-Real Operators

Move Temporary Real

Zero Temporary Real

Save Temporary Real

Add Temporary Real

Subtract Temporary Real

Multiply Temporary Real

Divide Temporary Real

Remainder Temporary Real

Negate Temporary Real

Square Root Temporary Real

Absolute Value Temporary Real

Equal Temporary Real

Equal Zero Temporary Real

Greater Than Temporary Real

Greater Than or Equal Temporary Real

Positive Temporary Real

Negative Temporary Real

Convert Temporary Real to Ordinal

Convert Temporary Real to Integer

Convert Temporary Real to Short Real

Convert Temporary Real to Real

## Access Descriptor Movement Operators

Copy Access Descriptor

Null Access Descriptor

### Rights Manipulation Operators

Amplify Rights

Restrict Rights

### Type Definition Manipulation Operators

Create Public Type

Create Private Type

Retrieve Public Type Representation

Retrieve Type Representation

Retrieve Type Definition

### Refinement Operators

Create Generic Refinement

Create Typed Refinement

Retrieve Refined Object

### Segment Creation Operators

Create Data Segment

Create Access Segment

Create Typed Segment

Create Access Descriptor

Access Path Inspection Operators

Inspect Access Descriptor

Inspect Access

Object Interlock Operators

Lock Object

Unlock Object

Indivisibly Add Short Ordinal

Indivisibly Add Ordinal

Indivisibly Insert Short Ordinal

Indivisibly Insert Ordinal

Branch Operators

Branch

Branch True

Branch False

Branch Indirect

Branch Intersegment

Branch Intersegment without Trace

Branch Intersegment and Link

Context Communication Operators

Enter Access Segment  
Enter Global Access Segment  
Set Context Mode  
Call Context  
Call Context with Message  
Return from Context

Process Communication Operators

Send  
Receive  
Conditional Send  
Conditional Receive  
Surrogate Send  
Surrogate Receive  
Delay  
Read Process Clock

Processor Communication Operators

Send to Processor  
Broadcast to Processors  
Read Processor Status and Clock

Interconnect Operators

Move to Interconnect

Move from Interconnect

(Note) Each of these operators is identical to the operator with the same assigned number and is specified by the same operator code.

## iAPX 432 Timing/ Characteristics

Figures 10 through 15 contain input/output timing, clock input specifications, error checking timing, initialization timing, and microcode interrogation timing for the 43201. Tables 2, 3, and 4 contain the A. C., D. C., and capacitance specifications for the 43201.

Figures 16 through 19 contain input/output timing, BOUT timing, and input clock timing for the 43202. Tables 5, 6, and 7 include the A. C., D. C., and capacitance specifications for the 43202.

Table 2. 43201 D. C. Electrical Characteristics

Table 3. A. C. Electrical Characteristics

Table 4. 43201 Capacitance

Figure 10. 43201 Output Timing Specification

Figure 11. 43201 Input Timing Specification

Figure 12. Clock Input Specification

Figure 13. 43201 Hardware Error Check Timing

Figure 14. 43201 Initialization Timing

Figure 15. Microcode Interrogate Timing

Table 5. 43202 DC Characteristics

Table 6. 43202 AC Characteristics

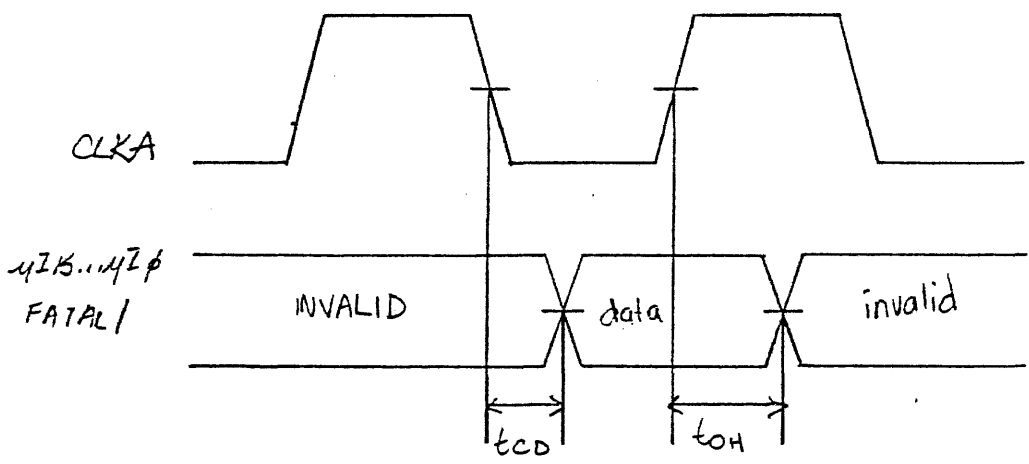
Table 7. 43202 Capacitance

Figure 16. 43202 Output Timing Specification

Figure 17. 43202 Input Timing Specification

Figure 18. 43202 BOUT/ Timing Specification





43201

FIGURE 10. ~~DATA~~ OUTPUT TIMING SPECIFICATION

108A

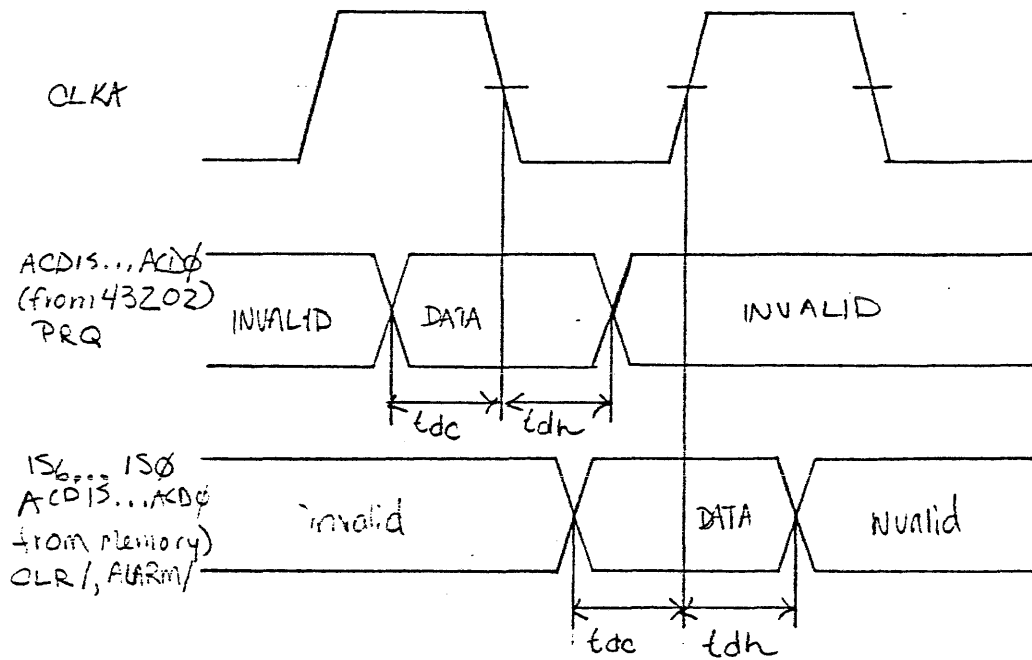
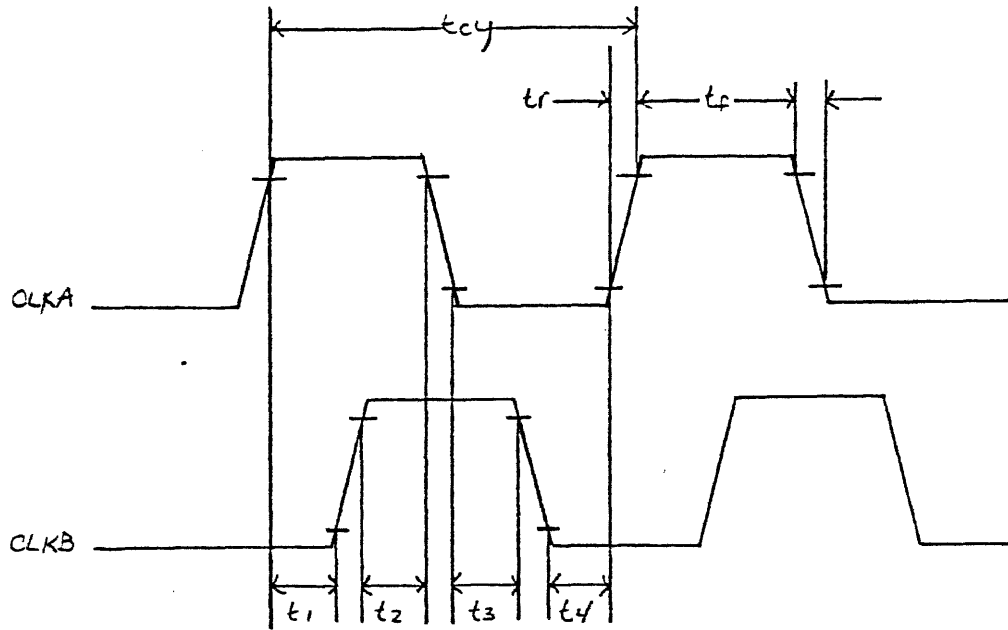


FIGURE 11. ~~43201~~ INPUT TIMING SPECIFICATION

108B



CLOCK INPUT SPECIFICATION

FIGURE 12 CLOCK INPUT SPECIFICATION

DISAGREEMENT

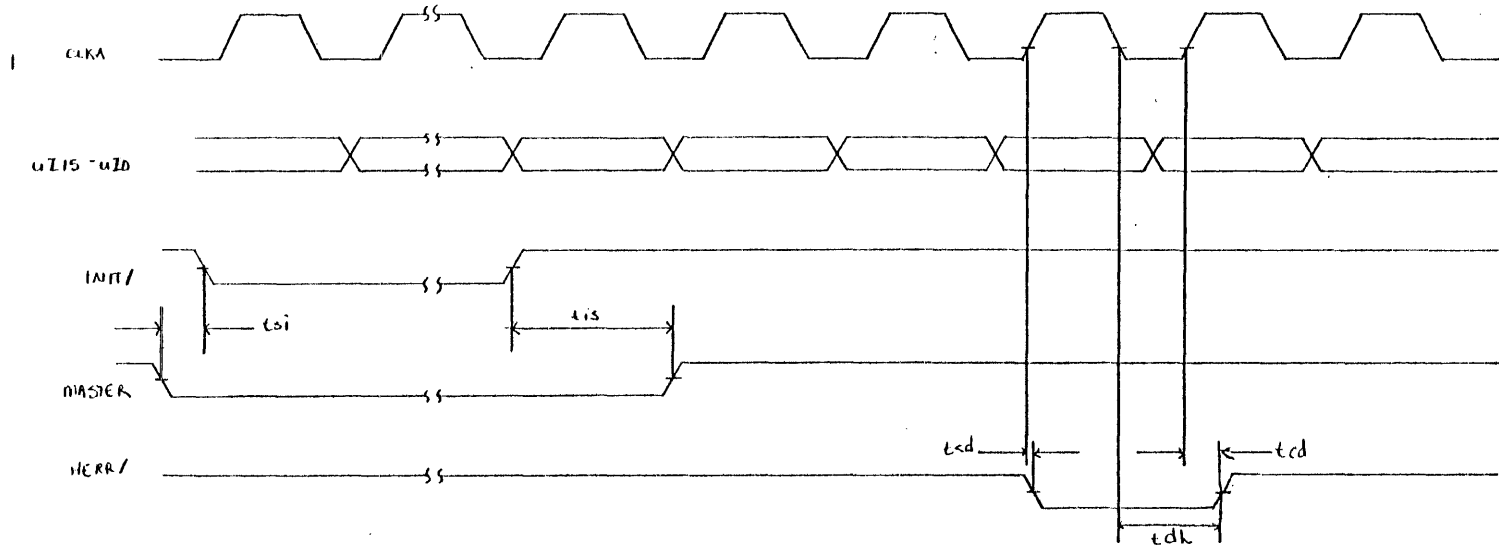


FIGURE 13.

43201  
HARDWARE ERROR CHECK TIMING

1251

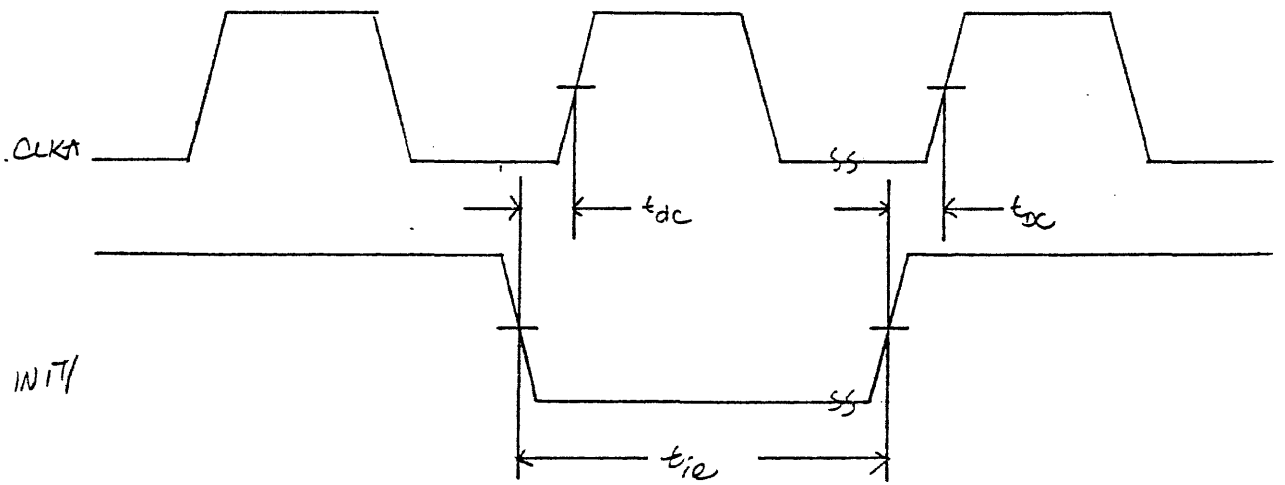


FIGURE 14 iAPX 43201 initialization timing

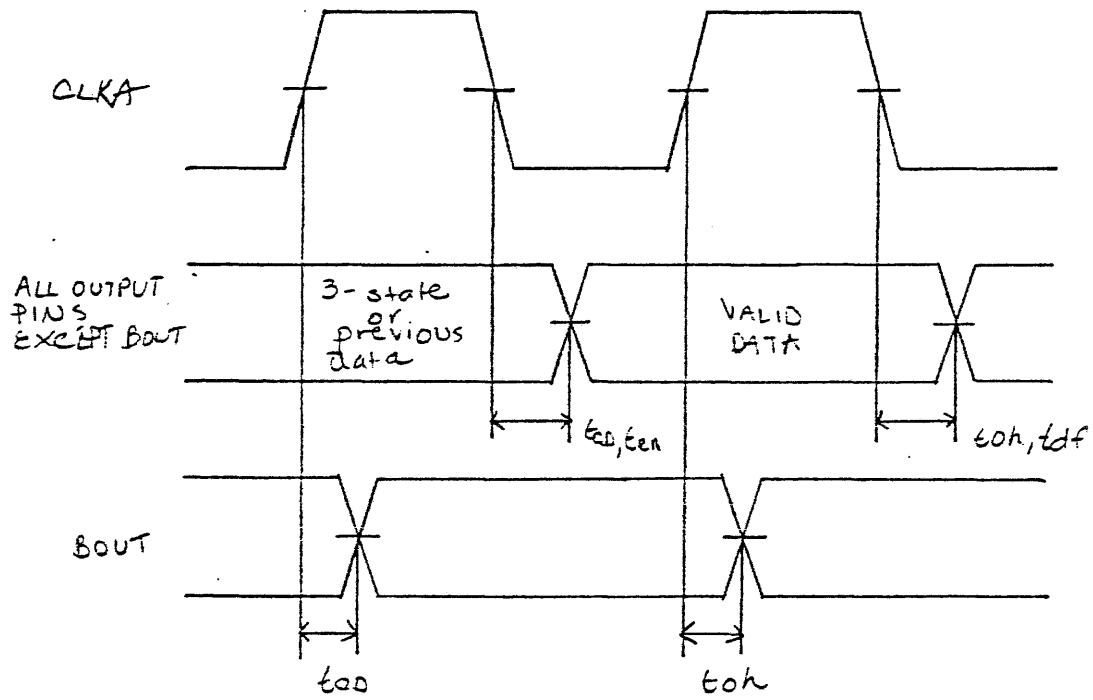
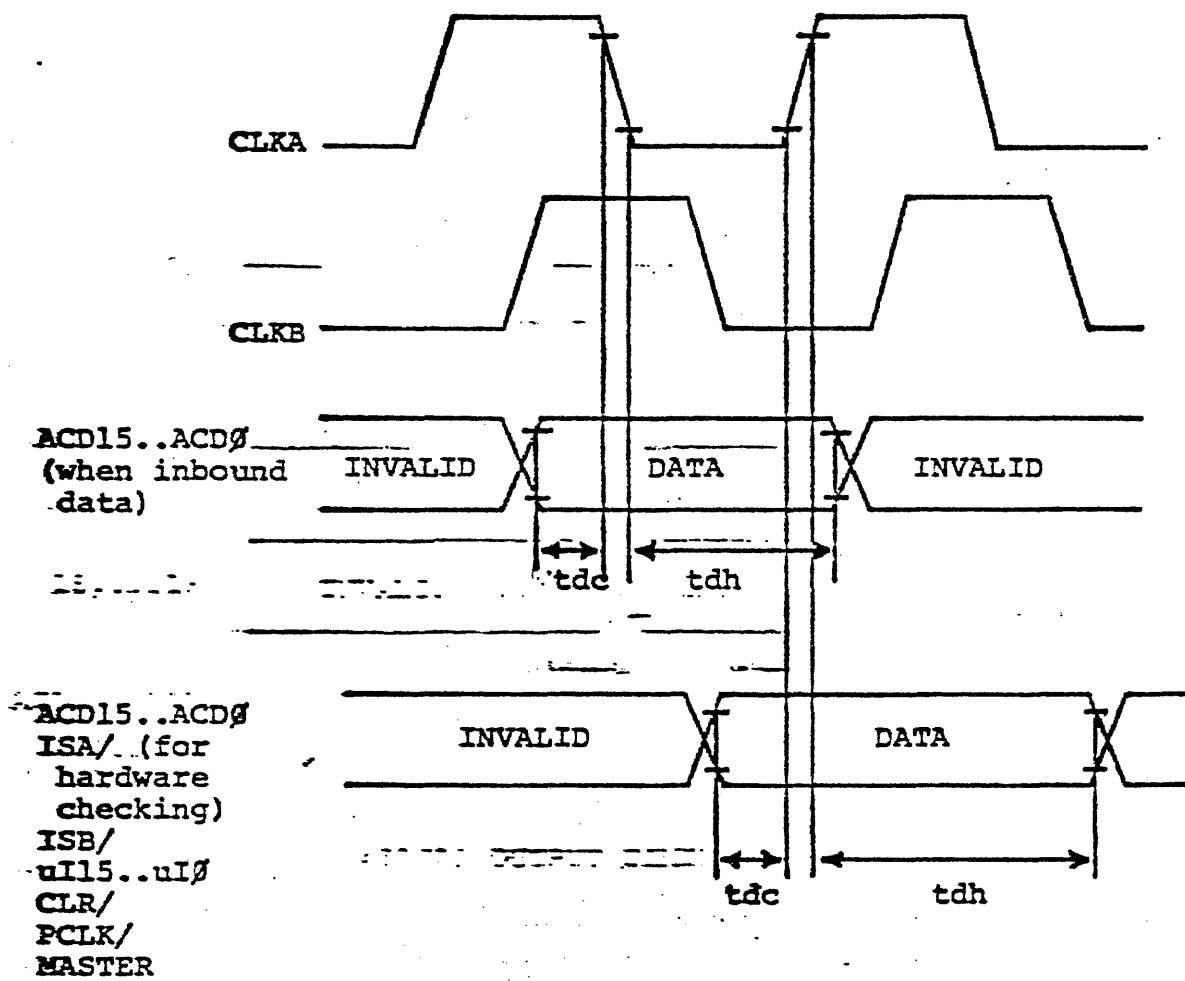


FIGURE 16 iAPX 43202 output timing specs



432/02 INPUT TIMING SPECIFICATION

FIG 17 INPUT TIMING

## Absolute Maximum Ratings

Ambient ~~temp~~ temperature Under Bias  $0$  to  $70^{\circ}\text{C}$

Storage  $-65^{\circ}\text{C}$  to  $+150^{\circ}\text{C}$ .

Voltage on any pin with respect to ~~ground~~  $-1$  to  $+7$

Power Dissipation  $1\text{01}$  2.5 Watt,  $1\text{02}$  3.5 Watt.

TABLE 1. ABSOLUTE MAXIMUM RATINGS



/01 ELECTRICAL CHARACTERISTICS

D.C. Characteristics

(TA = 0 degrees C to 70 degrees C; VCC = 5V ± 10%; VSS = 0V; unless otherwise specified)

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	CONDS
VILI (IS <sub>6</sub> ...IS <sub>0</sub> )	Input Low Voltage (IS <sub>6</sub> ...IS <sub>0</sub> )	-0.3	+0.7	V	
VIHI (IS <sub>6</sub> ...IS <sub>0</sub> )	Input High Voltage	3.0	VCC+0.5	V	
VILC	Clock Input Low Voltage	-0.3	+0.5	V	
VIHC	Clock Input High Voltage	VCC-1.0	VCC+0.5	V	
VIL(Remaining Inputs)	Input Low Voltage	-0.3	+0.8	V	
VIH(Remaining Inputs)	Input High Voltage	2.0	VCC+0.5	V	
VOL(Micro-Instruction Lines)	Output Low Voltage(uIx)	0	+0.35	V	IOL= 0.1mA
VOH(Micro-Instruction Lines)	Output High Voltage(uIx)	3.25	VCC	V	IOH= -0.1mA
VOL	Output Low Voltage		0.45	V	IOL= 2.0mA
VOH	Output High Voltage	2.4		V	IOH= -0.4mA
ICC	Power Supply Current		400	mA	
IIL	Input Leakage Current		±10	uA	VIN=VCC
ILO	Output Leakage Current		±10	uA	.45V ≤ VOUT ≤ VCC

~~TABLE 1~~

TABLE 2 43201 ELECTRICAL CHARACTERISTICS

Capacitance

SYMBOL	PARAMETER	TYP.	UNIT	CONDITION
Cin	Input Capacitance	6	pF	fc = 1MHz
Cout	Output Capacitance <sup>*</sup>	12	pF	VIN=VOUT=0V

\* Outputs in high impedance state.  
~~Input/output pins are considered~~  
~~output~~

TABLE 4 43201 CAPACITANCE

(OK)

# 102 Electrical Characteristics

(TA = 0°C to 70°C; VCC = 5V ± 10%; VSS = 0V;

unless otherwise specified)

Judgy - this stuff should be in the same table format as the 101

D.C. Characteristics

<u>SYMBOL</u>	<u>PARAMETER</u>	<u>MIN</u>	<u>MAX</u>	<u>UNIT</u>
VCC	Supply Voltage	4.50	5.50	V
ICC	Supply Current		455	mA
VIL	Clock Input Low Voltage	-0.3	0.5	V
VIH	Clock Input Hi Voltage for CLKA, CLKB inputs	VCC-1.0	VCC+0.5	V
VILuI	uI Input Low Voltage	-0.3	0.7	V
VIHuI	uI Input Hi Voltage for uI <sub>0</sub> - uI <sub>15</sub>	3.0	VCC+0.5	V
VIL	Input Low Voltage	-0.3	0.8	V
VIH	Input Hi Voltage for remaining pins	2.0	VCC+0.5	V
VOL	Output Low Voltage @IOL = 2 mA for ACDx, ISA/, HERR/; 8 mA for BOUT		0.45	V
VOH	Output Hi Voltage @IOH = 50 uA for ACDx; 200 uA for BOUT	2.4	VCC	V
VOL (IS <sub>6</sub> ...IS <sub>0</sub> )	Output Low Voltage @IOL = .1 mA		0.35	V
VOH (IS <sub>6</sub> ...IS <sub>0</sub> )	Output Hi Voltage @IOH = -.1 mA	3.25	VCC	V
ILI	Input Leakage (Except MASTER)		±10.0	uA
ILIL	Input Leakage MASTER input		-400	uA
ILO	Output Leakage @Vout = (0.45 .. Vcc)		±10.0	uA

TABLE 5 43202  
DC CHARACTERISTICS

~~108K~~  
108K

Capacitance

SYMBOL	PARAMETER	TYP.	UNIT	CONDITION
C <sub>in</sub>	Input Capacitance	6	pF	f <sub>c</sub> = 1MHz
C <sub>out</sub>	Output Capacitance <sup>*</sup>	12	pF	V <sub>IN</sub> =V <sub>OUT</sub> =0V

\*Outputs in high impedance state.  
Input/output pins are considered  
outputs.

TABLE 7 43202  
CAPACITANCE RATINGS

~~TABLE 6~~

## iAPX 43203

### VLSI INTERFACE PROCESSOR

- o Fully protected I/O interface to 432 memory
- o Silicon operating system instruction set extensions for attached iAPX processors
- o Buffered data path for high speed burst mode transfers
- o Multibus<sup>TM</sup> system compatible interface
- o Initialization/diagnostic interface to 432 systems
- o Functional redundancy checking mode for hardware error detection
- o Multiple 43203's per system provide incremental I/O capacity

The Intel 43203 Interface Processor (IP) provides I/O facilities in iAPX 432 micromainframe systems employing

unprotected peripheral subsystems. An IP maps a portion of peripheral subsystem address space into iAPX 432 system memory. As does any iAPX 432 processor, the IP operates in an object-oriented, descriptor-based, transparent multiprocessing environment.

The 43203 is a VLSI device, fabricated with Intel's highly reliable +5 Volt, depletion load, N-channel, silicon gate HMOS technology, and is packaged in a 64-pin Quad In-Line Package (QUIP).

Figure 1. 43203 Pinout

Figure 2. 43203 Logic Symbol

Figure 3. 43203 Block Diagram

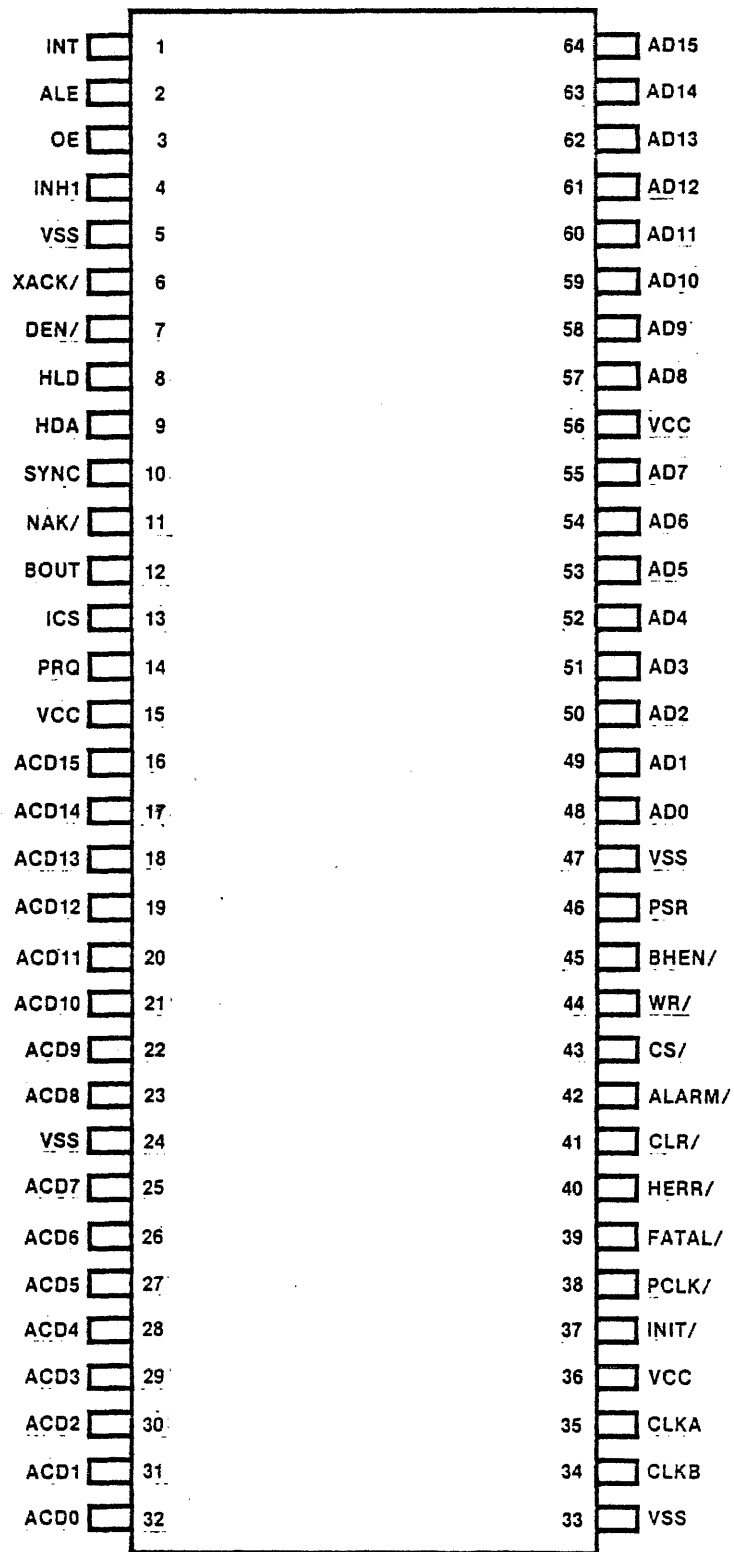


FIGURE 1.

IAPX 432/03 INTERFACE PROCESSOR PIN CONFIGURATION

1104

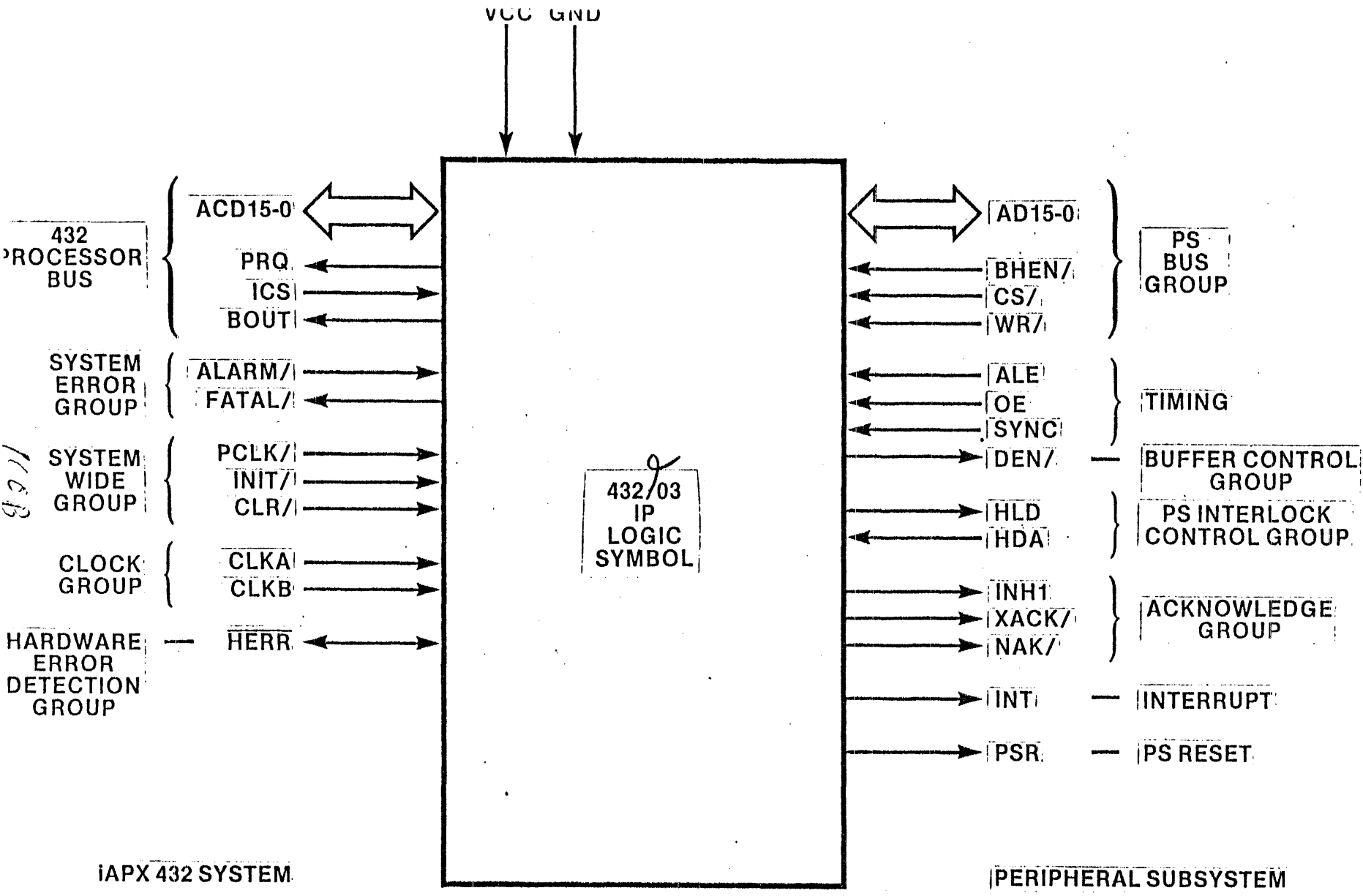


FIGURE 2 43203 IP LOGIC SYMBOL



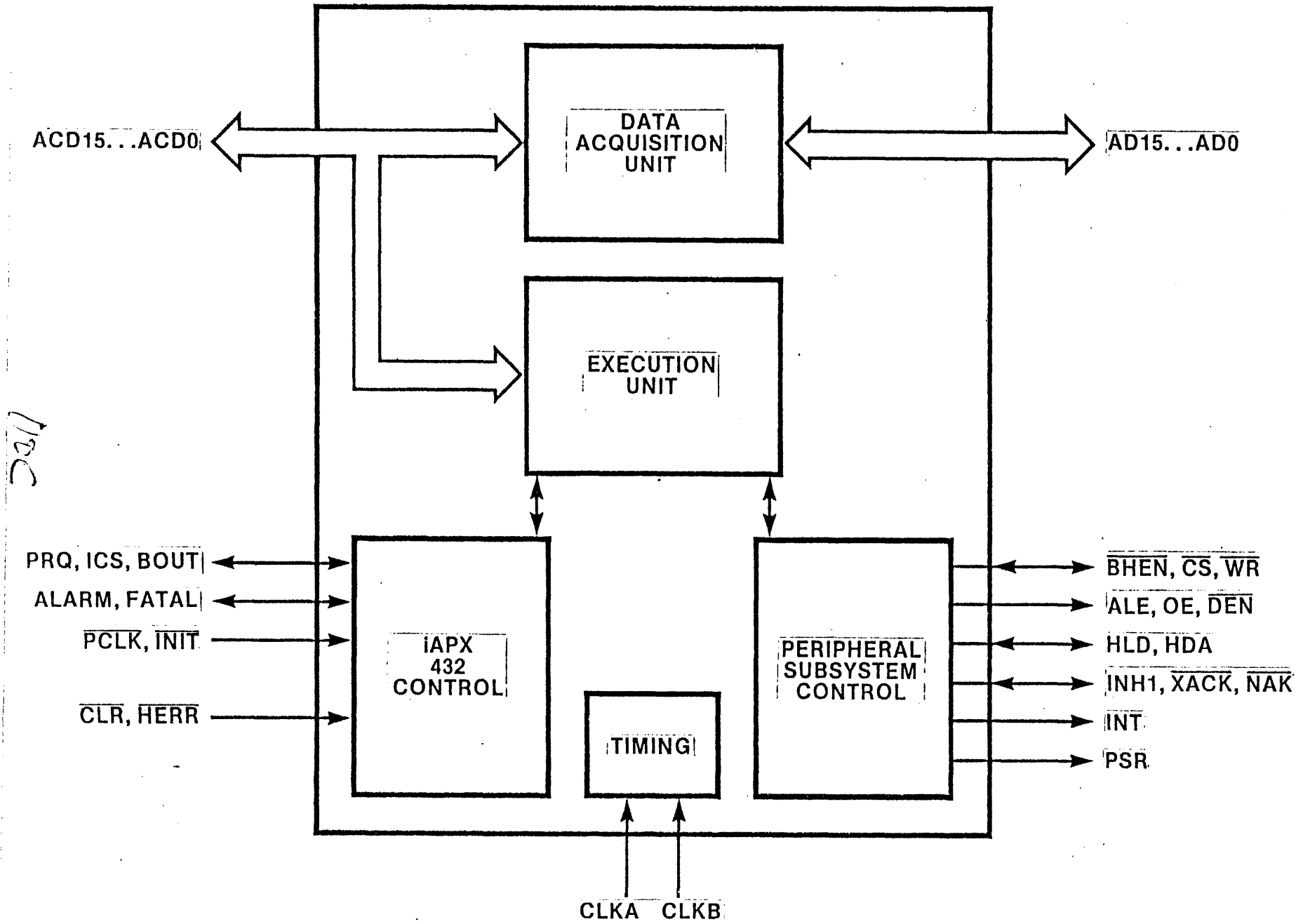


FIGURE 3. IAPX 432/03 IP FUNCTIONAL BLOCK DIAGRAM

## Functional Description

This data sheet describes the iAPX 432 Interface Processor (IP) Component. It is oriented toward hardware designers of iAPX 432 systems. iAPX 432 architectural information is provided in the Intel iAPX 432 General Data Processor Architecture Reference Manual (Order No. 171860). Detailed information about the IP is contained in the Intel iAPX 432 Interface Processor Architecture Reference Manual (Order No. 171863-001). The Intel iAPX 432 Component User's Guide (Order No. 171861) provides additional hardware support information.

The IP, operating in conjunction with an Attached Processor (AP), forms the logical I/O processor of an iAPX 432 system. The IP acts as a slave to the AP, mapping a portion of the AP's peripheral subsystem address space into iAPX 432 system memory with the same protection mechanisms as any iAPX 432 processor. Five peripheral subsystem (PS) memory subranges may be mapped into iAPX 432 memory segments. These five windows (labeled 0 through 4) allow the AP to reference iAPX 432 memory with logical addresses or, in special circumstances, with direct 24-bit physical addresses.

The IP does not execute instructions from an iAPX 432 memory segment, but rather accepts function requests from the AP and performs the specified operations in the iAPX 432 system. Window 4 is designated for this function.

The AP may reference operands in iAPX 432 memory in random mode (random addressing order) or buffered mode (sequential addressing order). Only window 0 can be used in buffered mode. It contains a 16-byte FIFO buffer which postwrites-/prefetches data.

Window 1 may be used to reference the iAPX 432 interconnect address space when the IP is in physical mode.

The sections of this data sheet are:

- o iAPX 43203 Pin Summary
- o 43203 Electrical Characteristics
- o Interfacing Peripheral Subsystems to the IP
- o IP/GDP Operator Comparison

Table 1. 43203 Interface Processor Pin Summary

# TABLE 1

## IAPX 43203 INTERFACE PROCESSOR PIN SUMMARY

### 432 SYSTEM SIDE

PIN GROUP	PIN NAME	DIRECTION	HARDWARE ERROR DETECTION
PROCESSOR	ACD15...ACD0	I/O	X
PACKET BUS GROUP	PRQ	0	X
	ICS	I	
	BOUT	0	
SYSTEM ERROR GROUP	ALARM/ FATAL/ CLR/	I 0 (I at Initialization) I	
SYSTEM-WIDE GROUP	PCLK/ INIT/	I I	
SYSTEM CLOCK GROUP	CLKA CLKB	I I	
HARDWARE ERROR DETECTION GROUP	HERR/	0 (I at Initialization)	

### PERIPHERAL SUBSYSTEM SIDE

PIN GROUP	PIN NAME	DIRECTION	HARDWARE ERROR DETECTION
PERIPHERAL SUBSYSTEM BUS GROUP	AD15...ADO	I/O	X
	BHEN/	I	
	CS/	I	
	WR/	I	
PS TIMING GROUP	ALE	I	
	OE	I	
	SYNC	I	
PS BUFFER CONTROL GROUP	DEN/	0	X
PS INTERLOCK GROUP	HLD	0	X
	HDA	I	
PS SYNCHRONIZATION GROUP	XACK/	0	X
	NAK/	0	X
	INH1	0	X
PS INTERRUPT GROUP	INT	0	X
PS RESET GROUP	PSR	0	X

## 43203 Pin Description

### Processor Packet bus Group

The following pins, shown in Table 2, fully conform to the specification in the iAPX 432 Processor Packet bus definition of the iAPX 432 Component User's Guide. The IP is capable of all defined state transitions. It uses only a subset of the allowable data transfer lengths: operands of 1, 2, 4, 6, and 8 bytes are supported. Figure 4 illustrates the Processor Packet bus states for the iAPX 43203 and also conforms to the Packet bus definition.

Table 2. Processor Packet bus Group

Figure 4. Processor Packet Bus State Diagram for iAPX 43203

### System Error Group

**ALARM/** (Alarm, Input, low asserted)

The ALARM/ input signals the occurrence of an unusual system-wide condition such as power failure. ALARM/ is

iAPX 43203 PIN DESCRIPTION

107

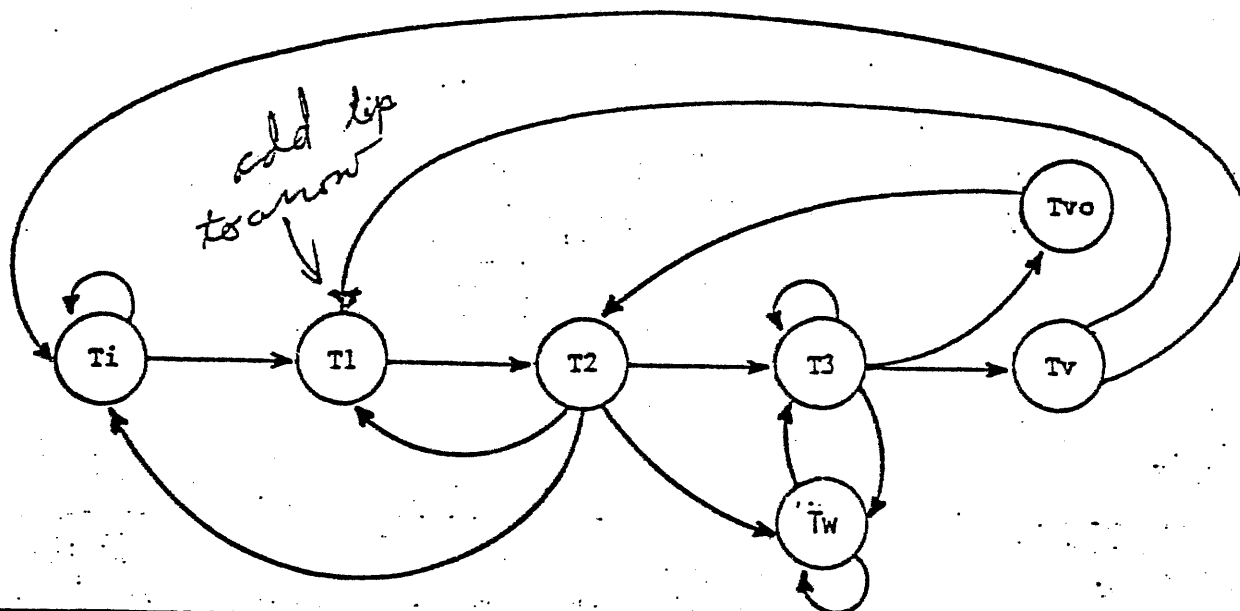
Processor Packet Bus Group

The following pins fully conform to the specification in the iAPX 432 Processor Packet Bus Definition of the iAPX 432 Component User's Guide. The IP is capable of all defined state transitions. The 43203 uses only a subset of the allowable data transfer lengths: operands of 1,2,4,6, and 8 bytes are supported.

PIN NAME	I/O	DESCRIPTION	
PRQ	0	<u>P</u> acket <u>R</u> equ <u>e</u> st	Asserted High
ICS	I	<u>I</u> nter <u>c</u> on <u>n</u> e <u>c</u> t <u>S</u> tatus	
BOUT	0	Enable External <u>B</u> uffers for <u>O</u> utput	Asserted High
ACD15...ACD0	I/O	<u>A</u> ddress, <u>C</u> ontrol, and <u>D</u> ata Bus	

TABLE 2 PACKET BUS SIGNALS

Processor Packet Bus State Diagram



<u>INITIAL STATE</u>	<u>NEXT STATE</u>	<u>TRIGGER</u>
Ti	T1	Bus cycle desired
	Ti	No bus cycle desired
T1	T2	Unconditional
T2	T3	ICS high
	Tw	ICS low
	T1	Cancelled, Access pending
	Ti	Access Cancelled, No Access pending
T3	T3	Additional transfer required
	Tw	ICS low
	Tv	All transfers completed if current cycle
	Tvo	overlapped write
Tv	Ti	is lead or if write but no pending write
	T1	Current write with pending write
	T2	Current write with overlapped write
Tvo	T2	Overlapped write
Tw	Tw	ICS low
	T3	ICS high

FIGURE 4 43203 PACKET BUS STATE  
~~Figure~~ STATE DIAGRAM

113B

sampled on the rising edge of CLKA.

**FATAL/** (Fatal, Output, low asserted)

FATAL/ is asserted by the IP under microcode control when the processor is unable to continue due to various error or fault conditions. Once FATAL/ is asserted, it can only be reset by assertion of INIT/ (no hardware error checking).

#### System Wide Group

**CLR/** (Clear, Input, Low asserted).

Clear Hardware Error. CLR/ is designed to be used in a system employing Hardware Error Detection. Assertion of CLR/ results in a microprogram trap which causes the IP to immediately terminate any bus transactions or internal operations which may be in progress at the time of the error and start a microsubroutine written to handle that situation. Once the service routine has been started, it cannot be interrupted by a second CLR/ assertion. Response to CLR/ can only be reenabled by assertion of the INIT/ pin. Normally, after a hardware error fault the microcode will execute its hardware error routine and then wait for an IPC which will cause it to reset the IP under microprogram



control.

After the CLR/ pin is asserted, the IP master and checkers will try to "sync up" to each other. The earliest time that the components can be assumed to be resynchronized again is at the beginning of the fourth cycle after CLR/ is asserted.

CLR/ is sampled by the IP on the rising edge of CLKA.

**PCLK/** (P Clock, Input, low asserted)

Assertion of PCLK/ for one cycle causes the internal timers in the IP to "tick" (process timer decrements and system timer increments). Assertion of PCLK/ for two or more cycles causes the system timer to be reset.

**INIT/** (Initize, Input, low asserted)

Assertion of INIT/ causes the internal state of the IP to be reset and start execution of the initialization microcode. INIT/ must be asserted for a minimum of 8 clock cycles. After the INIT/ pin is returned to its nonasserted state, IP microcode will initialize all of the internal registers and windows and will wait for a local IPC.

During the first two clock periods that the INIT/ pin is asserted the HERR/ pin will be sampled to determine whether this IP is to be a master or a slave processor. At this time, if it is a master, the IP will enable all of its hardware error detected signals so that they will be valid when INIT/ goes high.

### System Clock Group

**CLKA, CLKB** (Clock A, Clock B, Inputs)

CLKA provides the basic timing reference for the IP. CLKB follows CLKA by one quarter cycle and is used to assist internal timings.

### Hardware Detection Group

**HERR/** (Hardware Error Output, low asserted) (Open Drain Output) (Master Input, low asserted)

Assertion of the HERR/ pin by the IP indicates that the IP has encountered a hardware error, either as a checker checking a master or as a master checking itself. HERR/ is asserted the cycle following the internal detection of a hardware error except for pins AD<sub>15</sub>-AD<sub>0</sub> where it may be up to five clocks. Because

of the asynchronous nature of hardware error detection on the AD pins (see AD pin description), HERR/ may tend to be asynchronous. For this reason when using HERR/ it is recommended that this pin be synchronized externally.

After HERR/ has been asserted for a cycle it is released for the next cycle to allow an external pullup resistor to bring it high again. After that cycle, the processor may reassert HERR/. Upon assertion of HERR/ the chip select will become deselected.

While INIT/ is asserted HERR/ carries information regarding whether the IP is to perform hardware error detection on the peripheral subsystem side. If the pin is high, the IP will configure as a checker. If the pin is low, the IP will configure as a master. HERR/ must provide the master/checker information for at least two cycles preceding the rising edge of INIT/.

### Peripheral Subsystem Bus Group

AD<sub>15</sub>-AD<sub>0</sub> (Address/Data, Input/output)

These pins constitute a multiplexed address and data input/output bus. When the attached processor bus is

idle or during the first part of an access, these pins normally view the bus as an address. The address is asynchronously checked to see if it falls within (matches) any one of the five window address ranges. The address is latched on the falling edge of ALE thereby maintaining the state of a match or no match for the remainder of the access cycle. The addresses are then unlatched on the falling edge of OE.

Once SYNC has pulsed high, the AD<sub>15</sub>-AD<sub>0</sub> pins become data input and output pins. When WR/ is high (read mode), data is now accessed in the IP and the output buffers are enabled onto the AD pins if the OE is asserted. When WR/ is low (write mode), data is sampled by the IP after the rising edge of SYNC during the CLKA high time (refer to the discussion of programmable interface timing).

The address is always a 16-bit unsigned number. Data may be either 8 bits or 16 bits as defined by BHEN/ and AD<sub>0</sub>. 8-bit data may be transferred on either the high (AD<sub>15</sub>-AD<sub>8</sub>) or the low (AD<sub>7</sub>-AD<sub>0</sub>) byte. When 8-bit data is transferred on the high or low byte, the opposite byte is 3-stated. Twenty-bit addresses are accommodated by the external decoding of the additional address bits and incorporation in the external CS/ logic.

During the clock state in which write data is sampled, data must be set up before the rising edge of CLKA and must be held until the falling edge of that CLKA. Read data is driven out from a CLKA high and should be sampled on the next rising edge of CLKA.

Hardware error detection sampling is not done synchronously to CLKA. It is sampled by the falling edge of the OE pin. The internal AD pin hardware error detection signal is then clocked and output on the HERR/ pin. At this point it may still not be synchronous with CLKA and should be externally synchronized.

**BHEN/** (Byte High Enable, Input, low asserted)

This pin, together with AD<sub>0</sub>, determines whether 8 or 16 bits of data are to be accessed, and if it is 8 bits, whether it is to be accessed on the upper or lower byte position. This pin is latched by the falling edge of ALE and unlatched by the falling edge of OE. BHEN/ and AD<sub>0</sub> decode as follows:

BHEN/	AD <sub>0</sub>	DESCRIPTION
0	0	16-bit access
0	1	8 bits on upper byte,

		lower byte tristated
1	0	8 bits on lower byte, upper byte tristated
1	1	8 bits on lower byte, upper byte tristated

**CS/** (Chip Select, Input, Low Asserted)

Chip Select Specifies that this IP is selected and that a read or write cycle is requested. This pin is latched by the falling edge of ALE and unlatched by the falling edge of OE.

**WR/** (Write, Input, low asserted)

This pin specifies whether the access is to be a read or a write. **WR/** is asserted high for a read and asserted low for a write. This pin is latched by the falling edge of ALE and unlatched by the falling edge of OE.

#### PS Timing Group

**ALE** (Address Latch Enable, Input, rising- and falling-edge-triggered)

The rising edge of ALE sets a flip-flop which enables Transfer Acknowledge (XACK/) to become active. The falling edge of ALE latches the address on the AD<sub>15</sub>-AD<sub>0</sub> pins and latches WR/, BHEN/ and CS/.

**OE** (Data Output Enable, Input, high asserted)

During a read cycle the OE pin enables read data on to the AD<sub>15</sub>-AD<sub>0</sub> pins when it is asserted. During a read or write cycle the falling edge of OE signifies the end of the access cycle. Specifically, the falling edge of OE does three things:

1. Resets the XACK/ enable flip-flop, thereby terminating XACK/.
2. Terminates DEN/ (if read cycle).
3. Opens address latches WR/, BHEN/, and CS/.

**SYNC** (Synchronized Qualifier Signal, Input, high asserted)

A rising edge on this signal must be synchronized to the IP CLKA falling edge. This signal qualifies the address, BHEN/, CA/ and WR/ indicating a valid condition. SYNC also initiates any internal action on the IP's part to process an access. It starts the

request for data to the IP in a read access. In a write access, data is expected one or two CLKA's after SYNC pulses high. At initialization time, IP microcode sets the write sample delay. However, this can be modified to one clock cycle by making a function request to the IP to change the write sample delay.

When the hold/hold-acknowledge mechanism of the IP is used, and once HDA has pulsed high, a SYNC pulse is required to qualify the hold acknowledge since the HDA pin can be asynchronous.

#### PS Buffer Control Group

**DEN/** (Data Enable, Output, low asserted)

This pin enables external data buffers which would be used in systems where the address and data are not multiplexed as they are in a Multibus system. DEN/ assertion begins no sooner than the CLKA high time of the first clock of SYNC assertion if a valid, mappable address range is detected. It is terminated with the falling edge of OE. In a write access, it is also terminated after XACK assertion.



## PS Interlock Group

**HLD** (Hold Request, Output, high asserted)

The hold/hold-acknowledge mechanism is an interlocking mechanism between the peripheral subsystem and the IP. Hold is used by the IP to gain control of the subsystem bus to ensure that no subsystem processors will make an access to the IP while it alters internal registers.

This signal is put out synchronously with the rising edge of CLKA. Hardware error detection sampling occurs during CLKA low time.

In special cases it may not be necessary to use the HLD function interlocking. In this case HDA can be tied high and no SYNC pulse will be required for HDA qualification. The hardware detects this condition by noting that the HDA pin was high a half clock before HLD requests a hold. In this mode the HLD output still functions and can be monitored if desired.

**HDA** (Hold Acknowledge, Input, high asserted)

HDA is asserted by the peripheral subsystem when the IP's request for a hold has been granted. This pin need only be a high pulse and can be asynchronous to

CLKA. This pin must be followed by a SYNC pulse in order to synchronously qualify it.

### PS Synchronization Group

**XACK/** (Transfer Acknowledge, Output, low asserted)

XACK/ is used to acknowledge that a data transfer has taken place.

For random or local accesses, XACK/ indicates that a transfer to or from iAPX 432 memory has completed.

For buffered accesses where the XACK-Delay is not in the advanced mode, XACK/ signifies that the transfer from/to the prefetch/postwrite buffer in the IP has been completed.

For buffered accesses which use advanced acknowledge mode (XD=0) the formation of an advanced XACK/ signal is requested. This allows the possibility of interfacing to the peripheral subsystem without wait states. The acknowledge will be advanced if the access is a read operation and the buffer contains the required data or the access is a write operation and the buffer contains sufficient space to accept the write data. In addition, the access must be valid.

If XACK/ is preceded by a low pulse on NAK/, then XACK/ signifies that the access encountered a fault. If the access was a random access, other than window #4, the window will be placed in the faulted state and any further accesses to this window will be ignored by the IP.

If the IP is programmed to be in advanced acknowledge mode (XD=0) and XACK/ is not returned before the peripheral subsystem issued SYNC, then XACK/ will be postponed until valid data has been established on the AD<sub>15</sub>-AD<sub>0</sub> bus.

Five conditions affecting XACK/ behavior:

1. XACK-Delay, user programmable through an IP function request. This parameter establishes the minimum operating XACK-delay with respect to the SYNC signal.
2. XACK-enable-flip-flop, set by the rising edge of the ALE signal and reset by the falling edge of the OE signal.
3. Internal IP Registers. These are used to determine validity of the peripheral subsystem access and establish access modes.

4. Type of access behavior: Random, Local or Buffered.
5. Bus Faults, non existant memory, etc.

Hardware error detection occurs during the first clock of SYNC assertion.

**NAK/** Negative Acknowledge, Output, low asserted)

This signal precedes XACK/ by one half clock cycle in order to qualify it as a negative acknowledge. This pin pulses low for only one clock period.

When the IP is in physical mode and making a local access, the use of negative acknowledge may be used to indicate that the access was made to nonexistent local address space. This will allow determination of the system configuration by a subsystem processor at system initialization time.

This pin could be used to set a status bit and cause a special interrupt to transmit the information back to the subsystem.

This signal is synchronously driven from the falling edge of CLKA. Hardware Error Detection occurs during

CLKA high.

**INHI** (Inhibit, Output, high asserted)

This pin is asynchronously asserted by non-clocked logic when a valid mappable address range is detected. It can be used to override other memories in the peripheral subsystem whose address space is overlapped by an IP window. After initialization, the microcode sets the INHI mode for each window by loading registers in the IP for each window. Once the subsystem is allowed to make a function request, it can selectively disable or enable the inhibit mode on each window. This pin is gated off by CS/.

The selection of the inhibit mode for window 0, when in buffered mode, causes a corresponding built-in XACK-delay which delays the acknowledge from going active until two clock periods after the rising edge of SYNC. This was done to facilitate most Multibus systems that use INHI which require that the acknowledge be delayed. When the Advanced XACK/ mode is programmed, the inhibit mode should not be used on window 0 when in buffered mode, since the acknowledge will not be effectively delayed.

Hardware error detection occurs during the first clock of SYNC assertion.

## PS Interrupt Group

**INT** (Interrupt, Output, high asserted)

This output is a pulse 2 CLKA's wide, and is synchronously driven from the rising edge of CLKA. Hardware error detection occurs during CLKA low.

## PS Reset Group

**PSR** (Peripheral Subsystem Reset, Output, high asserted)

PSR is asserted by the IP under microprogram control. When asserted, the peripheral subsystem should be reset. In a debug type of control, it may be desirable to use this pin to set a status bit in an external register or possibly cause a special interrupt. This pin is normally asserted by the IP when the peripheral subsystem is believed to be faulty and would not respond to other means of control.

This signal is put out synchronously with the rising edge of CLKA. Hardware error detection sampling access during CLKA low time.

Table 3. 43203 Electrical Characteristics

Table 4. 43203 D. C. Characteristics

Table 5. 43203 A. C. Characteristics

Figure 5. Timing Diagram for ACD Parameters

Figure 6. Timing Diagram for Local Processor Bus Timing

Figure 7. Timing Diagram for Multibus Interface Timing

#### **Software Programmable Interface Timing**

To accommodate a wide variety of PS interfaces, there are two programmable IP timing parameters: XACK-delay and write sample delay. These parameters are located in a data structure in iAPX 432 system memory that is accessible to the IP via the function request facility.

XACK-delay is a two-bit quantity that specifies the minimum delay before XACK/ is signalled on a transfer. The minimum delay can only be attained with buffered accesses. Figure XT displays the representation of the XACK-delay codes.

Table 6. XACK/ Timing Parameters

## 43203 Electrical Characteristics

### Absolute Maximum Ratings

Ambient Temperature Under Bias	0° C to 70° C
Storage Temperature	-65° C to +150° C
Voltage on Any Pin with respect to GND	-1 V to +7 V
Power Dissipation	2.5 Watts

TABLE 3. 43203 ABSOLUTE MAXIMUM RATINGS



iAPX 43203 D.C. CHARACTERISTICS

VCC=5V+10%

Ta=0°C to 70°C

SPEC	DESCRIPTION	MIN	MAX	UNITS
Vilc	Clock Input Low Voltage	-0.3	+0.5	V
Vihc	Clock Input High Voltage	3.5	VCC+0.5	V
Vil	Input Low Voltage	-0.3	0.8	V
Vih	Input High Voltage	2	VCC+0.5	V
Icc	Power Supply Current	-	450	mA
Iil	Input Leakage Current	-	+10	uA
Io	Output Leakage Current	-	+10	uA
Iol	@0.45 Vol			
	HERR/	-	8	mA
	FATAL/	-	4	mA
	AD15...ADO	-	4	mA
	OTHER	-	2	mA
Ioh	@2.4V	-	-0.1	mA

TABLE 4. DC CHARACTERISTICS

iAPX 43203 A.C. CHARACTERISTICS

VCC = 5 ± 10% Ta = 00C to 700C

Loading: AD15...ADO 20 to 100pf  
OTHER 20 to 70pf

SYMBOL	DESCRIPTION	8 MHz.		5 MHz.		UNIT
		MIN	MAX	MIN	MAX	
<u>GLOBAL TIMING REQUIREMENTS</u>						
t <sub>cy</sub>	Clock Cycle Time	125	1000	200	1000	nsec.
t <sub>r</sub> ,t <sub>f</sub>	Clock Rise and Fall Time	-	10	-	10	nsec.
t <sub>1</sub> ,t <sub>2</sub>						
t <sub>3</sub> ,t <sub>4</sub>	Clock Pulse Widths	26	250	45	250	nsec.
t <sub>is</sub>	Signal to INIT/ Hold Time	10	-	10	-	nsec.
t <sub>ie</sub>	INIT/ Enable Time	8	-	8	-	t <sub>cy</sub>
<u>SYSTEM SIDE TIMING REQUIREMENTS</u>						
t <sub>dc</sub>	Signal to CLOCK Setup Time	5	-	5	-	nsec.
t <sub>cd</sub>	Clock to Signal Delay Time	-	55	-	75	nsec.
t <sub>dh</sub>	Clock to Signal Hold Time	25	-	35	-	nsec.
t <sub>si</sub>	Signal to INIT/ Setup Time	15	-	25	-	nsec.
<u>PERIPHERAL SUBSYSTEM SIDE TIMING REQUIREMENTS</u>						
t <sub>as</sub>	AD15...ADO,CS/,WR/,BHEN/ Setup Time to ALE Low	0	-	0	-	nsec.
t <sub>ah</sub>	AD15...ADO,CS/,WR/,BHEN/ Hold Time to ALE Low	32	-	35	-	nsec.
t <sub>ss</sub>	SYNC High Setup Time to CLKA High	50	-	60	-	nsec.
t <sub>sh</sub>	SYNC Low Hold Time to CLKA High	30	-	40	-	nsec.
t <sub>sw</sub>	SYNC High Pulse Width	50	2 t <sub>cy</sub>	60	2 t <sub>cy</sub>	nsec.
t <sub>ds</sub>	Write Data Setup to Sampling CLKA High	10	-	20	-	nsec.
t <sub>dh</sub>	Write Data Hold to Sampling CLKA Low (Advanced XACK/)	10	-	20	-	nsec.
t <sub>dhx</sub>	Write Data Hold to XACK/	5	-	5	-	nsec.
t <sub>asy</sub>	AD15...ADO,CS/,WR/,BHEN/ Setup to SYNC	120	-	160	-	nsec.
t <sub>sdh</sub>	CLKA High to HLD,INT,PSR	-	75	-	90	nsec.

TABLE 5(a) AC CHARACTERISTICS

SYMBOL	DESCRIPTION	8 MHz.		5 MHz.		UNIT
		MIN	MAX	MIN	MAX	
<u>PERIPHERAL SUBSYSTEM TIMING RESPONSES</u>						
tsdh	CLKA High to HLD,INT,PSR	-	75	-	90	nsec.
taih	Valid AD15...ADO,CS/ to Chip INH1 Valid Delay	-	80	-	85	nsec.
tede	OE to DEN/ Delay	-	65	-	70	nsec.
tead	OE to Enable AD15...ADO Buffers Delay (Read Cycle)	-	70	-	75	nsec.
tdad	OE to Disable AD15...ADO Buffers Delay (Read Cycle)	-	52	-	55	nsec.
tdsx	AD15...ADO Read Data Valid Setup to XACK/ Active (Non-advanced XACK/)	20	-	20	-	nsec.
tced	CLKA High to Enable AD15...ADO Buffers Delay	-	70	-	75	nsec.
tcvd	CLKA High to Valid Read Data Delay	-	80	-	90	nsec.
tox	OE Inactive to XACK/ Inactive Delay	-	80	-	90	nsec.
tdds	AD15...ADO Disable Setup to DEN/ High	0	-	0	-	nsec.
txde	XACK/ Low to DEN/ High (Write Cycle)	-	35	-	40	nsec.
tcde	CLKA High to DEN/ Low	-	70	-	75	nsec.
<u>XACK/ TIMING CHARACTERISTICS</u>						
tax	Buffered Accesses with XD=0 ALE High to XACK/ Valid	0	65	0	70	nsec.
tdsx	AD15...ADO Read Data Valid Setup to XACK/ Valid (When internal state does not allow XACK/ before SYNC)	20	-	20	-	nsec.
tadx	Valid AD15...ADO to XACK/ Valid (when internal state allows XACK/ before SYNC)	-	120	-	140	nsec.
tdsx	Buffered Accesses (With XD=1 or XD=2) or Random Accesses AD15...ADO Read Data Valid Setup to XACK/	20	-	20	-	nsec.
tsd1	Faulted Accesses CLKA Low to NAK/	-	75	-	90	nsec.
tsnx	Setup of NAK/ to XACK/	50	-	50	-	nsec.

Note: All timing parameters are measured at the 1.5 Volt level except for CLKA and CLKB which are measured at the 1.8 Volt level.

TABLE 5(B) AC CHARACTERISTICS

129D

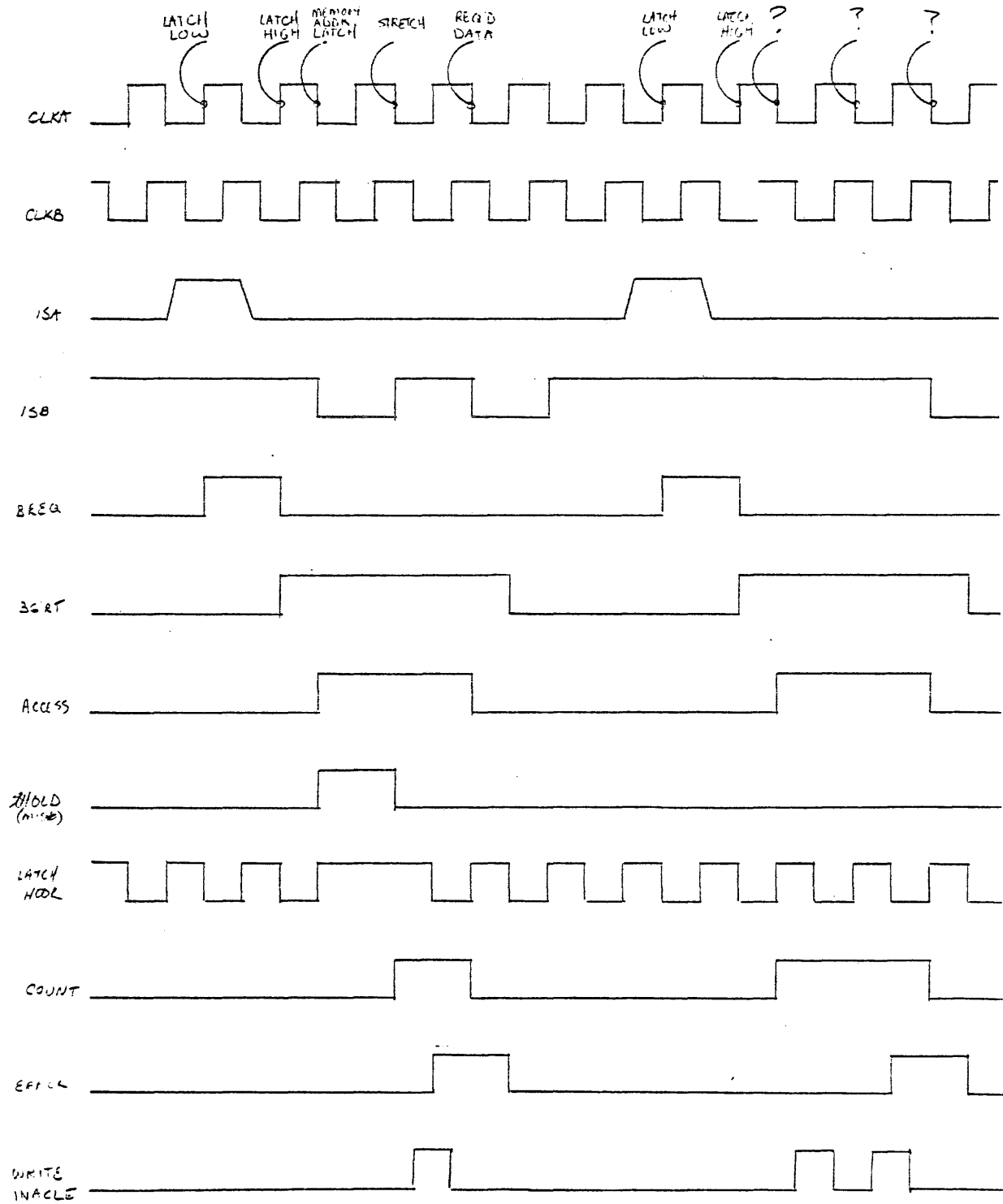
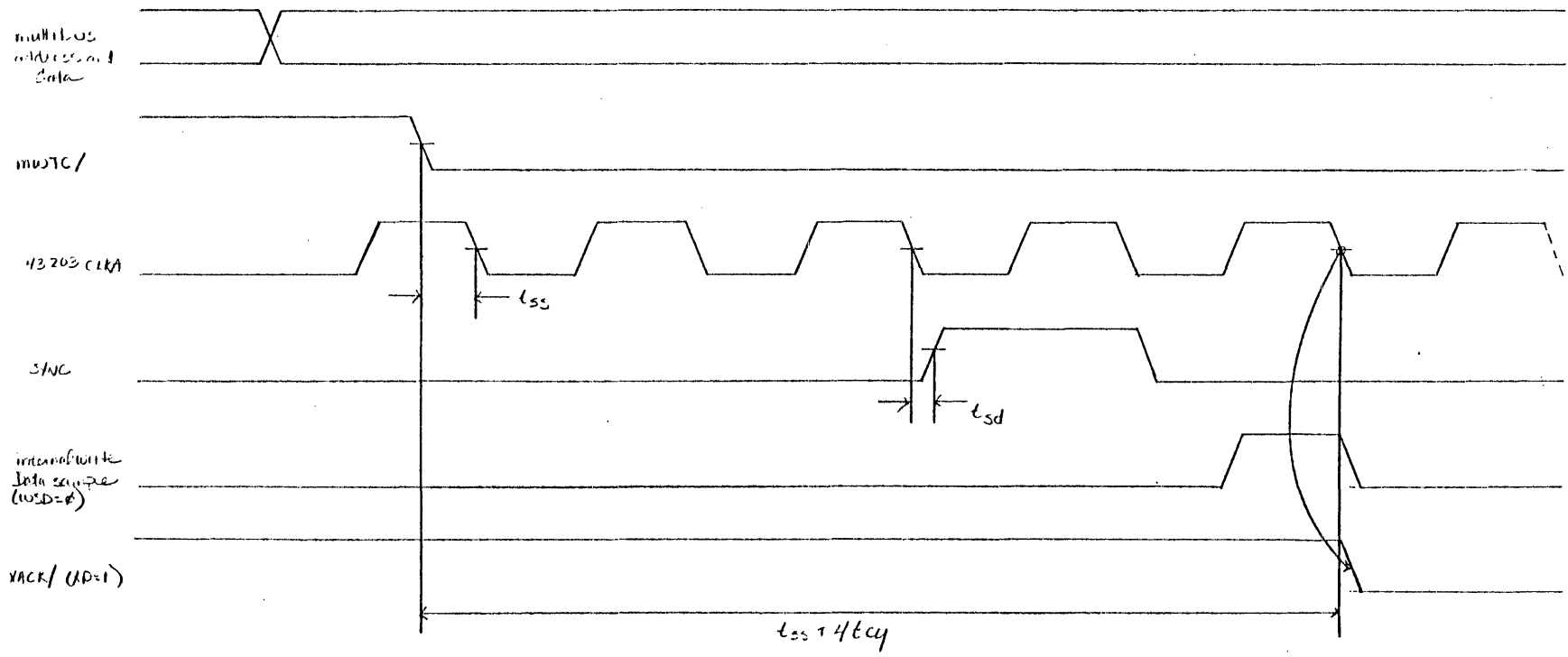


FIGURE 5 ~~FIGURE TIMING~~ <sup>ACD</sup> - TIMING DIAGRAM





129G

Multibus write cycle minimum access time  
 (buffered 43203 access with space in buffer for write data)

FIGURE 7 MULTIBUS INTERFACE TIMING

Inhibit Mode	WR/	XD <sub>1</sub>	XD <sub>0</sub>	XACK/ Formation
0	X	0	0	Advanced Acknowledge (XACK/ can occur before SYNC)
0	1	0	1	Rising edge of SYNC
0	0	0	1	Rising edge of SYNC plus 1 Clock
0	1	1	0	Rising edge of SYNC plus 1 Clock
0	0	1	0	Rising edge of SYNC plus 2 Clocks
1	X	1	0	Rising edge of SYNC plus 2 Clocks
1	X	0	1	Rising edge of SYNC plus 2 Clocks
X	X	1	1	Illegal condition

Note: X=don't care condition

**TAB: 6.** ~~Figure 4T~~ - XACK/ Timing Parameters

Write sample delay is a one-bit quantity that specifies the position of the internal write data sampling pulse with respect to the SYNC pulse. If WSD=0, write data is sampled one clock period after SYNC is asserted. If WSD=1, write data is sampled two clock periods after SYNC is asserted.

When initialized, the IP operates with the slowest interface timing (XD=10B, Write Sample Delay = 1B).

#### Hardware Programmable Interface Timing

In addition to the software programmable interface timing, the design of hardware external to the IP may control the delay in formation of XACK/.

Noting that the rising edge of ALE sets the XACK/ enable flip-flop, ALE (style 2 in Figure 8.) may be used to postpone the generation of XACK/. The falling edge of ALE in both styles latches AD<sub>15</sub>-AD<sub>0</sub>, CS/, WR/, and BHEN/.

Figure 8. Two Styles of ALE

#### Buffered Accesses

Window 0 has the special ability to be used in buffered



130A

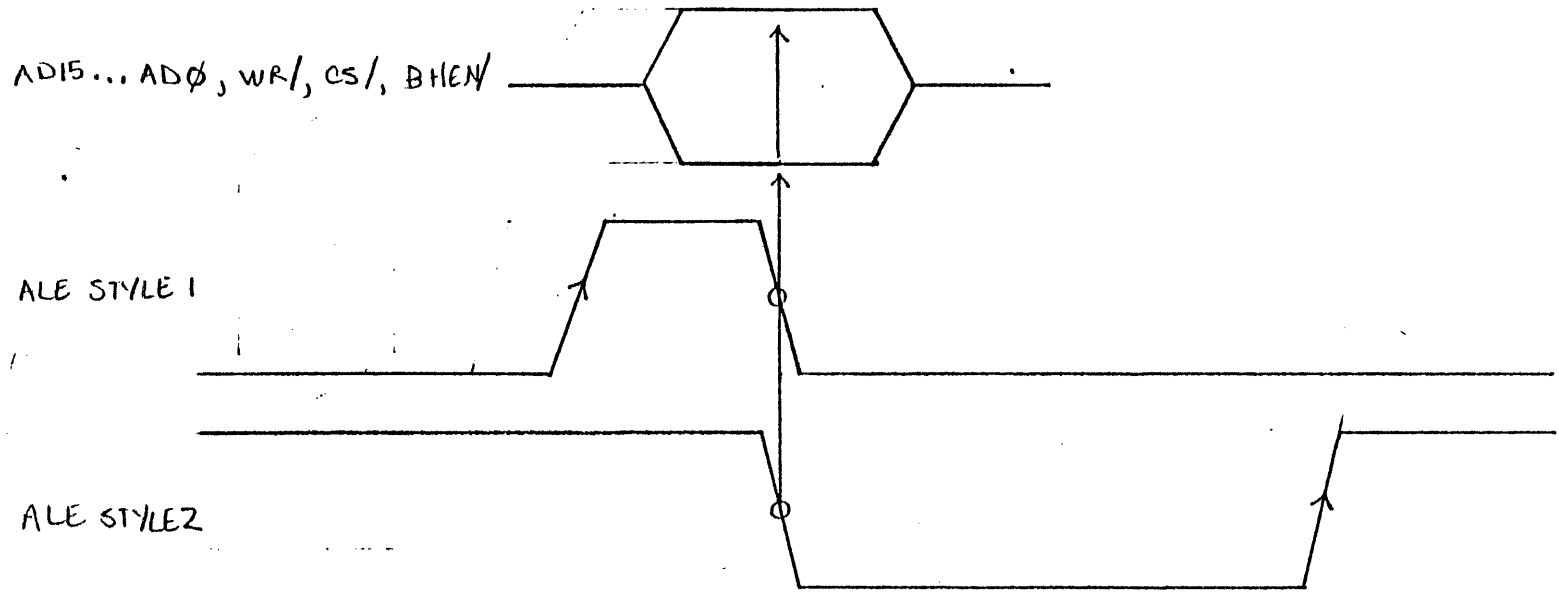
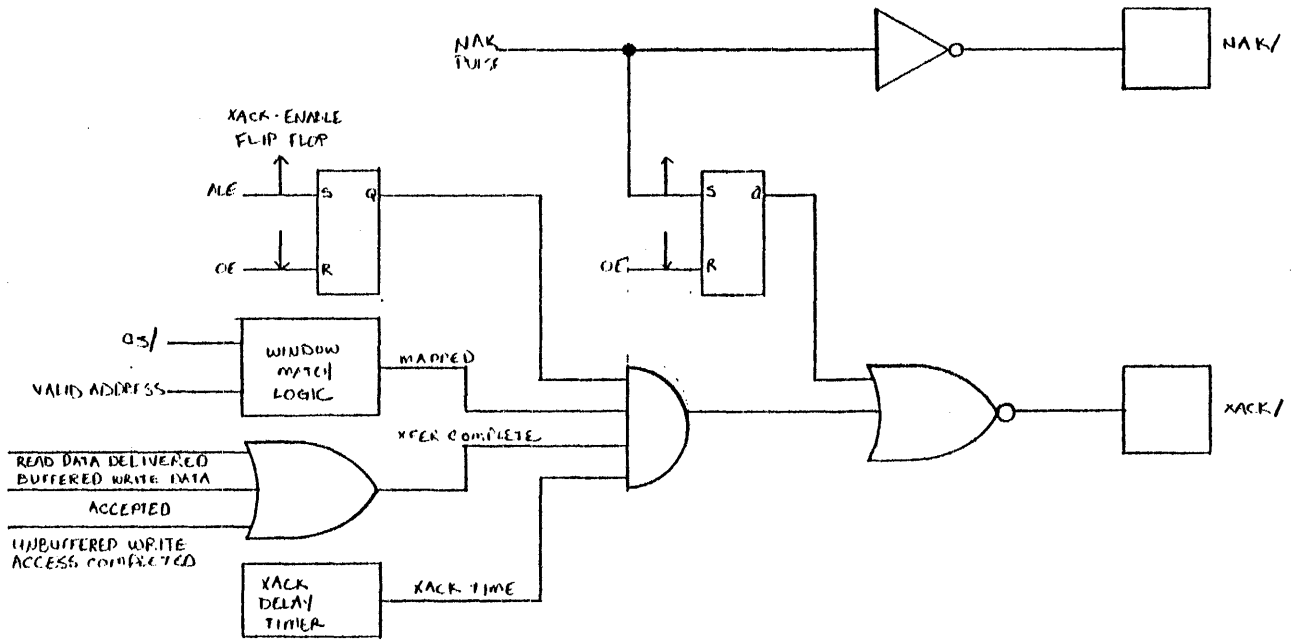


FIG ~~ALE~~<sup>130A</sup> - TWO STYLES OF ALE

27

130B



table

FIGURE 9 ~~Timing of Proprietary XACK Interface Timing~~

access mode. High speed data transfers to or from sequential locations in an iAPX 432 data segment are expedited by the IP through the use of an eight-double-byte (16-byte) FIFO. When an attached processor acquires data from an iAPX 432 data segment, the IP prefetches data from iAPX 432 memory. When an attached processor transfers data to an iAPX 432 system, the IP aggregates data in the FIFO before it postwrites into iAPX 432 system memory. Transfers into or out of the iAPX 432 system are performed in the largest size data packet as possible. The IP has the capability to form 8-, 6-, 4-, 2- and 1-byte transfer requests. An IP will transfer smaller sized packets when necessary to complete a transfer that is not an even multiple of 8 bytes in length.

Since data transferred on a processor Packet bus must always be right-justified, an IP performs byte packing or unpacking when data is moved. Read data from the iAPX 432 memory is acquired in 8-byte packets. The attached processor may use an 8- or 16-bit bus interface to the IP. Tables 7 and 8 display the FIFO action for transfers of data from an attached processor to iAPX 432 memory via the IP.

Table 7. 8-bit Processor Interface

Table 8. 16-bit Processor Interface

Hardware Error Detection with the 43203

BHEN/	A0	AD7..A0	Time	
1	0	Byte 6	6	
1	1	Byte 5	5	
1	0	Byte 4	4	
1	1	Byte 3	3	AP Bus
1	0	Byte 2	2	Transfers
1	1	Byte 1	1	
1	0	Byte 0	0	

XXXX	XXXX
XXXX	XXXX
XXXX	XXXX
XXXX	XXXX
XXXX	Byte 6
Byte 5	Byte 4
Byte 3	Byte 2
Byte 1	Byte 0

8 Double Byte FIFO

XXXX	Byte 6
Byte 5	Byte 4
Byte 3	Byte 2
Byte 1	Byte 0

Data Fields of  
Processor  
Bus Packets

XXXX- Undefined

TABLE 7 ~~Fig. 8~~ - 8 bit processor interface

BHEN/	A0	AD15	AD8	AD7	AD0	Time	
0	0	Byte 8		Byte 7		5	
0	0	Byte 6		Byte 5		4	
1	0	Hi-Z		Byte 4		3	AP Bus
0	1	Byte 3		Hi-Z		2	Transfers
0	0	Byte 2		Byte 1		1	
1	0	Hi-Z		Byte 0		0	

XXXX	XXXX	
XXXX	XXXX	
XXXX	XXXX	
XXXX	Byte 8	8 Double Byte FIFO
Byte 7	Byte 6	
Byte 5	Byte 4	
Byte 3	Byte 2	
Byte 1	Byte 0	

XXXX	Byte 8	Data Fields of Processor Bus Packets
Byte 7	Byte 6	
Byte 5	Byte 4	
Byte 3	Byte 2	
Byte 1	Byte 0	

XXXX - Undefined

TABLE 8 ~~Figure 16~~ - 16 bit processor interface

The 43203 presents an additional challenge in the area of hardware error detection, since two separate processor interfaces are supported: the iAPX 432 system and the peripheral subsystem.

When INIT/ is asserted, the FATAL/ and HERR/ pins of the 43203 are examined to establish the mode that each interface is to enter.

#### Representation of MASTER/CHECKER mode at initialization

FATAL/	HERR/	iAPX 432 Side	Peripheral Subsystem Side
0	0	MASTER	MASTER
0	1	MASTER	CHECKER
1	0	CHECKER	MASTER
1	1	CHECKER	CHECKER

Logic external to the 43203 must provide these signals.

#### Peripheral Subsystem Interface Timing

iAPX 432 systems are synchronous digital systems. The peripheral subsystem(s) employed with an iAPX 432 system

need not share the common (CLKA, CLKB) time base. Rather, the PS may operate at an independent frequency, allowing the system designer to tailor the cost/performance ratio of the PS to product needs.

The asynchronism of the PS to the iAPX 432 is resolved by the IP signal SYNC. A synchronizer external to the IP is used to generate SYNC, allowing many forms of peripheral subsystem to be attached. Two examples of interfaces to standard Intel peripheral subsystems are described in this section (Refer to Figures 10. and 11):

#### Interfacing 8086 Component Bus to the IP

#### Interfacing the Multibus to the IP

Timing calculations are included to show interface design and performance.

#### Interfacing the 8086 Component Bus to the IP

The following diagrams and calculations illustrate the design considerations in interfacing the 8086 component bus to the IP. Timing calculations are shown for both read and write accesses. The read access example assumes that the IP is operating in buffered mode and the buffer contains the required data. The calculations shown allow 8086 operation without wait states.

Table 9.

Maximum Mode 8086 System

8086 Write Data Setup Performance =  $-tcclh + tclcl + tcldv$

$$-15 + 250 + 110 = 345 \text{ ns; } 4 \text{ MHz. } 8086$$

$$-15 + 200 + 110 = 295 \text{ ns; } 5 \text{ MHz. } 8086$$

43203 Write Data Setup Requirements =  $tss + 2tcy + 0.5tcy + tds$

$$8 + 2(200) + 0.5(200) + 20 = 528 \text{ ns; } 5 \text{ MHz. } 43203$$

$$8 + 2(125) + 0.5(125) + 10 = 331 \text{ ns; } 8 \text{ MHz. } 43203$$

8086 Write Data Hold Performance

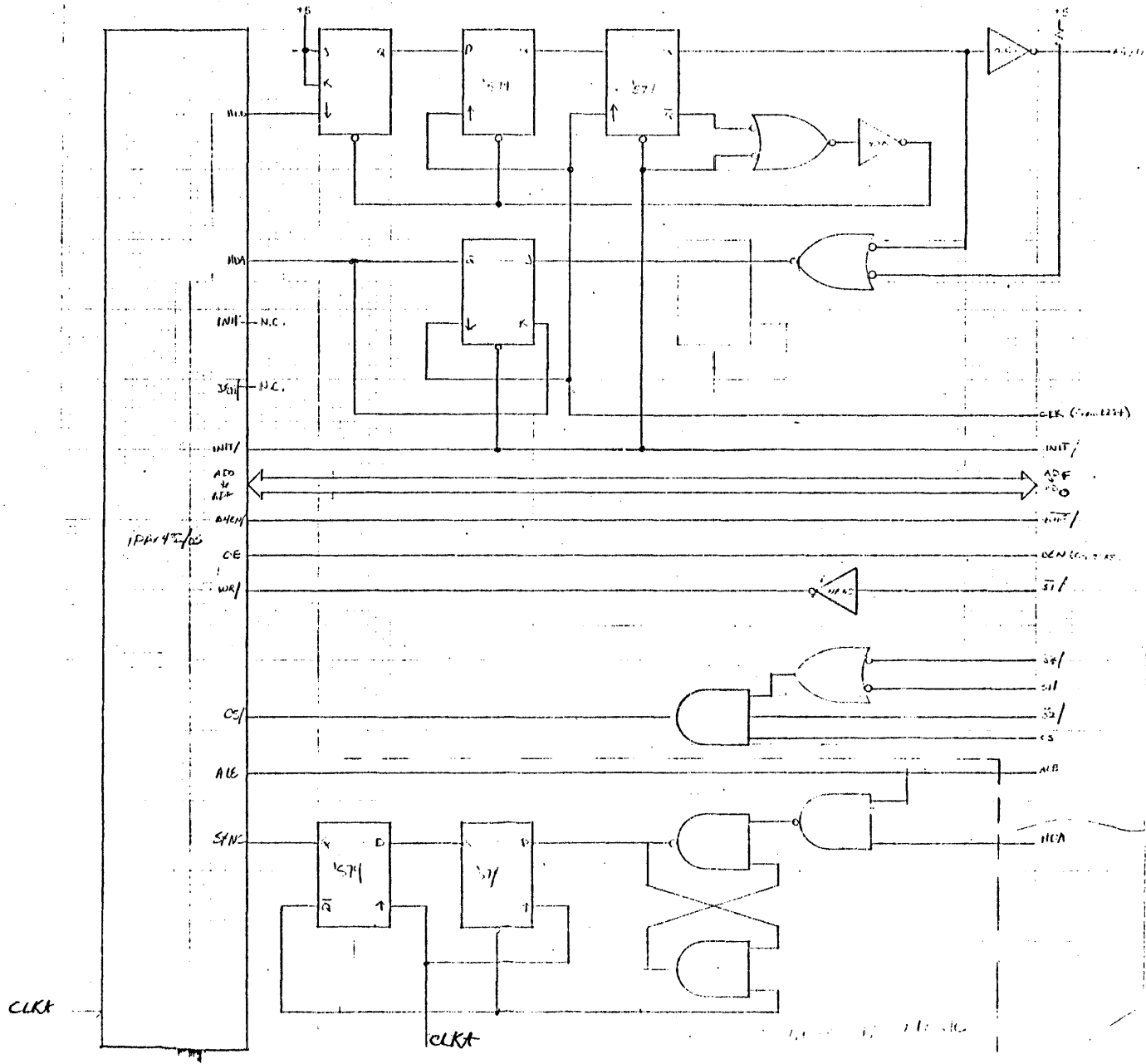
$$= -tcllh + 3tclcl + tclch + tchdx$$

$$-15 + 3(250) + 151 + 10 = 896 \text{ ns; } 4 \text{ MHz. } 8086$$

$$-15 + 3(200) + 118 + 10 = 713 \text{ ns; } 5 \text{ MHz. } 8086$$



FIGURE 10 MAXIMUM MODE INTERFACE (PO86)



133A

43203 Write Data Hold Requirements =  $-t_{ss} + 4t_{cy} + t_{dh}$

$$-8 + 4(200) + 20 = 812 \text{ ns; } 5 \text{ MHz. } 43203$$

$$-8 + 4(125) + 10 = 502 \text{ ns; } 8 \text{ MHz. } 43203$$

8086 Read Data Setup Requirements =  $3t_{clcl} - t_{cllh} - t_{dvcl}$

$$3(250) - 15 - 30 = 705 \text{ ns; } 4 \text{ MHz } 8086$$

$$3(200) - 15 - 30 = 555 \text{ ns; } 5 \text{ MHz } 8086$$

43203 Read Data Setup Performance

$$= t_{ss} + 2t_{cy} + t_{sd} + 0.5t_{cy} + t_{cvd}$$

$$8 + 2(200) + 8 + 0.5(200) + 90 = 516 \text{ ns; } 5 \text{ MHz. } 43203$$

$$8 + 2(125) + 8 + 0.5(125) + 80 = 331 \text{ ns; } 8 \text{ MHz. } 43203$$

### Multibus Interfacing

Table 10 demonstrates the interface of an IP to an Intel

Multibus peripheral subsystem. Calculations are included for minimum Multibus access time.

Table 10. Multibus Interface to the IP Calculations

Table 11. IP/GDP Operator Comparison Table

(where is this table?)

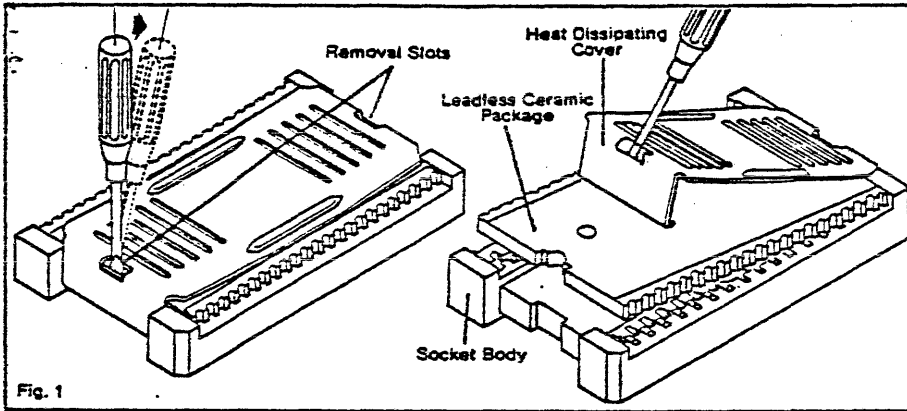


Fig. 1

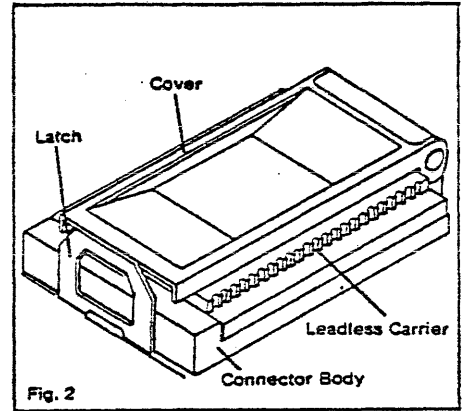


Fig. 2

## Convenient ZIF Socket

Rapid interconnection to circuitry is accomplished with a compact 64 lead zero-insertion-force connector. Gold tipped wiping leaf contacts insure a clean, gas tight interface with the ceramic carrier. The connector insulating material is the same durable glass-filled thermoplastic proven in our "Scotchflex" prod-

uct line and is flammability rated at 94 V-O. Solder pins are located on standard .100" centers making it compatible with existing printed circuit board design guidelines and standard assembly equipment. A "snap-in" heat dissipating cover holds the chip carrier in place. An ordinary screwdriver permits quick removal. See figure 1.

## Complete Burn-in Capability

The 3M QUIP System includes burn-in connector. This connector facilitates high volume production testing at a minimum cost. The durable contacts and body ensure reliable testing up to 200°C. The heat dissipating cover features a positive locking quick release latch. See figure 2.

### TYPICAL PROPERTIES

<b>PHYSICAL</b>	
CERAMIC PACKAGE	Ceramic: 94% Al <sub>2</sub> O <sub>3</sub> —Black
3M PART NO. 3534 SOCKET	Metalization: Gold (60μ in. min.) over nickel and refractory.
3M PART NO. 3362 SOCKET (BURN-IN)	Cover: copper alloy. Body: glass filled polyester. Contacts: copper alloy 725 with .000030" gold over nickel interface.
	Latch: stainless steel. Cover: copper alloy. Contacts: gold plated (.000010 in.) BeNi. Body: polyphenylene sulfide (Ryton R4)
<b>THERMAL</b>	
64 LEAD QUIP SYSTEM $\theta_{JA}$ :	50°C/watt maximum thermal resistance
<b>ELECTRICAL</b>	
64 LEAD QUIP SYSTEM	Maximum resistance: .500Ω. Maximum interlead capacitance: 5 pfd. Dielectric withstanding voltage: 1000 volts at sea level. Current rating: 1 amp. per lead, limited to 30°C rise per lead.
<b>ENVIRONMENTAL</b>	
64 LEAD QUIP SYSTEM	Temperature rating: -55°C to 105°C.
3M PART NO. 3362 SOCKET (BURN-IN)	Temperature rating: -55°C to 200°C.

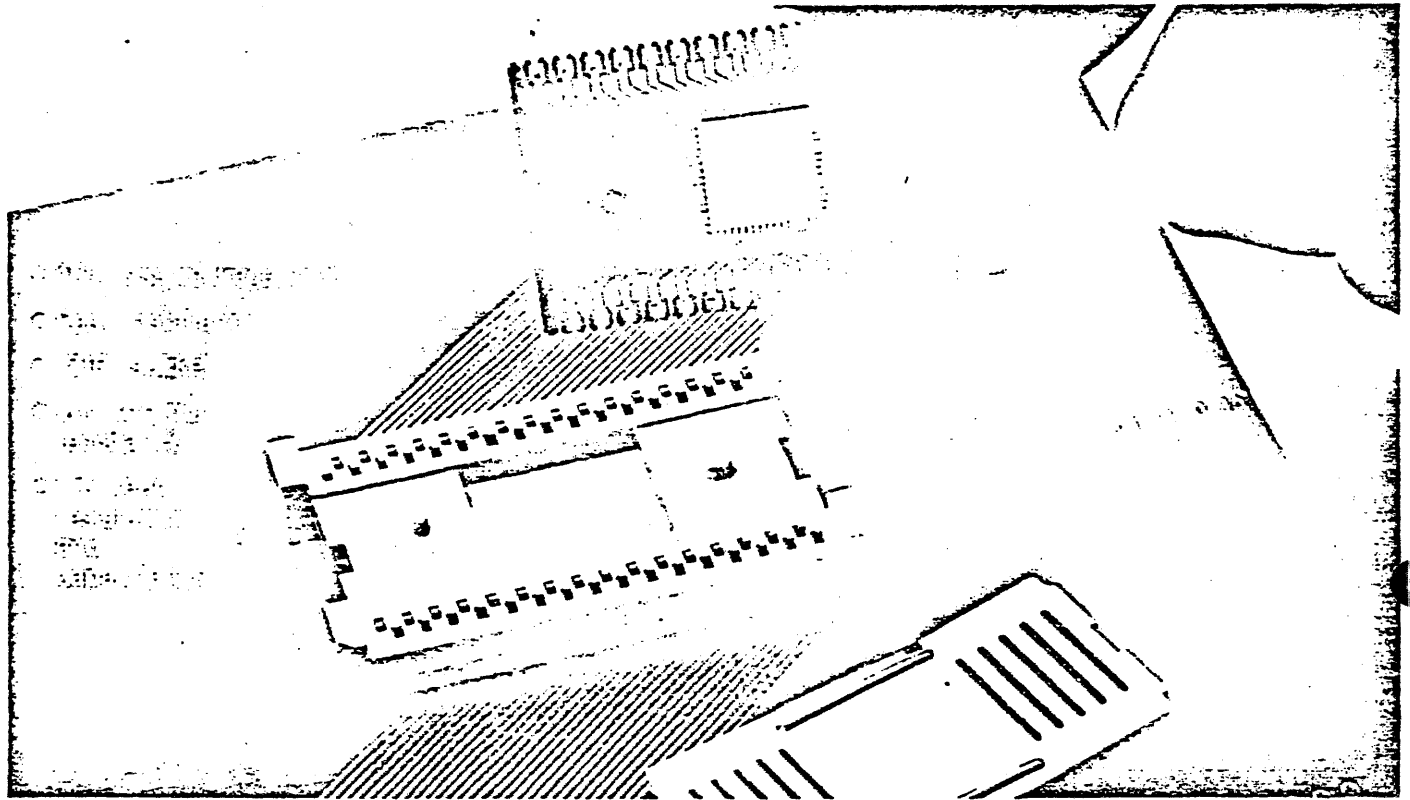
\*Ryton is a registered trademark of Phillips Chemical Co.

Electronic Products Division/3M  
3M CENTER • SAINT PAUL, MINNESOTA 55101

# New Product Bulletin

# 3M

## 3M Brand 64 Lead Quip System



### A Complete Quip System From The Source

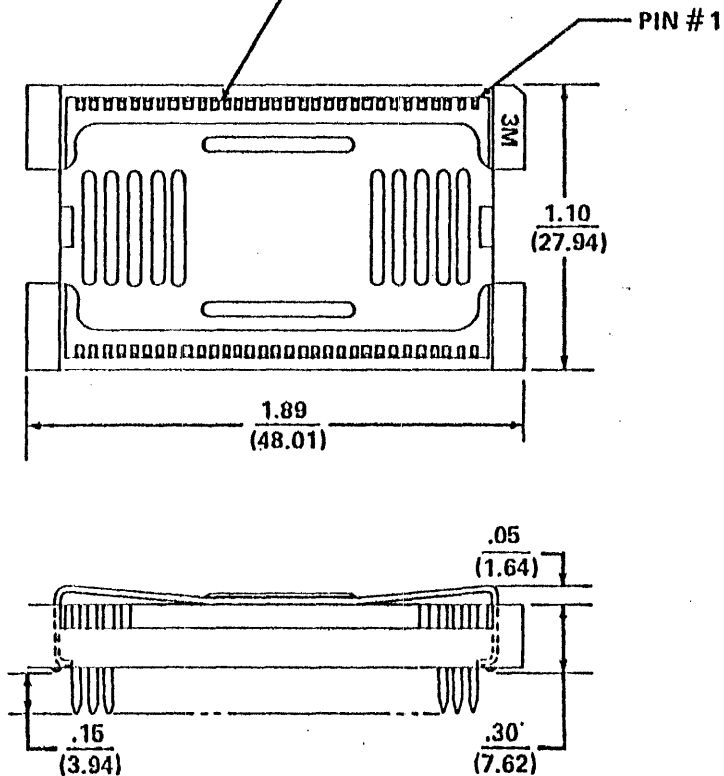
The 3M Brand Quad-In-Line package system brings built-in simplicity and lower total costs to microprocessor packaging. Its smaller size and low-profile cut package area by one-third, allowing for greater board density than with conventional 64 lead dual-in-line configurations. Shorter trace lengths result in lower lead resistance and

capacitance. Faster switching times and improved system performance can be achieved. A carrier to connector polarizing feature eliminates costly assembly errors.

### Reliable Ceramic Package

The heart of the 3M QUIP System is a reliable, co-fired multilayer leadless package. Its design permits easy replacement and field repair. Costly brazing and metallized leads are eliminated. The rugged, optically opaque ceramic structure is chemically inert, dimensionally stable and thermally conductive. A low thermal resistance makes possible the use of IC's with greater power requirements, increasing overall circuit performance.

**LEADLESS QUAD IN-LINE CERAMIC PACKAGE**



**CONNECTOR DETAIL**

**NOTES:**

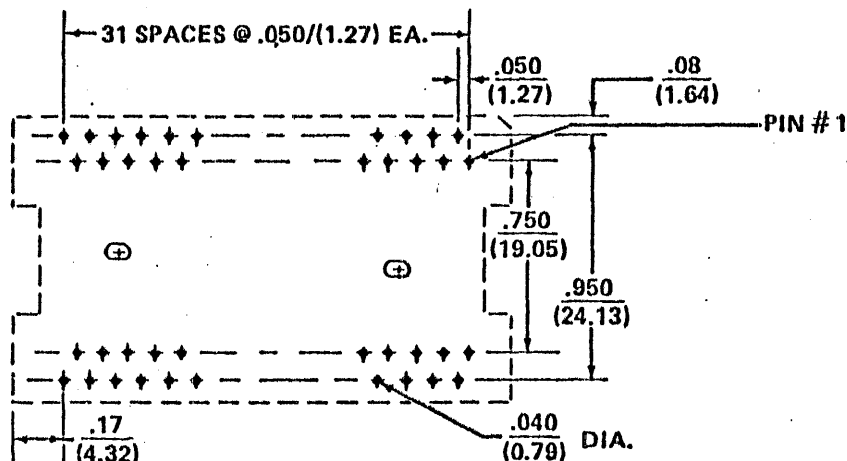
**SCOPE:** THIS SPECIFICATION DETAILS THE REQUIREMENTS FOR A MULTIPOSITION LEADLESS CERAMIC PACKAGE SOCKET

**SPECIFICATIONS -**

**PHYSICAL:** INSULATOR MATERIAL - GLASS REINFORCED GRAY THERMO PLASTIC U.L. FLAMMABILITY RATING 94V-0  
 CONTACT BASE METAL - COPPER-NICKEL-TIN ALLOY 725  
 CONTACT INTERFACE - 0.000030" (1.762 um) MIN. GOLD OVER 0.000050" (1.27 um) MIN. NICKEL

**ENVIRONMENTAL:**

TEMPERATURE RATING - -67°F to +221°F (-55°C to +105°C)



**HOLE PATTERN DETAIL**

136C

**USED ON**

**IMPORTANT NOTICE TO PURCHASER:** All statements, technical information and recommendations contained herein are based on tests we believe to be reliable, but the accuracy or completeness thereof is not guaranteed, and the following is made in lieu of all warranties, express or implied.

Seller's and manufacturer's only obligation shall be to replace such quantity of the product proved to be defective. Neither seller nor manufacturer shall be liable for any injury, loss or damage, direct or consequential, arising out of the use of or the inability to use the product. Before using, user shall determine the suitability of the product for his intended use, and user assumes all risk and liability whatsoever in connection therewith. No statement or recommendation not contained herein shall have any force or effect until in an agreement signed by officers of seller and manufacturer.



ELECTRONIC PRODUCTS

St. Paul  
Minnesota

ISSUE	ISSUE DATE AND CHANGE RECORD	REV.	CH.
-------	------------------------------	------	-----

**3534 QUIP SOCKET SPECIFICATION**

64 LEADS

SK-77-073

TOLERANCE UNLESS NOTED			
	.0	.00	.000
INCH	± .1	± .02	± .005
mm	± .5	± .13	///

INCH (mm)	DR.
SCALE	APP.

SCALE	A
-------	---

